Resolução e Eficiência computacionais: Qui pro quo!

Ana Maria de Almeida

Dep. de Matemática da FCTUC / CISUC

$10^{\underline{o}}$ Problema de Hilbert:

Encontre um procedimento sistemático que teste se

$$P(x_1, x_2, \ldots, x_n) = 0$$

tem soluções inteiras...

Diofantinamente falando, será que existe um algoritmo geral para provar

$$P(x_1, x_2, \ldots, x_n) = Q(x_1, x_2, \ldots, x_n)$$

tem soluções inteiras não negativas?

Exemplo:

A equação :

$$\begin{cases} 3x^2y - 7y^2z^3 &= 18 \\ -7y^2 + 8z^2 &= 0 \end{cases}$$

tem soluções inteiras SE E SÓ SE,

$$(3(x_1-x_2)2(y_1-y_2)-7(y_1-y_2)2(z_1-z_2)3-18)2+(-7(y_1-y_2)2+8(z_2)2)2 = 0$$

tem soluções naturais!

Matiyasevich, Turing, *et al*

★ o $10^{\underline{\text{o}}}$ problema de Hilbert é um desafio para decidir se um sistema de equações de coeficientes inteiros tem solução inteira.

Matiyasevich, Turing, *et al*

- ★ o $10^{\underline{o}}$ problema de Hilbert é um desafio para decidir se um sistema de equações de coeficientes inteiros tem solução inteira.

- ★ Yuri Matiyasevich provou, em 1970, um teorema que implica que o $10^{\underline{o}}$ problema de Hilbert é indecidível (não tem solução!!!).

Matiyasevich, Turing, *et al*

- ★ o $10^{\underline{\text{o}}}$ problema de Hilbert é um desafio para decidir se um sistema de equações de coeficientes inteiros tem solução inteira.

- ★ Yuri Matiyasevich provou, em 1970, um teorema que implica que o $10^{\underline{\text{o}}}$ problema de Hilbert é indecidível (não tem solução!!!).

- ★ Problema da Paragem: Dado um programa e o seu *input* inicial, decidir se o programa, quando iniciado com este *input*, pára.

Matiyasevich, Turing, *et al*

- o $10^{\underline{o}}$ problema de Hilbert é um desafio para decidir se um sistema de equações de coeficientes inteiros tem solução inteira.
- Yuri Matiyasevich provou, em 1970, um teorema que implica que o $10^{\underline{o}}$ problema de Hilbert é indecidível (não tem solução!!!).
- Problema da Paragem: Dado um programa e o seu *input* inicial, decidir se o programa, quando iniciado com este *input*, pára.
- Alan Turing provou, em 1936, que um algoritmo geral para resolver todas as concretizações do Problema da Paragem para todos os *inputs* possíveis não pode existir: o problema é indecidível!

O Diabo dos Números, Enzensberger

"– (...) Imagina que viajas para a América e que tens lá 25 amigos, cada vive numa cidade diferente, e tu queres visitá-los a todos. Agarras no mapa e pensas na melhor maneira de o fazer, ou seja, o mínimo de quilómetros possível de modo a poupares o máximo de tempo e gasolina. Qual é o caminho mais curto? Como poderás descobri-lo?

(...)

– Onde está o problema insolúvel? Só preciso de calcular quantos percursos existem e escolher o mais curto.

– Ahá! - gritou o velhote - Se fosse assim tão simples! Mas com 25 amigos já tens 25! possibilidades, e isto é um número horrorosamente grande. Mais ou menos

$$1600\,000\,000\,000\,000\,000\,000\,000\,00$$

É impossível experimentá-las todas para se saber qual é a mais curta. Nunca chegarias ao fim, mesmo com o mais potente computador que exista.

– Por isso, e numa palavra, não é possível."

O Diabo dos Números, Enzensberger

" – Isso depende muito. Já pensámos muito sobre este assunto. Os Diabos dos Números mais inteligentes já experimentaram com todos os truques possíveis e imaginários, e chegaram à conclusão de que às vezes resulta e outras não.

