

Introdução à Teoria do NP-Completo

1- Introdução

Uma vez calculada a ordem de complexidade de um algoritmo, como reconhecer se o mesmo é ou não **eficiente**? Vamos definir o seguinte: *um algoritmo é eficiente quando a sua complexidade de pior caso for um polinômio no tamanho da entrada.*

Assim, algoritmos de complexidade de pior caso $O(1)$, $O(n)$, $O(n^2)$, $O(n^3)$, $O(n^5)$, são algoritmos chamados de eficientes para problemas de entrada de tamanho n . Embora $\log n$ não seja um polinômio, seu valor é menor que n , ou seja, não é pior que um polinomial, por isso pode ser encaixado entre os **polinomiais**. Da mesma forma, $O(\log n)$, $O(n^2 \log n)$ e outros, também estão entre os polinomiais. Os demais são genericamente chamados de **exponenciais**, como $O(2^n)$, $O(3^n)$, $O(n!)$, $O(n^n)$, e outros cuja complexidade não pode ser expressa por um polinômio.

A distinção entre *algoritmos polinomiais eficientes* e *algoritmos exponenciais ineficientes* possui algumas exceções:

- Um algoritmo 2^n é mais rápido que um algoritmo n^5 , para $n \leq 20$. É bom lembrar que na prática, os algoritmos polinomiais tendem a ter grau 2 ou 3 no máximo, e não possuem coeficientes muito grandes (n^{100} ou $10^{99}n^2$ em geral não ocorrem)
- Existem algoritmos exponenciais que são muito úteis na prática. Por exemplo, o algoritmo Simplex para Programação Linear é exponencial, mas executa rápido na prática.

Podemos estender esta definição para um problema. Considere a coleção de todos os algoritmos para resolver um certo problema. O interesse é então saber se, nessa coleção, existe algum algoritmo que seja eficiente, isto é, de complexidade polinomial. Se existir tal algoritmo o problema é chamado de **tratável** ou bem resolvido. Caso contrário, é chamado de **intratável**¹. A idéia é que um problema tratável pode ser resolvido para entradas e saídas de tamanho razoável, através de um programa de computador. Por outro lado, um algoritmo de complexidade não polinomial, para um problema intratável, poderia em certos casos levar séculos para computar os dados de entrada e saída de tamanhos relativamente reduzidos. Suponha que um algoritmo $O(2^n)$ gaste 1 hora para resolver um problema de tamanho $n = 50$, em um computador que realiza 10^9 operações por segundo. Então, levaria cerca de 2 horas para $n = 51$, e um ano para $n = 59$. Ou seja, pequenas alterações no tamanho da entrada fazem com que o tempo de resposta aumente exponencialmente. Mesmo um milhão de computadores trabalhando juntos, cada um deles um milhão de vezes mais rápido que os atuais, não seriam suficientes para resolver esse problema para uma entrada de tamanho 100.

De acordo com esta definição, *para mostrarmos que um problema é tratável, basta apresentar um algoritmo polinomial que o resolva*. Por outro lado, *para verificar que*

¹ "intratável" do ponto de vista prático, pois levaria muito tempo (anos, séculos, ...) para ser resolvido.

é intratável, há necessidade de provar que todo algoritmo que o resolva não possui complexidade polinomial. Isto é bastante difícil de se mostrar, e conforme veremos mais adiante, existem vários problemas que não se sabe se são tratáveis ou não. O que se sabe é que *por enquanto* não são tratáveis.

Os problemas para os quais conhecemos apenas algoritmos exponenciais são geralmente problemas onde existe um grande número de alternativas e precisamos verificá-las para saber qual responde ao problema. Quando o número de alternativas é exponencial fica difícil desenvolver um algoritmo eficiente. Se testarmos uma de cada vez o algoritmo se torna exponencial. Se conseguirmos descobrir alguma propriedade particular do problema que elimine algumas alternativas sem necessidade de verificá-las, podemos melhorar o desempenho do algoritmo. Isto depende de um estudo aprofundado ou de um *insight* sobre o problema na busca destas propriedades. Se conseguíssemos testar todas as alternativas simultaneamente, o desempenho seria consideravelmente melhorado. Isto pode ser feito usando programação distribuída numa rede de computadores, onde as tarefas são distribuídas e cada computador verifica uma alternativa. Mas o número de computadores necessários seria exponencial, e o tempo para dividir o problema em vários subproblemas e combinar as soluções parciais também deve ser considerado. Por enquanto, parece não existir solução para o problema.

2. Tipos de problemas

Existem algumas classes gerais de problemas que podem ser resolvidos por algoritmos, entre elas os problemas de **decisão**, de **localização** e de **otimização**. Num **problema de decisão**, o objetivo consiste em decidir se a resposta é SIM ou NÃO. Num **problema de localização**, o objetivo é localizar uma certa estrutura que satisfaça um conjunto de propriedades. Se estas propriedades envolvem algum critério de otimização através da comparação de soluções, então o problema torna-se um **problema de otimização**.

Problema de Decisão:

Existe uma estrutura S que satisfaça uma propriedade de P ?

Resposta: SIM ou NÃO (dizer se há ou não uma solução)

Problema de Localização:

Encontrar uma estrutura S que satisfaça uma propriedade P

Resposta: uma estrutura S (encontrar uma solução)

Problema de Otimização:

Encontrar uma estrutura S que satisfaça uma propriedade P e que seja a melhor segundo algum critério C de medida.

Resposta: uma estrutura S tal que não haja outra melhor

Em alguns casos existem triplas de problemas, um de decisão, outro de localização e outro de otimização, que podem estar associados, conforme ilustram os exemplos a seguir:

a) Problema de Clique² em um grafo

Decisão: Dado o grafo G , existe um clique de tamanho maior ou igual a 3?

Localização: Dado o grafo G , encontre um clique de tamanho maior ou igual a 3.

² um clique em um grafo é um subconjunto de vértices contendo todas as arestas possíveis entre eles.

Otimização: Dado o grafo G , qual o maior clique?

| | |
|--------------------------|---|
| <p>Figura 1 - Clique</p> | <p>Decisão: SIM (já que existe um clique > 3)</p> <p>Localização: $\{c, e, f\}$ (é um clique de tamanho > 3)</p> <p>Otimização: $\{c, d, e, f\}$ (o maior clique, de tamanho $= 4$)</p> |
|--------------------------|---|

b) Problema do Caixeiro Viajante³

Decisão: Dado o grafo G , existe um percurso de peso menor ou igual a 17?

Localização: Dado o grafo G , encontre um percurso de peso menor ou igual a 17.

Otimização: Dado o grafo G , qual o percurso de menor peso total?

| | |
|-------------------------------------|--|
| <p>Figura 2 - Caixeiro Viajante</p> | <p>Decisão: SIM (já que existe um percurso de peso total ≤ 17)</p> <p>Localização: a, b, c, d, a (que é um percurso de peso total $= 16$)</p> <p>Otimização: a, b, d, c, a (que é o menor percurso, de peso total $= 11$)</p> |
|-------------------------------------|--|

Naturalmente, o custo de se resolver um problema de decisão não é maior que o de se resolver um problema de localização associado. Resolvendo um de localização teremos resolvido o de decisão. E o custo de se resolver um problema de localização não é maior que o de resolver o do problema de otimização associado. Resolvido o de otimização, temos uma resposta para o de localização.

Para a divisão de problemas nas classes P, NP, NP-Completo, NP-Difícil, que veremos a seguir, vamos utilizar geralmente problemas de decisão. A justificativa para esta escolha é simplificar a formulação do problema sem diminuir a complexidade. Se o problema de decisão é intratável, também o de localização e de otimização o serão.

Os problemas de decisão tratáveis estão na classe P de problemas, detalhada a seguir.

3. A Classe P

A classe P de problemas de decisão contém os problemas cuja complexidade de pior caso é polinomial no tamanho da entrada. Para se demonstrar que um problema pertence à classe P, basta mostrar um algoritmo polinomial de pior caso que o resolva.

³ o percurso do caixeiro viajante deve passar por todos os vértices exatamente uma vez, e voltar a origem.

Como exemplos de problemas nesta classe estão o problema de ordenação e o problema do caminho mais curto, descritos a seguir:

Ordenação: dada uma lista contendo n itens, colocá-los em ordem segundo algum critério de ordenação. Existem algoritmos polinomiais para resolver este problema, como Bolha, Inserção e Seleção, que são $O(n^2)$ no pior caso; *Heapsort*, $O(n \log n)$ no pior caso. Portanto, o problema de ordenação é da classe P.

Caminho mais curto: dado um grafo G com pesos positivos nas arestas contendo n vértices, e dois vértices v e w do grafo, encontrar o caminho mais curto entre v e w . Se o grafo for armazenado em uma matriz de adjacência, o algoritmo proposto por Dijkstra [Sczwarcfiter, 1984] encontra a solução em tempo $O(n^2)$ no pior caso, logo, este problema também é da classe P.

Mas para demonstrar que um problema não pertence à classe P, *não* basta mostrar um algoritmo exponencial, pois isto não mostra que não há algum polinomial. E nem dizer que todos os algoritmos conhecidos para resolvê-lo sejam exponenciais. Devemos provar que não existe e nem nunca existirá algoritmo polinomial para resolvê-lo. Por exemplo, os **algoritmos exatos** conhecidos até agora para o problema do caixeiro viajante são todos exponenciais. Entretanto, não é conhecida uma prova de que seja impossível obter um algoritmo polinomial para resolver o caixeiro viajante. Sabemos apenas que, até o momento, ninguém foi capaz de criar um algoritmo polinomial que o resolva. Não se sabe, portanto se o caixeiro viajante pertence ou não à classe P. Como veremos na seção 4, muitos problemas ainda estão indefinidos nesta questão.

4. Alguns problemas aparentemente difíceis

Os problemas a seguir possuem em comum o fato de que são exponenciais todos os algoritmos conhecidos até o momento para resolver qualquer um deles. Esses problemas parecem fáceis à primeira vista, mas desenvolver um algoritmo eficiente para eles não é fácil.

a) Satisfabilidade (SAT)

Entrada: Uma expressão booleana E na FNC (Forma Normal Conjuntiva).

Decisão: E pode ser satisfeita?

Considere um conjunto de variáveis booleanas x_1, x_2, x_3, \dots e seus complementos x_1', x_2', x_3', \dots . Se a variável x_i é verdadeira (V), seu complemento é falso (F), e vice-versa. Uma expressão está na FNC quando as variáveis estão dispostas em cláusulas ligadas umas às outras por AND (representado por \wedge). Cada cláusula é um conjunto de variáveis ligadas por OR (representado por \vee). Verificar se uma expressão pode ser satisfeita consiste em se verificar se existe alguma forma de se atribuir valores V ou F para as variáveis, de tal forma que a expressão seja verdadeira.

Por exemplo, dada a expressão $E = (x_1 \vee x_2') \wedge (x_1' \vee x_2' \vee x_3) \wedge (x_1' \vee x_2 \vee x_3')$, verificar se ela pode ser satisfeita. Resposta: SIM. Basta atribuir os valores $x_1=V, x_2=V, x_3=V$, que $E=V$. Existem outros valores que também provam que a resposta é SIM.

Como outro exemplo, dada a expressão $E = x_1 \wedge x_1'$, verificar se ela pode ser satisfeita. Claramente a resposta é NÃO. Se $x_1=V$, $x_1'=F$ e vice-versa. Assim, a expressão E sempre será falsa.

b) Clique

Entrada: Um grafo G e um inteiro $k > 0$

Decisão: G possui um clique de tamanho $\geq k$?

Dado um grafo $G=(V, A)$, um clique é um subconjunto V' de V tal que todo par de vértices de V' seja adjacente. Isto é, se v e w pertence a V' , então a aresta (v, w) deve existir em A . No grafo da Figura 3, $\{d, b, e\}$ é um clique de tamanho 3. O problema consiste em, dado um grafo, verificar se há um clique de tamanho pelo menos igual a um valor k dado.

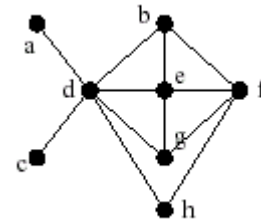


Figura 3 - Clique

c) Conjunto Independente

Entrada: Um grafo G e um inteiro $k > 0$

Decisão: G possui um conjunto independente de tamanho $\geq k$?

Dado um grafo $G=(V, A)$, um conjunto independente de vértices é um subconjunto V' de V tal que todo par de vértices de V' não seja adjacente. Isto é, se v e w pertencem a V' , então a aresta (v, w) não existe em A . No grafo da Figura 3, $\{a, b, c, g, h\}$ é um conjunto independente de tamanho 5. O problema consiste em, dado um grafo, verificar se há um conjunto independente de tamanho pelo menos igual a um valor k dado.

d) Ciclo Hamiltoniano Direcionado

Entrada: Dígrafo D

Decisão: D possui um ciclo hamiltoniano?

Um ciclo hamiltoniano em um dígrafo (grafo com arestas direcionadas) é um ciclo que contém exatamente uma vez cada vértice de D . Assim no grafo $G1$ da Figura 4, o ciclo a, c, f, e, d, b, a é um ciclo hamiltoniano. E o grafo $G2$ não possui ciclo hamiltoniano.

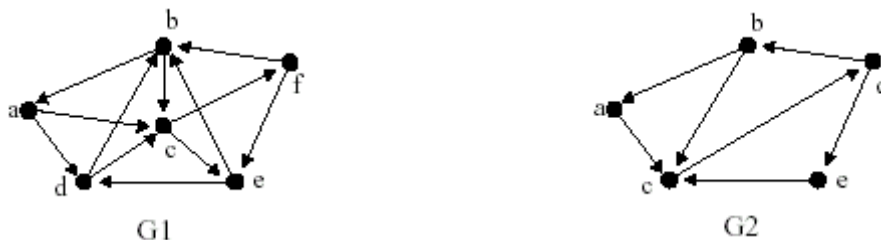


Figura 4 - Ciclo Hamiltoniano

e) Ciclo Hamiltoniano Não Direcionado

Entrada: Grafo G .

Decisão: G possui um ciclo hamiltoniano?

O mesmo que o item d, porém, em um grafo não direcionado.

f) Coloração

Entrada: Grafo G e um inteiro $k > 0$.

Decisão: G possui uma coloração com no máximo k cores?

Uma coloração em um grafo G consiste em se atribuir cores aos vértices de G de tal forma que vértices adjacentes possuam cores diferentes. O grafo da Figura 5 admite uma coloração com 3 cores.

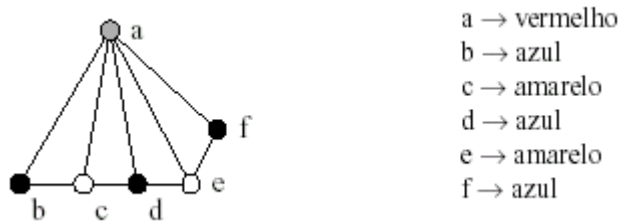


Figura 5 - Coloração

5. A Classe NP

“A common error when speaking of P and NP is to misremember that NP stands for “non-polynomial”; avoid this trap, unless you want to prove it.”

Suponha que, por algum processo, um problema de decisão π foi resolvido. Então, deve haver alguma forma de se testar se a solução SIM ou NÃO dada ao problema realmente é a solução. Para fazer isto, a solução é composta por um conjunto de argumentos que, quando interpretados, atestam a veracidade da resposta SIM ou NÃO dada ao problema.

Considere o problema de clique em um grafo dado, onde se procura um clique de tamanho $\geq k$. Uma justificativa para a resposta SIM pode ser obtida apresentando-se um clique C de tamanho $\geq k$. O teste de veracidade será feito observando se, de fato, C é um clique, e se tem tamanho $\geq k$. Uma justificativa para a resposta NÃO pode ser feita listando-se todos os cliques do grafo. O teste de veracidade consiste em observar se a lista realmente contém os cliques e se o tamanho de cada um deles é $< k$.

O processo de se justificar respostas a problemas de decisão compõe-se de duas fases distintas:

Fase 1: Exibição

- exibir uma justificativa

Fase 2: Reconhecimento

- verificar se a justificativa apresentada no passo de exibição é, de fato, satisfatória.

Considere novamente o problema de Ciclo Hamiltoniano para o grafo G1 a seguir (Figura 6), cuja solução é SIM.

Na justificativa, o passo de exibição consiste da seqüência C de vértices a,b,c,d,e,f,a.