– Que pena! - disse o Roberto - Se resulta apenas às vezes, ficamos a meio do caminho.

– E, o que é pior ainda, não conseguimos provar, definitivamente, que não existe uma solução perfeita. Nesse caso não teríamos que continuar a procurar. Teríamos ao menos demonstrado que não existe demonstração, o que, afinal de contas, seria também uma demonstração."

Problema do Caixeiro Viajante

CAIXEIRO VIAJANTE (Procura)

Instância: Lista de cidades, $c_1, \ldots, c_n$, e distâncias inteiras positivas, $d(c_i, c_j)$, $i, j = 1, \ldots, n, i \neq j$.

Resposta: Permutação π do conjunto dos índices que minimiza

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

Problema da pesquisa exaustiva . . .

Pesquisa exaustiva: Gerar todas as possíveis permutações de n cidades (2^n), calculando a respectiva soma de distâncias;
Escolher a menor!

Problema da pesquisa exaustiva . . .

Pesquisa exaustiva: Gerar todas as possíveis permutações de n cidades (2^n), calculando a respectiva soma de distâncias;
Escolher a menor!

Para um computador que leva 1 ns por permutação:

Problema da pesquisa exaustiva . . .

Pesquisa exaustiva: Gerar todas as possíveis permutações de n cidades (2^n), calculando a respectiva soma de distâncias;
Escolher a menor!

Para um computador que leva 1 ns por permutação:

n	ns (2^n)	s	dias	anos
10	1024	1.024×10^{-6}	1.18519×10^{-11}	3.24708×10^{-14}
20	1.04858×10^{6}	0.00104858	1.21363×10^{-8}	3.32501×10^{-11}
30	1.07374×10^{9}	1.07374	0.0000124276	3.40481×10^{-8}
40	1.09951×10^{12}	1099.51	0.01272558	0.0000348653
50	1.1259×10^{15}	1.1259×10^{6}	13.0312	0.0357021
60	1.15292×10^{18}	1.15292×10^{9}	13344	36.559
70	1.18059×10^{21}	1.18059×10^{12}	1.36643×10^{7}	37436.3
80	1.208939×10^{24}	1.20893×10^{15}	1.39922×10^{10}	3.83348×10^{7}

Paradigma da Resolução de Problemas

Fenómeno (“Mundo Real” “Mundo Cientifíco”, Realidade, ...)

$\downarrow$

Problema (Formulação Matemática)

$\downarrow$

Algoritmo (Método de Resolução)

$\downarrow$

Programa Computacional (Resolução $\Longleftrightarrow$ Solução)

Algoritmos “Eficientes”

Algoritmos capazes de solucionar problemas em tempo útil

Algoritmo: método de resolução de um dado problema

↘

aplicado a uma particularização garante a obtenção de uma solução

Construir algoritmos que resolvam de um modo eficiente o problema proposto

Algoritmos “Eficientes”

Algoritmos capazes de solucionar problemas em tempo útil

Algoritmo: método de resolução de um dado problema

↘

aplicado a uma particularização garante a obtenção de uma solução

Construir algoritmos que resolvam de um modo eficiente o problema proposto

Algoritmo eficiente ????????

Problemas “duros” e Problemas “fáceis”

Algoritmos com complexidade $\leq \mathcal{O}(n^k)$ ($k \in \{1, 2, 3, 4\}$): (requisitos temporais relativamente baixos)
(algoritmos eficientes: algoritmos polinomiais, log–lineares, lineares)
Algoritmos com complexidade $\mathcal{O}(2^n)$:
algoritmos não-eficientes: algoritmos exponenciais, factoriais,. . .