O passo de reconhecimento consiste em verificar se C é de fato um ciclo hamiltoniano. O processo de reconhecimento é simples. Deve-se verificar que (i) C é um ciclo e (ii) C contém cada vértice de G exatamente uma vez cada.

O algoritmo correspondente ao processo de reconhecimento é facilmente implementável em tempo polinomial no tamanho do grafo. Para (i) basta verificar que existem as arestas (a,b) , (b,c) , (c,d) , (d,e) , (e,f) e (f,a) . Para (ii) basta ordenar os vértices e verificar que não há vértice repetido e que a lista contém todos os vértices.



Figura 6 – Ciclo Hamiltoniano

O problema de Ciclo Hamiltoniano para o grafo $G2$, cuja resposta é NÃO, se comporta de forma diferente. O passo de exibição da justificativa consiste em se apresentar as quatro seqüências: $\{a,b,c,a; e,c,d,e; e,c,f,e; e,d,c,f,e\}$. O passo de reconhecimento consiste em comprovar que (i) cada seqüência de vértices é um ciclo não Hamiltoniano e (ii) todo ciclo simples de G está presente na seqüência exibida.

Note que o algoritmo de reconhecimento da justificativa NÃO não é tão simples quanto aquele de justificativa SIM. A operação (i) deve ser realizada para cada ciclo exibido (ao contrário do caso anterior que requeria verificações sobre um único ciclo). O número de ciclos pode ser exponencial, então resultaria em um algoritmo exponencial para verificar *todos*. Além disso, uma idéia para comprovar (ii) seria enumerar *todos* os ciclos do grafo G e verificar se todos estão presentes na justificativa exibida. Como o número total de ciclos pode ser exponencial com o tamanho de G , esse algoritmo é também de natureza exponencial.

Até o momento, é desconhecido se existe algum outro processo para justificar a resposta NÃO a Ciclo Hamiltoniano, tal que o passo de reconhecimento possa ser feito por algum algoritmo polinomial.

Define-se então a classe **NP** de problemas como sendo aquela que compreende todos os problemas de decisão π , tais que existe uma justificativa à resposta SIM para π , cujo passo de reconhecimento pode ser realizado por um algoritmo polinomial no tamanho da entrada de π .

Vale ressaltar que na definição da classe NP não se exige uma solução polinomial para π , apenas que exista um algoritmo polinomial para verificar a resposta SIM. Além disso, o tamanho da justificativa dada pelo passo de exibição não pode ser exponencial com a entrada de π , pois o tamanho da justificativa seria grande demais, e qualquer algoritmo levaria um tempo exponencial no tamanho da entrada de π para verificá-la, mesmo um algoritmo polinomial em relação ao *tamanho da justificativa*. Também não é exigido nada em relação a justificativa NÃO dada pelo passo de exibição. De fato, há problemas de NP que admitem algoritmos polinomiais para suas justificativas NÃO e outros para os quais tal algoritmo não é conhecido.

5.1. Computação Não-determinística: uma forma de caracterizar NP

O termo **NP** vem de **N**ão-determinístico **P**olinomial. A idéia é que um problema NP pode ser resolvido polinomialmente por uma **máquina não-determinística**. Uma máquina não-determinística consegue escolher, entre várias alternativas, aquela que leva à solução. Ou seja, ela “adivinha” o caminho correto. Desta forma, o resultado não é determinístico, ou seja, os passos não são seguidos de uma forma sistemática, sempre havendo apenas um caminho a seguir; ela consegue decidir por si mesma o passo a tomar entre várias opções. Sempre que existir um conjunto de opções que leve a um término com sucesso então este conjunto é escolhido. Uma máquina não-determinística possui uma função **ESCOLHE** que toma estas decisões de forma arbitrária. A ordem de complexidade da função ESCOLHE é $O(1)$.

Vamos ver alguns exemplos de utilização dessa função ESCOLHE em algoritmos para resolver alguns problemas conhecidos:

Exemplo 1: Algoritmo para pesquisar um elemento x em um conjunto de elementos $A[1..n]$, $n \geq 1$.

```
Algoritmo NdSearch // Non-deterministic Search
{
    i := ESCOLHE(A);
    if A[i] = x then sucesso
    else falha
}
```

► A complexidade do algoritmo é $O(1)$. Para um algoritmo determinístico sabemos que a complexidade é $\Omega(n)$.

Apesar de parecer irreal, o conceito de computação não-determinística é um conceito teórico importante e utilizado para caracterizar a classe NP^4 . Um computador não-determinístico, quando diante de duas ou mais alternativas, é capaz de produzir cópias de si mesmo e continuar a computação para cada alternativa.

Exemplo 2: Algoritmo para ordenar um conjunto A contendo n inteiros, $n \geq 1$.

```
Algoritmo NdSort(A) // Non-deterministic Sort
{
    for i := 1 to n do B[i] := 0;
    for i := 1 to n do
    {
        j := ESCOLHE(A);
        B[i] := A[j]
    }
}
```

► Ao final da execução, B contém o conjunto ordenado. Cada inteiro de A é obtido de forma não-determinística com a função ESCOLHE. A complexidade do algoritmo é $O(n)$. Para um algoritmo determinístico, vimos que a complexidade é $\Omega(n \lg n)$.

Exemplo 3: Algoritmo para avaliar se uma expressão é satisfatível.

⁴ A classe NP foi estudada originalmente no contexto de não determinismo. Assim, uma forma de mostrar que um problema pertence à classe NP é exibindo um algoritmo não determinístico polinomial que o resolva.


```

Algoritmo NdSAT(E) // Non-deterministic SAT
{
  for i := 1 to n do
     $x_i$  := ESCOLHE({true, false});
    if E( $x_1, x_2, \dots, x_n$ ) = true then sucesso
    else falha
  }
}

```

► O algoritmo obtém uma das 2^n atribuições possíveis de forma não-determinística em $O(n)$.

Exemplo 4: Algoritmo para encontrar o Ciclo Hamiltoniano de um grafo

```

Algoritmo NdHC // Non-deterministic Hamiltonian Cycle
{
  Entrada: grafo  $G = (V, A)$ 
   $n = |V|$  // número de vértices do grafo
  for i = 1 to n do
    marca[i] = não visitado
  x = 1
  marca[x] = visitado
  for i = 2 to n do
  {
    prox = ESCOLHE(V) // escolhe um dos vértices
    if (marca[prox] = visitado) or (aresta(x, prox)  $\notin$  A) then
      return NÃO
    marca[prox] = visitado
    x = prox
  }
  if (aresta(x, 1)  $\in$  A) then return SIM // fechou o ciclo
  else return NÃO
}

```

► O algoritmo escolhe os vértices do caminho, marcando cada vértice escolhido. Se conseguir escolher todos os vértices, sem repetição, com aresta ligando cada vértice ao próximo, e com aresta ligando o último escolhido ao início do caminho, então retorna SIM. A complexidade do algoritmo é $O(n)$.

► Se escolher um vértice mais de uma vez, ou escolher algum que não tenha aresta ligando ao anterior no caminho, retorna NÃO.

► Em uma máquina não-determinística, se houver um ciclo hamiltoniano em G , a função ESCOLHE escolherá os vértices na ordem correta.

► O algoritmo acima pode ser facilmente modificado para resolver o problema de localização do ciclo hamiltoniano. Basta guardar o caminho, ou seja, a sequência de vértices escolhida.

Algoritmos não-determinísticos, embora muito poderosos teoricamente, também possuem suas limitações. Nem todos os problemas podem ser resolvidos eficientemente por algoritmos não-determinísticos. Por exemplo, suponha que o problema é determinar se o clique máximo em um dado grafo é exatamente k . Podemos usar um algoritmo de clique não-determinístico para encontrar um clique de tamanho k ,

se ele existir, mas não podemos determinar facilmente (nem não-deterministicamente) que não existe um clique de tamanho maior.

6. Como mostrar que um problema pertence a classe NP?

Para comprovar que um problema π pertence à NP procede-se da seguinte maneira:

- (i) Define-se uma justificativa J conveniente para a resposta SIM ao problema π .
- (ii) Elabora-se um algoritmo *polinomial* para reconhecer se J está correta. A entrada desse algoritmo consiste de J e da entrada de π .

Se não pudermos elaborar um algoritmo polinomial para reconhecer a justificativa SIM, não significa que o problema não pertence a NP, mas que ainda não sabemos se pertence ou não. Existem vários problemas que ainda não é conhecido se pertencem ou não a NP. Existem também alguns problemas para os quais são conhecidas provas da não pertinência a NP.

A seguir, vamos verificar se os problemas de Clique, Ciclo Hamiltoniano e Clique Máximo pertencem a NP.

a) Clique \in NP?

- (i) A justificativa para a solução SIM é um subconjunto V' de vértices do grafo G .
- (ii) Para verificar a justificativa SIM basta:
 - Verificar o tamanho do subconjunto apresentado, $O(|V'|)$ – contar os vértices – e verificar se o tamanho é maior ou igual ao valor k de entrada. $O(1)$ – uma comparação simples.
 - Verificar se existem arestas em G entre todos os pares de vértices de V' , que pode ser feito em no máximo $O(|V'|^2)$ – para cada vértice de V' verificar se existem arestas para os outros.

Como tudo pode ser feito em tempo polinomial, então Clique \in NP. Note que $|V'| \leq |V|$, logo é polinomial na entrada do problema, não apenas no tamanho da justificativa.

b) Ciclo Hamiltoniano \in NP?

- (i) A justificativa para a solução SIM é uma seqüência de vértices de G
- (ii) Para verificar a justificativa SIM basta:
 - Verificar se cada vértice aparece exatamente uma vez – contar quantos existem, $O(n)$, e verificar se não há repetido, $O(n)$ – basta marcar enquanto verifica e observar se não marcou mais de uma vez.
 - verificar se existem arestas ligando os vértices consecutivos, $O(n)$ – à medida que percorre a seqüência observar o valor em uma matriz de adjacência.

Tudo pode ser feito em tempo polinomial, logo Ciclo Hamiltoniano \in NP.

c) Clique Máximo \in NP?

A justificativa para a solução SIM é um subconjunto V' de vértices de G . Suponha que V' contenha p vértices e G contenha n vértices. Pode ser verificado que V' realmente é um clique de G em tempo polinomial conforme descrito acima (na prova de que Clique \in NP).

Mas para mostrar que p é o tamanho do maior clique em G é necessário verificar se não existem cliques de tamanho $p+1$. Para isto ser feito, deve ser mostrado que nenhum subconjunto de vértices de G de tamanho $p+1$ é um clique. A verificação de cada um deles é feita em tempo polinomial, mas existem $n!/((p+1)!(n-(p+1))!)$ subconjuntos a serem analisados. Portanto, a verificação de *todos* seria exponencial.

Ainda não se conhece uma forma polinomial de se verificar uma justificativa SIM ao problema de Clique Máximo. Portanto, parece que Clique Máximo \notin NP, embora ninguém tenha provado tal conjectura.

7. A questão $P = NP$

Facilmente demonstramos que $P \subseteq NP$, ou seja, todo problema pertencente à classe P pertence à classe NP. A demonstração pode ser feita da seguinte forma:

Seja $\pi \in P$ um problema de decisão. Existe então um algoritmo A que apresenta a solução de π em tempo polinomial no tamanho de sua entrada. Em particular A pode ser utilizado como algoritmo de reconhecimento para uma justificativa à resposta SIM de π . Logo, $\pi \in NP$.

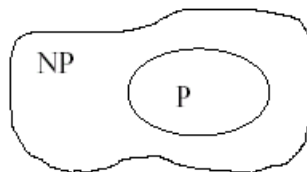


Figura 7 – Relação entre as classes P e NP (Horowitz et al. 1998, p.505)

A pergunta natural seguinte seria: **$P = NP$** ou **$P \neq NP$** ? Ou seja, se todos os problemas para os quais conhecemos um algoritmo polinomial para verificar a justificativa SIM também admitirem um algoritmo polinomial para serem solucionados, embora ainda não conheçamos tal algoritmo, então **$P = NP$** . Entretanto, se existem problemas em NP para os quais não há como se propor um algoritmo polinomial para solucioná-lo, mesmo conhecendo um algoritmo para se verificar a solução SIM, então **$P \neq NP$** .

Como ilustração considere que você tenha chegado a uma festa e quer saber se existe alguém conhecido no recinto. Para solucionar o problema você teria que olhar para cada pessoa, o que seria $O(n)$, sendo n o número de pessoas na festa. Suponha que alguém se apresente como seu conhecido. A verificação da solução SIM pode ser feita em $O(1)$, é claro. Outros exemplos interessantes são os seguintes: se alguém te diz que o número 13.717.421 pode ser escrito como produto de dois outros números menores, como acreditar nele? Entretanto, se ele te disser que pode ser escrito

como 3607×3803 é fácil verificar⁵. Se alguém te der um quebra-cabeça você demostra para montá-lo, deverá praticamente testar a junção de todas as peças, descartando apenas aquelas com formato ou figura totalmente incompatíveis. Mas se o quebra-cabeça estiver pronto, é fácil verificar que está correto. Estes exemplos são meras ilustrações para mostrar que provavelmente encontrar a solução de um problema é mais difícil que verificar uma dada solução.

Até o momento não se conhece uma resposta a esta pergunta $P = NP$ ou $P \neq NP$?⁶. Há fortes evidências de que $P \neq NP$, pois a classe NP incorpora vários problemas com grande interesse prático, para os quais inúmeros pesquisadores, financiados por grandes empresas e agências de pesquisa, já desenvolveram esforços para elaborar algoritmos eficientes, e ainda não obtiveram sucesso. Mas o fato de ninguém ter descoberto não significa que não haja solução. O problema permanece em aberto [Delahaye, 2006].

8. Transformações Polinomiais de Problemas

Suponha que queiramos resolver um certo problema P1 e já conhecemos um algoritmo para resolver um outro problema P2. Uma transformação do problema P1 (com dados de entrada E1 e solução S1) no problema P2 (com dados de entrada E2 e solução S2) consiste em:

- (i) Transformar os dados de entrada E1 do problema P1 em dados de entrada E2 para o problema P2.
- (ii) Executar o algoritmo conhecido para resolver P2
- (iii) Transformar a solução S2 de P2 obtida com os dados de entrada E2 em uma solução S1 de P1.

A Figura 8 ilustra este processo:



Figura 8 – Transformação polinomial do problema P1 no problema P2

Um problema P1 é dito *polinomialmente transformável* em um problema P2, denotado por $P1 \propto P2^7$, quando:

- (i) As transformações de E1 em E2 e de S2 em S1 podem ser feitas em tempo polinomial.

⁵ Em agosto de 2002, foi proposto um algoritmo polinomial para este problema, mostrando que, neste problema, tanto verificar quanto resolver possuem algoritmos eficientes.

⁶ Em maio de 2000, foi proposto um prêmio de um milhão de dólares para quem responder a esta pergunta. As regras podem ser vistas em <http://www.claymath.org/millennium>.

⁷ alguns autores representam por $P1 \leq_p P2$, ou ainda por $P1 \leq_p P2$.

- (ii) Para toda possível entrada $E1$ de $P1$, $P2$ admite resposta SIM se e somente se $P1$ possui resposta SIM.

Quando um problema $P1$ é *polinomialmente transformável* em $P2$, significa que resolvendo $P2$ também resolveremos $P1$.

Observe que quando $P1 \propto P2$, a transformação f deve ser aplicada sobre uma instância genérica de $P1$. A instância que se obtém de $P2$, contudo, é particular, pois é fruto da transformação f utilizada. Assim sendo, de certa forma, $P2$ pode ser considerado como um problema de dificuldade maior ou igual a $P1$. Pois que, mediante f , um algoritmo que resolve a instância particular de $P2$, obtida através de f , resolve também a instância arbitrária de $P1$ dada.

A importância da transformação de entradas e soluções serem feitas em tempo polinomial reside no fato de que elas preservam a natureza do algoritmo (polinomial ou não) empregado na solução $P2$. Assim, se o algoritmo para resolver $P2$ for polinomial, e as transformações forem polinomiais, então $P1$ também pode ser resolvido em tempo polinomial, ou seja, se $P2 \in P$ então $P1 \in P$, e se $P2 \in NP$, então $P1 \in NP$.

O contrário não é necessariamente verdadeiro, ou seja, se $P1 \in P$ não implica que $P2 \in P$. Podemos transformar um problema simples em um complexo e mostrarmos que a solução do problema complexo encontrada é uma solução do simples. Isto de forma alguma mostra que o problema complexo na verdade é simples e nem que o problema simples é complexo. É como usar um canhão para matar algo que uma espingarda de chumbinho poderia matar. Isto não mostra que o problema (o alvo) é difícil de se abater, e nem que o algoritmo (a arma – canhão) é simples. O interessante da transformação polinomial é que, se algo pode ser morto por uma espingarda de chumbinho, então está provado que não é necessário um canhão. Ou seja, se existe um algoritmo eficiente para $P2$, e $P1 \propto P2$, então existe um algoritmo eficiente para $P1$.