Problemas “duros” e Problemas “fáceis”

Algoritmos com complexidade $\mathcal{O}(2^n)$:
algoritmos não-eficientes: algoritmos exponenciais, factoriais,. . .

n	n^5	2^n	n^3
2	32	4	8
4	1 024	16	64
10	100 000	1 024	1 000
16	1 048 576	65 536	4 096
22	5 153 632	4 194 304	10 648
23	6 436 343	8 388 608	12 167
25	9 765 625	33 554 432	15 625
26	11 881 376	67 108 864	17 576
30	24 300 000	1 073 741 824	27 000

Eficiência $\leftrightarrow$ Tempo

Tempo expresso em termos do tamanho do problema: quantidade de informação que descreve a particularização em causa – sequência fi nita de símbolos de um dado alfabeto

Complexidade de um algoritmo:
$\rightarrow$ limite superior ($\mathcal{O}$) para o tempo ocupado na resolução do problema pelo algoritmo

Eficiência $\leftrightarrow$ Tempo

Tempo expresso em termos do tamanho do problema: quantidade de informação que descreve a particularização em causa – sequência fi nita de símbolos de um dado alfabeto

Complexidade de um algoritmo:
$\rightarrow$ limite superior ($\mathcal{O}$) para o tempo ocupado na resolução do problema pelo algoritmo

Classificação de problemas relativamente à sua dificuldade computacional intrínseca

Limite superior – $\mathcal{O}$

Diz-se que f é *da ordem de* g e escreve-se $f(n) = \mathcal{O}\big(g(n)\big)$ se e só se

$$\exists c > 0, n_0 > 0, \forall n \geq n_0, f(n) \leq c\, g(n)$$

$$\Updownarrow$$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} \leq c$$

Exemplos

$$\begin{array}{lcl} 73 & = & \mathcal{O}(1) \\ 25n - 5 & = & \mathcal{O}(n) \\ 2n^4 + 2n - 3 & = & \mathcal{O}(n^4) \\ n^2 & = & \mathcal{O}(n^3) \\ n^3 & \neq & \mathcal{O}(n^2) \\ 3n \log n + 7 & = & \mathcal{O}(n \log n) \end{array}$$

($\log$ aqui significa logaritmo de base 2)

Exemplos de Problemas difíceis

★ Coloração de Grafos: Dado um grafo $G = (V, E)$ e um conjunto finito de cores, S, uma coloração para G é uma aplicação $C : V \longrightarrow S$ t.q. se $(v, w) \in E$ então $C(v) \neq C(w)$. O número cromático, $\chi(G)$, é o menor número de cores necessárias para colorir G. Determinar a coloração óptima: C t.q. $C(V) = \chi(G)$?

Exemplos de Problemas difíceis

- ★ Coloração de Grafos: Dado um grafo $G = (V, E)$ e um conjunto finito de cores, S, uma coloração para G é uma aplicação $C : V \longrightarrow S$ t.q. se $(v, w) \in E$ então $C(v) \neq C(w)$. O número cromático, $\chi(G)$, é o menor número de cores necessárias para colorir G. Determinar a coloração óptima: C t.q. $C(V) = \chi(G)$?
- ★ Bin Packing (Empacotamento): Dados n objectos com tamanhos $0 < s_i \leq 1, i = 1, 2, \ldots, n$, quantos contentores de capacidade máxima 1 são precisos para empacotar os objectos?

Exemplos de Problemas difíceis

- ★ Coloração de Grafos: Dado um grafo $G = (V, E)$ e um conjunto finito de cores, S, uma coloração para G é uma aplicação $C : V \longrightarrow S$ t.q. se $(v, w) \in E$ então $C(v) \neq C(w)$. O número cromático, $\chi(G)$, é o menor número de cores necessárias para colorir G. Determinar a coloração óptima: C t.q. $C(V) = \chi(G)$?
- ★ Bin Packing (Empacotamento): Dados n objectos com tamanhos $0 < s_i \leq 1, i = 1, 2, \ldots, n$, quantos contentores de capacidade máxima 1 são precisos para empacotar os objectos?
- ★ Satisfiability (Exequibilidade): Dada uma asserção lógica com n literais (variáveis sobre o conjunto das letras das proposições) na forma conjuntiva normal, existirá uma atribuição de valor para os literais que torne a asserção verdadeira?