A relação \propto é transitiva, ou seja, se $P1$, $P2$ e $P3$ são problemas, e $P1 \propto P2$ e $P2 \propto P3$ então $P1 \propto P3$.

A seguir ilustramos este processo de transformação polinomial do problema de Clique para o problema do Conjunto Independente de Vértices:

Clique \propto Conjunto Independente

Seja $P1$ o problema Clique e $P2$ o problema Conjunto Independente. A entrada $E1$ para o problema Clique consiste em um grafo $G = (V, A)$ e um inteiro $k > 0$. Para transformar esta entrada na entrada $E2$ para o problema Conjunto Independente, considere o grafo $G' = (V, A')$ e o mesmo inteiro k (Figura 9). O grafo G' contém os mesmos vértices que G , mas contém apenas as arestas que não existem em G . Ou seja, dois vértices ligados por uma aresta em G não são ligados em G' , e dois vértices não ligados em G são ligados em G' . Demonstramos que Clique \propto Conjunto Independente, pois:

- (i) A construção de G' a partir de G é polinomial. A transformação da solução também é polinomial, na verdade é $O(1)$ pois nem há necessidade de transformação neste caso.

- (ii) Existe um conjunto independente de tamanho $\geq k$ em G' se e somente se existem um clique de tamanho $\geq k$ em G .

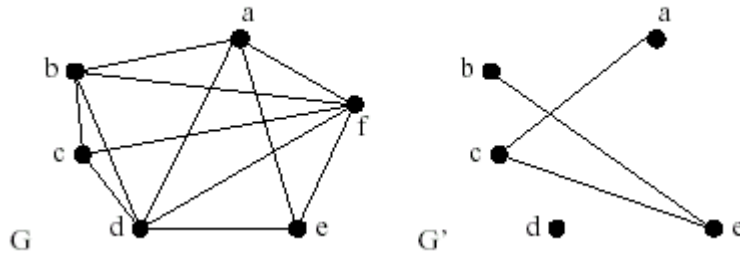


Figura 9 – Transformando o problema de Clique no problema do Conjunto Independente

Note que $\{a, b, d, f\}$ é um conjunto independente de G' de tamanho 4. O mesmo conjunto é um clique em G . O mesmo ocorre para $\{b, c, d, f\}$. Já $\{a, b, c, f\}$ não é conjunto independente em G' e, portanto, não é clique em G .

Portanto, se existir um algoritmo que resolve o problema do Conjunto Independente de Vértices em tempo polinomial, este algoritmo também pode ser utilizado para resolver o problema de Clique também em tempo polinomial. Então Clique \propto Conjunto Independente de Vértices.

Problemas como Clique e Conjunto Independente são chamados *problemas equivalentes*, ou de idêntica dificuldade (no que se refere à existência ou não de algoritmo polinomial para resolvê-los). Dois problemas $P1$ e $P2$ são chamados equivalentes quando $P1 \propto P2$ e $P2 \propto P1$. O fato de que clique é polinomialmente transformável em Conjunto Independente é mostrado no exemplo anterior. Uma idéia semelhante mostra que Conjunto Independente também é polinomialmente transformável em Clique.

Observe que é possível utilizar a relação para dividir NP em classes de problemas equivalentes entre si. Obviamente, os problemas pertencentes a P formam uma dessas classes. E, nesse sentido, podem ser considerados como os problemas de *menor dificuldade* em NP.

Em contrapartida, existe outra classe de problemas equivalentes entre si, que correspondem aos de *maior dificuldade* dentre todos em NP. Os problemas dessa nova classe são denominados **NP-Completo**, cuja definição segue na próxima seção.

9. A Classe NP-Completo

A classe **NP-completo** envolve os problemas de “maior dificuldade” entre todos os problemas de NP. Os problemas pertencentes à classe NP-Completo são todos equivalentes entre si.

Definição 1:

Um problema π pertence à classe NP-Completo quando as seguintes condições forem ambas satisfeitas.

- (i) $\pi \in \text{NP}$
- (ii) todo problema de decisão $\pi' \in \text{NP}$ satisfaz $\pi' \propto \pi$

Observe que (ii) implica que todo problema da classe NP pode ser transformado polinomialmente no problema π NP-Completo. Nisto reside a importância desta classe de problemas: se um problema NP-Completo puder ser resolvido em tempo polinomial, então *todos* os problemas NP também podem... (e conseqüentemente $P = \text{NP}$). Logo, se $\pi \in \text{NP-Completo}$, e alguém mostrar futuramente que $\pi \in P$, então $P = \text{NP}$. Este é o chamado **Teorema de Cook**, formulado por Stephen Cook em 1971. Por isto dizemos que os problemas NP-Completo são os mais difíceis de NP, pois se um deles puder ser resolvido em tempo polinomial, então todos de NP o serão.

Para demonstrar que um problema é NP-Completo utilizando a definição 1 acima, teríamos que mostrar que *todos* os problemas pertencentes a NP podem ser polinomialmente transformados para ele. Isto seria inviável de ser feito na prática, mesmo porque nem sabemos quais são todos os problemas da classe NP. Mas podemos utilizar o seguinte:

*Sejam π_1 e π_2 problemas de decisão $\in \text{NP}$.
Se π_1 é NP-Completo e $\pi_1 \propto \pi_2$ então π_2 é também NP-Completo*

O fato acima, apesar de simples, é bastante poderoso. Vem nos dizer que para mostrar que um problema π é NP-Completo, não precisamos mostrar que *todos* os problemas NP se reduzem a ele, basta escolher um problema qualquer, comprovadamente NP-Completo, e mostrar que este pode ser polinomialmente transformado no problema π em questão.

Definição 2:

Para se provar que $\pi \in \text{NP-Completo}$ é suficiente provar que:

- (i) $\pi \in \text{NP}$
- (ii) um problema NP-Completo conhecido π' é tal que $\pi' \propto \pi$

Isto é suficiente porque, por transitividade, se todos de $\text{NP} \propto$ um problema NP-Completo e este problema NP-Completo $\propto \pi$, então todos de $\text{NP} \propto \pi$, mostrando que π é NP-Completo. A grande vantagem é que dessa forma necessitamos fazer apenas uma transformação polinomial.

Contudo, para que este esquema de prova possa ser utilizado, é necessário escolher algum problema π' que seja NP-Completo. Já existem diversos problemas catalogados como NP-Completo (em particular, todos da seção 4 "*Alguns problemas aparentemente difíceis*" são problemas NP-Completo). Mas, para se identificar o *primeiro* problema desta classe, o processo acima (definição 2) não se aplicou.

Stephen Cook, em 1971, encontrou o ponto de partida através do problema de **Satisfabilidade (SAT)**. Seus trabalhos lhe conferiram o *Prêmio Turing*⁸ em 1982. Ele propôs a seguinte questão: "*Há algum problema em NP que, se for mostrado que ele*

⁸ o "*Prêmio Nobel*" da Ciência da Computação.

está em P, então $P = NP$? Ele mesmo encontrou a resposta através do seguinte teorema:

Satisfabilidade está em P, se e somente se $P = NP$

Isto foi feito através de uma transformação polinomial genérica de todos os problemas da classe NP no problema de Satisfabilidade, cuja demonstração foge do escopo deste texto⁹. Uma vez que este primeiro problema NP-Completo foi identificado, a tarefa para outros problemas fica bem mais simples utilizando a definição 2.

Richard Karp¹⁰, em 1972, apresentou outros 24 problemas importantes que são polinomialmente transformáveis no problema da Satisfabilidade, mostrando assim que também são NP-Completo. Em [Dune, 2004] é apresentada uma lista comentada de 88 problemas NP-completo.

A seguir, apresentamos exemplos de provas de pertinência de alguns problemas a NP-Completo:

a) CLIQUE é NP-Completo?