Exemplos de Problemas difíceis

- ★ Coloração de Grafos: Dado um grafo $G = (V, E)$ e um conjunto finito de cores, S, uma coloração para G é uma aplicação $C : V \longrightarrow S$ t.q. se $(v, w) \in E$ então $C(v) \neq C(w)$. O número cromático, $\chi(G)$, é o menor número de cores necessárias para colorir G. Determinar a coloração óptima: C t.q. $C(V) = \chi(G)$?
- ★ Bin Packing (Empacotamento): Dados n objectos com tamanhos $0 < s_i \leq 1, i = 1, 2, \ldots, n$, quantos contentores de capacidade máxima 1 são precisos para empacotar os objectos?
- ★ Satisfiability (Exequibilidade): Dada uma asserção lógica com n literais (variáveis sobre o conjunto das letras das proposições) na forma conjuntiva normal, existirá uma atribuição de valor para os literais que torne a asserção verdadeira?
- ★ Circuito Hamiltoniano: Dado um grafo $G = (V, E)$, existirá um circuito fechado (ciclo) que passe em todos os vértices uma e uma só vez?

Exemplos de Problemas difíceis

- ★ Coloração de Grafos: Dado um grafo $G = (V, E)$ e um conjunto finito de cores, S, uma coloração para G é uma aplicação $C : V \longrightarrow S$ t.q. se $(v, w) \in E$ então $C(v) \neq C(w)$. O número cromático, $\chi(G)$, é o menor número de cores necessárias para colorir G. Determinar a coloração óptima: C t.q. $C(V) = \chi(G)$?
- ★ Bin Packing (Empacotamento): Dados n objectos com tamanhos $0 < s_i \leq 1, i = 1, 2, \ldots, n$, quantos contentores de capacidade máxima 1 são precisos para empacotar os objectos?
- ★ Satisfiability (Exequibilidade): Dada uma asserção lógica com n literais (variáveis sobre o conjunto das letras das proposições) na forma conjuntiva normal, existirá uma atribuição de valor para os literais que torne a asserção verdadeira?
- ★ Circuito Hamiltoniano: Dado um grafo $G = (V, E)$, existirá um circuito fechado (ciclo) que passe em todos os vértices uma e uma só vez?
- ★ Caixeiro viajante

Problemas

Algoritmo polinomial: **no pior dos casos** é limitado por uma função polinomial no tamanho do *input*

Problema polinomial: existe (pelo menos um) algoritmo polinomial para resolver qualquer uma das suas instâncias

Classe P: conjunto de problemas (de decisão) polinomialmente limitados

- ★ nem todos os problemas em P possuem algoritmos eficientes
- ★ todos os problemas fora de P são muito “dispendiosos”, ou mesmo, impossíveis de resolver computacionalmente
- ★ para os problemas do acetato anterior não são conhecidos algoritmos polinomiais, logo, estão fora de P

Problemas

Algoritmo polinomial: **no pior dos casos** é limitado por uma função polinomial no tamanho do *input*

Problema polinomial: existe (pelo menos um) algoritmo polinomial para resolver qualquer uma das suas instâncias

Classe P: conjunto de problemas (de decisão) polinomialmente limitados

- ★ nem todos os problemas em P possuem algoritmos eficientes
- ★ todos os problemas fora de P são muito "dispendiosos", ou mesmo, impossíveis de resolver computacionalmente
- ★ para os problemas do acetato anterior não são conhecidos algoritmos polinomiais, logo, estão fora de P

Problema difícil: problema para o qual não se conhece nenhum algoritmo eficiente
NP – Non Deterministic Polinomial Resolution

NP – Non Deterministic Polinomial Resolution

"*By showing that even less ambitious goals than worst-case polynomial exact solution are unattainable, NP-completeness is thus a most useful tool for repeatedly pruning unpromising research directions and thus redirecting research to new ones (in a manner reminiscent of the struggle between Hercules and the monster Hydra).*"