Os dados de entrada de Clique são um grafo e um inteiro positivo k . Seja C um clique do grafo. Pode-se reconhecer se C é um clique e computar seu tamanho em tempo polinomial no tamanho da entrada de Clique, logo Clique pertence a NP. É necessário agora mostrar que algum problema NP-Completo pode ser polinomialmente transformável em Clique. Vamos usar o problema SAT. Seja E uma expressão genérica de entrada para o problema SAT, contendo as cláusulas L_1, L_2, \dots, L_p . A questão de decidir se E pode ou não ser satisfeita será transformada numa questão de decidir se um certo grafo $G=(V, A)$ possui ou não um clique de tamanho $\geq p$.

O grafo G é construído da seguinte maneira (veja exemplo na Figura 10): criar um vértice diferente em G para cada ocorrência de variável em E ; criar uma aresta (v_i, v_j) em G , para cada par de variáveis x_i, x_j de E , tais que $x_i \neq x_j'$, e x_i, x_j ocorrem em cláusulas diferentes de E . Desta forma, cada aresta (v_i, v_j) de G é tal que as variáveis x_i e x_j , correspondentes em E , estão em cláusulas diferentes e podem assumir o valor verdadeiro simultaneamente. Logo um clique em G com p vértices corresponde em E a p variáveis, uma em cada cláusula (pois não há arestas entre variáveis da mesma cláusula), que podem assumir o valor verdadeiro simultaneamente (pois não há aresta ligando variáveis x_1 a x_1' , x_2 a x_2' ...). A recíproca é verdadeira. Portanto, decidir se E pode ser satisfeita é equivalente a decidir se G possui um Clique de tamanho $\geq p$. A construção de G pode ser feita a partir de E em tempo polinomial com o tamanho de E . Assim, já que:

(i) Clique \in NP

(ii) SAT é NP-Completo e SAT \propto Clique então, **Clique é NP-Completo.**

⁹ O esboço da prova para o Teorema de Cook pode ser encontrado em [Horowitz & Sanhi, 1978, pp. 513-521]. A primeira parte da prova é relativamente simples: SAT está em NP (basta apresentar um algoritmo não determinístico que executa em tempo polinomial). Logo, se $P=NP$, então SAT está em P.

¹⁰ Vencedor do Prêmio Turing em 1985.

$$E = (x_1 \vee x_2') \wedge (x_1' \vee x_2' \vee x_3) \wedge (x_1' \vee x_2 \vee x_3')$$

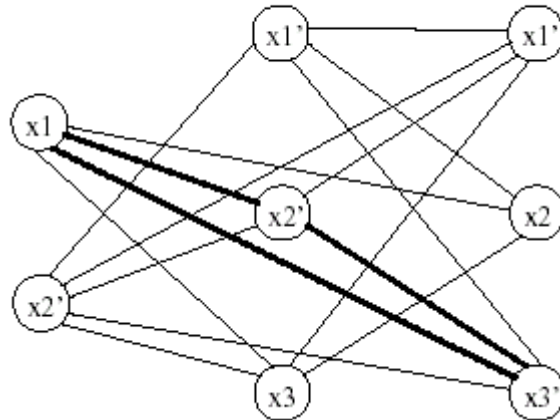


Figura 10 – Transformando uma instância de SAT em instância de Clique

Note que cada clique de tamanho 3 no grafo da Figura 10 corresponde a uma atribuição de Verdadeiro às variáveis correspondentes em E de tal forma que E seja satisfeita.

Em particular o clique mostrado diz que se $x_1 = V$, $x_2' = V$ e $x_3' = V$, então $E = V$. Ou seja, uma solução é $x_1 = V$, $x_2 = F$ e $x_3 = F$.

b) Problema de Decisão do Caixeiro Viajante é NP-Completo?

Este problema de decisão consiste em se determinar se há um ciclo que passa por todos os vértices apenas uma vez com peso total no máximo um valor k . Precisamos mostrar que este problema é NP e que algum NP-Completo é polinomialmente transformável nele. Para isto usaremos o problema NP-Completo Ciclo Hamiltoniano.

(i) Decisão do Caixeiro Viajante é NP

A exibição da justificativa SIM pode ser dada por uma seqüência de vértices. A verificação da justificativa apresentada consiste em se verificar que cada vértice aparece na seqüência apenas uma vez, $O(n)$ – percorrer a lista marcando os vértices, observando se já não foram marcados, verificar se contém todos os vértices, $O(1)$ – durante a marcação contar os vértices, e verificar se o peso total não ultrapassa o valor k , $O(n)$ – percorrer a lista somando-se os pesos das arestas, descobrir cada peso é $O(1)$ se os dados estão em uma matriz de adjacência. Como o algoritmo é polinomial, então o problema é da classe NP

(ii) Decisão do Ciclo Hamiltoniano \propto Decisão do Caixeiro Viajante

Seja um grafo G com n vértices para o qual queremos decidir se existe ou não um ciclo hamiltoniano. Construímos um grafo G' a partir de G contendo todos os vértices e arestas de G . Atribuímos peso 1 a todas estas arestas. Criamos todas as demais arestas não existentes em G' com peso 2 (ver exemplo na Figura 11). É fácil notar que existe um ciclo em G' , que passa em todos os vértices exatamente uma vez, com peso no máximo n , se e somente se existe ciclo hamiltoniano em G , pois assim haveria um ciclo em G' passando apenas por arestas de peso 1. Se não houver, um ciclo em G' deverá passar por aresta de peso 2, o que ultrapassará o peso total n .

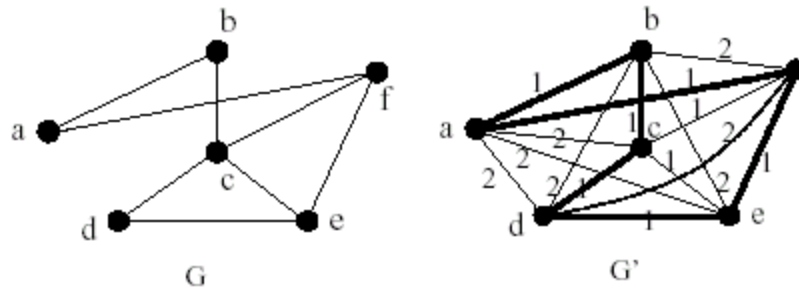


Figura 11 - Transformando uma instância de CH em instância de CV

O ciclo $\{a,b,c,d,e,f,a\}$ em G' tem peso total 6, e representa um ciclo hamiltoniano em G , um grafo de 6 vértices.

Logo, como:

- Decisão do Ciclo Hamiltoniano é NP-Completo (dado);
- Decisão do Caixeiro Viajante é NP;
- Decisão do Ciclo Hamiltoniano \propto Decisão do Caixeiro Viajante;

então **Decisão do Caixeiro Viajante é NP-Completo.**

10. A Classe NP-Difícil

A classe NP-Difícil (do inglês, *NP-hard*) compreende os problemas para os quais apenas o passo (ii) do esquema para se demonstrar que um problema é NP-Completo é considerada. Ou seja, um problema π é NP-Difícil se um problema NP-Completo π' é tal que $\pi' \propto \pi$ não importando se π é ou não NP. Desta forma, a classe NP-Completo é a intersecção da classe NP-Difícil e NP (Figura 12).

Uma outra forma de caracterizar a classe NP-Difícil é:

"Um problema π é NP-Difícil se e somente se $SAT \propto \pi$ ".

Somente problemas de decisão podem ser NP-Completo¹¹, já que em problemas de otimização precisamos verificar se uma dada solução é realmente a melhor, e isto não poderá ser feito em tempo polinomial. Além disso, se π_1 é um problema de decisão e π_2 o correspondente de otimização, é quase certo que $\pi_1 \propto \pi_2$. Isto acontece, por exemplo, para o problema da Mochila 0/1 e para o problema de Clique.

¹¹ Cuidado: Isto não é o mesmo que dizer que "todo problema de decisão é NP-Completo". Por exemplo, temos problemas de decisão NP-difíceis (ex.: *Halting Problem*).

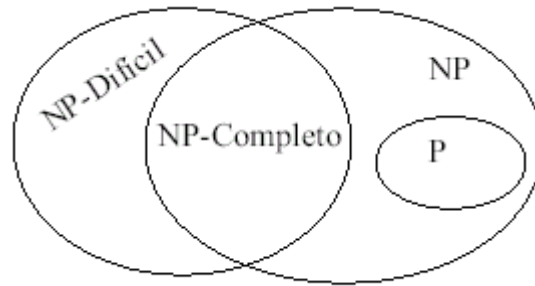


Figura 12 – Possível relacionamento entre as classes de problemas P, NP, NP-Completo e NP-Difícil ([Horowitz et al., 1998, p. 506])

a) Problema de Otimização do Caixeiro Viajante é NP-Difícil?

Já mostramos que o problema de decisão do caixeiro viajante é NP-Completo. Portanto, encontrar um ciclo em um grafo G que passe por todos os vértices, sem repetição, com peso total no máximo k , é NP-Completo. Se resolvemos o problema de otimização do caixeiro viajante para este grafo G , o peso total p do ciclo encontrado é o menor valor possível. Se $p \leq k$, então o problema de decisão tem resposta SIM. Se $p > k$ então o problema de decisão tem resposta NÃO. Logo, é possível transformar polinomialmente Decisão do Caixeiro Viajante em Otimização do Caixeiro Viajante, já que a transformação da entrada – o grafo é o mesmo – e da saída – uma comparação – é $O(1)$. Como o problema de decisão é NP-Completo, então a versão de otimização do caixeiro viajante é NP-Difícil.

b) Halting Problem é NP-Difícil?

Um exemplo de um problema NP-Difícil que não é NP-Completo é o Problema de Parada (*Halting Problem*) [Turing, 1936]. O problema consiste em se decidir para *qualquer algoritmo*¹² e qualquer entrada se o algoritmo vai terminar ou entrar em *loop* infinito. Para mostrar que *Halting Problem* é NP-Difícil, mostraremos que $SAT \propto Halting Problem$. A entrada do algoritmo é a expressão E com n variáveis. O algoritmo testa as 2^n possibilidades de valores para as variáveis. Se algum delas satisfizer a expressão E , o algoritmo pára. Senão, entra em *loop* infinito. Claramente um algoritmo para *Halting Problem* resolve o problema de satisfabilidade, ou seja, se fosse feito um algoritmo para *Halting Problem*, ele poderia ser usado para resolver satisfabilidade. Logo, *Halting Problem* é NP-Difícil. Mas este problema é **indecidível**, não há algoritmo de qualquer complexidade que o resolva. Então ele não é NP. Logo não pode ser NP-Completo.

11. Palavras Finais

Suponha uma situação em que você esteja tentando desenvolver um algoritmo para resolver um determinado problema do mundo real modelado computacionalmente (numa empresa, durante o seu mestrado/doutorado, etc). Depois de vários esforços, você percebe que seu algoritmo até resolve o problema, mas apenas para entradas pequenas. Para entradas maiores, ele gasta um tempo muito grande. Quais das respostas a seguir para o seu padrão/orientador seria a melhor?

¹² Uma discussão interessante sobre a resolução do *Halting Problem* por humanos pode ser encontrada em [Wikipedia, 2004].

💡 *Não consigo fazer melhor que isto. Talvez eu não seja muito esperto... Melhor contratar outra pessoa...*

💡 *Não consigo fazer melhor que isto. E se eu não consigo é porque não tem jeito... um algoritmo eficiente é impossível!*

Com certeza nenhuma destas é uma boa resposta. Com a primeira você perde o emprego. A segunda é difícil que seu chefe acredite... Para quem conhece a Teoria NP-Completo e consegue mostrar que o problema é NP-Completo ou NP-Difícil poderia responder:

💡 *Este problema é reconhecidamente difícil. Existem muitos outros semelhantes, e durante muito tempo, vários dos melhores pesquisadores tem procurado encontrar algoritmos eficientes mas não conseguiram ainda. Vou esperar até que eles consigam...*

👍 *Este problema é reconhecidamente difícil. Existem muitos outros semelhantes, e durante muito tempo, vários dos melhores pesquisadores tem procurado encontrar algoritmos eficientes mas não conseguiram ainda. Vou então implementar uma heurística, ou seja, um algoritmo que não garante encontrar a melhor resposta 100% das vezes, mas que geralmente se aproxima dela, e, além disso, seja rápido.*

Com certeza a segunda opção é melhor. Algumas contribuições práticas da Teoria do NP-Completo é que ela nos ajuda a concluir o que o que podemos fazer ao encontrar um problema NP-Completo ou NP-Difícil:

- Utilizar um algoritmo exato para resolvê-lo (*backtracking*, busca exaustiva, ...) e procurar escolher os melhores caminhos primeiro, resolvendo apenas para entradas pequenas.
- Encontrar um algoritmo que ache uma resposta que, mesmo não sendo ótima, é garantida ser próxima da ótima (método guloso, *branch&bound* versão localização, heurísticas e metaheurísticas¹³). A Tabela 1 apresenta alguns tipos de estratégias de solução algorítmicas e suas características.
- Encontrar um algoritmo que funcione bem na média, mas não necessariamente em todos os casos (a complexidade no pior caso é alta, mas na média é satisfatória).
- Tentar resolver parte do problema à mão ou através de um especialista, para ajudar o algoritmo encontrar a solução.

¹³ Resumidamente, uma metaheurística é uma heurística que sai de ótimos locais, ou seja, aceita movimentos de piora no processo de busca da solução. Exemplos: Busca Tabu, *Simulated Annealing*, GRASP, Algoritmos Genéticos, entre outras. Para mais detalhes, ver [Freitas, 2004].

| Tipos de Algoritmos | Características |
|----------------------------|---|
| Exatos (clássicos) | Podem ser muito lentos. Nunca mentem. Sempre param. Sempre previsíveis. Não usam números randômicos. |
| Heurísticos | Rápidos, se pararem. Podem não resolver o problema. Podem não parar. Podem não ser previsíveis. Às vezes, usam números randômicos. |
| de Aproximação | Sempre rápidos. Dão uma resposta aproximada para o problema ou uma resposta exata para um problema aproximado. Sempre param. Sempre previsíveis. Podem usar números randômicos. |
| Randômicos | Usualmente rápidos. Nunca mentem. Sempre param. Usualmente imprevisíveis. Usam números randômicos. |
| Probabilísticos | Sempre rápidos. Usualmente dizem a verdade. Sempre param. Podem ser imprevisíveis. Podem usar números randômicos. |
| Ergódicos | Sempre rápidos, se pararem. Usualmente dizem a verdade. Podem não parar. Imprevisíveis. Usam números randômicos. |

Tabela 1 – Comparando estratégias de solução algorítmicas ([Rawlins, 1992])

Um bom cientista da computação deveria entender bem a Teoria do NP-Completo. Muitos problemas interessantes, que a princípio não parecem ser mais difíceis que um simples problema de ordenação, são, de fato, problemas NP-Completo. A questão " $P \neq NP$ " é um dos mais perplexos problemas de pesquisa em aberto na Ciência da Computação teórica. Se P é um problema NP-Completo e você encontra um algoritmo determinístico que o resolve em tempo polinomial, então você resolve todos os problemas em NP em tempo polinomial, provando que $P = NP$, e conseqüentemente ganha o *Prêmio Turing* e alguns milhões de dólares.

O estudo de problemas NP-Completo fornece um mecanismo que permite descobrir se um novo problema é "fácil" ou "difícil". Se encontrarmos um algoritmo eficiente para o problema, então não há dificuldade. Se mostrarmos que tal problema é NP-Completo então não temos tanta esperança de encontrar um algoritmo eficiente, mas podemos tentar resolver o problema utilizando outras estratégias.

Resumo

- Um algoritmo é chamado eficiente, ou polinomial, quando pode ser executado em tempo polinomial no tamanho da entrada.
- Um problema é dito tratável quando existe um algoritmo polinomial que o resolva.
- Os problemas podem ser classificados em problemas de decisão, de localização e otimização.
- A classe P contém os problemas para os quais se conhece um algoritmo polinomial para solucioná-lo.

- A classe NP contém os problemas para os quais se conhece um algoritmo polinomial para verificar a justificativa SIM dada como resposta ao problema. A classe NP-Completo compreende os problemas da classe NP que podem resolver todos os outros problemas da classe NP. Ou seja, todos os problemas da classe NP podem ser polinomialmente transformados nos problemas da classe NP-Completo.
- Sabe-se que $P \subseteq NP$. Não se sabe se $P = NP$ ou $P \neq NP$.
- Para se mostrar que um problema é P basta apresentar um algoritmo polinomial que o resolva.
- Para se mostrar que um problema é NP basta apresentar uma forma de representar uma justificativa da resposta SIM e um algoritmo polinomial para verificar esta justificativa.
- Para se mostrar que um problema é NP-Completo basta mostrar que ele é NP e que algum problema NP-Completo é polinomialmente transformável nele.
- Para se mostrar que um problema é NP-Difícil basta mostrar que algum problema NP-Completo é polinomialmente transformável nele.

Exercícios de Fixação

- 1) Elabore o problema de *Multiplicação de Polinômios* como (i) um problema de decisão, (ii) um problema de localização e (iii) um problema de otimização.
- 2) Mostre que o problema de CLIQUE pertence a classe NP.
- 3) Mostre que o problema de CIRCUITO HAMILTONIANO pertence a classe NP.
- 4) Mostre que Conjunto Independente \propto Clique.
- 5) Mostre que CLIQUE é NP-Completo. Utilize SAT como NP-Completo conhecido.
- 6) Mostre que a versão de decisão de Caixeiro Viajante é NP-Completo. Utilize a versão de decisão do Circuito Hamiltoniano como NP-Completo conhecido.
- 7) Seja C um circuito composto das portas lógicas AND, OR e NOT. Sejam x_1, \dots, x_n as entradas desse circuito ($x_i \in \{0,1\}$, $i:1..n$) e f a saída do circuito. Mostre que "decidir se um circuito C tem saída f igual a 1" é um problema NP-Completo.

Obs.: este problema tem grande importância na área de otimização de hardware. Se um subcircuito sempre produz 0, esse subcircuito pode ser substituído por um subcircuito mais simples que omite todas as portas lógicas e fornece o valor constante 0 como sua saída. Um algoritmo de tempo polinomial para esse problema teria considerável utilidade.

- 8) Considere o seguinte problema (*Problema dos Convidados*): Dado um conjunto P de pessoas e um conjunto I de incompatibilidades entre pares de pessoas em P , existe alguma maneira de dispor essas pessoas numa mesa redonda de forma a que não fiquem lado a lado duas pessoas incompatíveis?

Por exemplo, dados $P = \{\text{João, Maria, Ana, Rui}\}$ e $I = \{(\text{João, Rui}), (\text{Maria, Ana})\}$, uma disposição possível seria:

| | | |
|-------|------|-----|
| | João | |
| Maria | | Ana |
| | Rui | |

Se, adicionalmente, o João for incompatível com a Ana, não existe nenhuma disposição possível.

- a) Diz-se que um problema de decisão pertence à classe **NP** (de algoritmo não determinístico de verificação em tempo polinomial), se pode ser resolvido em tempo polinomial por uma máquina de Turing não determinística, ou, equivalentemente, se uma solução pode ser verificada em tempo polinomial por uma máquina de Turing determinística (em última instância, um algoritmo determinístico de tempo polinomial).

Demonstre que o Problema dos Convidados pertence à classe NP.

- b)** Diz-se que um problema de decisão P_1 é redutível a outro problema de decisão P_2 em tempo polinomial se é possível converter em tempo polinomial os dados de entrada de uma instância de P_1 a dados de entrada de uma instância equivalente de P_2 .

Mostre que o Problema dos Convidados é redutível em tempo polinomial ao Problema do Circuito Hamiltoniano. Recorde-se que um circuito Hamiltoniano num grafo não dirigido é um circuito simples (sem vértices duplicados) que passa em todos os vértices.

- 9)** As seguintes afirmativas envolvem o famoso problema **SAT** (Satisfabilidade). Indique quais afirmativas são falsas (F) e quais são verdadeiras (V). Justifique.

1. ____ O problema SAT é NP-Difícil.
2. ____ O problema SAT é NP-Completo.
3. ____ Se encontrarmos um algoritmo que resolva em tempo polinomial o pior caso de um problema da classe NP, então $SAT \in P$.
4. ____ Seja π um problema da classe NP-Completo. Se $SAT \propto \pi$ e $\pi \propto SAT$, então $P = NP$.
5. ____ O problema SAT consiste em verificar se uma expressão booleana na forma normal conjuntiva é sempre verdadeira (tautológica).
6. ____ A importância de SAT reside no fato de ele ser o mais recente problema demonstrado ser NP-Completo.
7. ____ Se $SAT \in P$, então $P = NP$.
8. ____ Se $SAT \propto \pi$ então existe um algoritmo que resolve π em tempo polinomial no pior caso.
9. ____ Se $SAT \propto CLIQUE$, então os dados de entrada de SAT podem ser transformados, em tempo polinomial, nos dados de entrada de CLIQUE.
10. ____ Se $SAT \propto \pi_1$ e $\pi_1 \propto \pi_2$, então $\pi_2 \propto SAT$.

- 10)** Suponha que em um futuro próximo você esteja diante de um novo problema X e suspeita fortemente que ele seja NP-Completo. O problema é bem parecido com a versão de decisão do Problema da Coloração de Grafos (PCG), que você já sabe ser NP-Completo. Diante disto, existem os seguintes caminhos a seguir para comprovar que o seu problema X também é NP-Completo:

1. desenvolver um algoritmo polinomial determinístico que verifica uma solução para o problema X.
2. desenvolver um algoritmo polinomial determinístico que encontra uma solução para o problema X.
3. provar que $X <_{\propto} PCG$.
4. procurar na literatura se X já está catalogado como um problema reconhecidamente NP-Completo.
5. provar que a versão de decisão do PCG $<_{\propto} X$.

6. provar que a versão de otimização do PCG \in NP-Difícil.

Em quais desses caminhos, e em que ordem, você trabalharia? Justifique.

11) Em relação à Teoria do NP-Completo, podemos afirmar:

I) Para mostrar que um problema é intratável, basta apresentar um algoritmo que resolve o seu pior caso em tempo exponencial.

II) O problema da Satisfabilidade (SAT) é NP-difícil.

III) Uma consequência imediata do Teorema de Cook é a seguinte: "Se encontrarmos um algoritmo que resolva em tempo polinomial o pior caso de um determinado problema de NP, então SAT também pertence a P".

- a) II
- b) II e III
- c) I, II e III
- d) I e III
- e) I e II

12) Dados os problemas de decisão P_1 e P_2 , e sabendo que $P_1 \propto P_2$, podemos afirmar:

I) Se P_1 pertence à classe *NP-Completo*, então P_2 também pertence à *NP-Completo*.

II) Se $P_2 \propto SAT$ e $SAT \propto P_1$ então SAT e P_1 são equivalentes.

III) Existe uma forma de transformar em tempo polinomial os dados de entrada de P_1 nos dados de entrada de P_2 .

- a) II
- b) II e III
- c) I, II e III
- d) I
- e) I e II

Bibliografia

1. DA SILVA, Elton José, Projeto e Análise de Algoritmos, notas de aula, Departamento de Informática, PUC-Rio, 2000.
2. Delahaye, Jean-Paul, Sorte ou Inteligência? Scientific American Brasil, Janeiro 2006, pp. 66-71.
3. DUNE, Paul E., *An Annotated List of Selected NP-complete Problems*, [online], disponível na Internet via WWW. URL: http://www.csc.liv.ac.uk/~ped/teachadmin/COMP202/annotated_np.html, arquivo capturado em 06 de dezembro de 2004.
4. FREITAS, Marcone J., Notas de Aula de Inteligência Computacional para Otimização, Departamento de Computação, Universidade Federal de Ouro Preto, 2004.
5. HOROWITZ, E. SAHNI, S. – *Fundamentals of computer algorithms*, Computer Science Press, 1978
6. HOROWITZ, Ellis; SAHNI, Sartah, RAJASEKARAN, Sanguthevar, *Computer Algorithms*, Computer Science Press 1998.
7. Rawlins, Gregory J. E., *Compared to What? Na introduction to the analysis of algorithms*, Computer Science Press, 1992.
8. SANTOS, André Gustavo, Problemas NP-Completo: o que pode e o que não pode ser resolvido eficientemente por computadores, Departamento de Ciência da Computação, UFMG, 1996.
9. SZWARCFITER, Jayme Luiz, *Grafos e algoritmos computacionais*, Editora Campus, 1984
10. TURING, A. M., *On computable numbers, with an application to the Entscheidungsproblem*, Proceedings of the London Mathematical Society, Series 2, 42 (1936), pp 230-265.
11. WIKIPEDIA, *Can humans solve the Halting Problem?* [online], disponível na Internet via WWW. URL: http://en.wikipedia.org/wiki/Halting_problem, arquivo capturado em 12 de dezembro de 2004.