Christos H. Papadimitriou, 1996

Problemas de Optimização

Problema de Optimização:

- ★ D_Π - um conjunto de instâncias;
- ★ a cada instância $x \in D_\Pi$ está associado $S_\Pi(x)$ - conjunto de soluções;
- ★ *função objectivo* - $\begin{array}{rccc} f_\Pi : & S_\Pi(x) & \longrightarrow & \mathbb{R} \\ & s & \longrightarrow & f_\Pi(s)\ . \end{array}$

O *problema de optimização*, $\Pi = (D_\Pi, S_\Pi, f_\Pi)$, consiste em, dada uma instância $x \in D_\Pi$, encontrar um extremo absoluto de f_Π sobre $S_\Pi(x)$.

Solução Óptima de Π:

$s^\star \in S_\Pi(x)$, t. q., $f_\Pi(s^\star) = extr\{f_\Pi(s) \ : \ s \in S_\Pi(x)\}$.

Aproximabilidade

Seja M um algoritmo tal que, dada $x \in D_\Pi$, devolve uma solução admissível, $M(x) = s \in S_\Pi(x)$.

Razão de Aproximação:

Define-se *razão de aproximação de* s *com respeito a* x por,

$$R(x,s) = max\left\{ \frac{f_\Pi(s^\star)}{f_\Pi(s)} \, , \, \frac{f_\Pi(s)}{f_\Pi(s^\star)} \right\}.$$

Quantificação de Aproximação:

1. Dada $r : \mathbb{N} \rightarrow]1, \infty[$, M é um $r(n)$–*aproximante* para Π, sse, $\forall x \in D_\Pi$, $R(x, M(x)) \leq r(|x|)$.
2. Se M é um $r(n)$–aproximante polinomial em tempo diz-se que Π é *aproximável em* $r(n)$.

Esquema de Aproximação:

Um algoritmo M diz-se um *esquema de aproximação* para Π, sse, $\forall x \in D_\Pi, \forall \varepsilon \in \mathbb{Z}^+$, $R(x, M(x)) \leq \varepsilon$.

Métodos “Aproximativos”

- Não garantem a obtenção da solução óptima;

Métodos “Aproximativos”

- Não garantem a obtenção da solução óptima;
- Um bom método de aproximação garante uma boa aproximação;

Métodos “Aproximativos”

- Não garantem a obtenção da solução óptima;
- Um bom método de aproximação garante uma boa aproximação;
- Um bom método de aproximação é (mais) eficiente que o algoritmo (quando este existe);

Métodos “Aproximativos”

- ★ Não garantem a obtenção da solução óptima;
- ★ Um bom método de aproximação garante uma boa aproximação;
- ★ Um bom método de aproximação é (mais) eficiente que o algoritmo (quando este existe);
- ★ Uma técnica que, usualmente, produz boas soluções ou resolve um problema mais simples que contém ou intersecta a solução pretendida é denominada de Heurística(George Pólia, 1945, How to solve it);

Métodos “Aproximativos”

- Não garantem a obtenção da solução óptima;
- Um bom método de aproximação garante uma boa aproximação;
- Um bom método de aproximação é (mais) eficiente que o algoritmo (quando este existe);
- Uma técnica que, usualmente, produz boas soluções ou resolve um problema mais simples que contém ou intersecta a solução pretendida é denominada de Heurística(George Pólia, 1945, How to solve it);
- Heurística: *é o mesmo que a arte de inventar ou descobrir*, e tem por adjectivo heurístico, do grego, *heurískein* ($\varepsilon u \rho i \sigma k \omega$) que significa achar, descobrir ou encontrar, e que respeita à descoberta ou que serve para descobrir.
  (Dicionário da Porto Editora)

Métodos “Aproximativos”

- Não garantem a obtenção da solução óptima;
- Um bom método de aproximação garante uma boa aproximação;
- Um bom método de aproximação é (mais) eficiente que o algoritmo (quando este existe);
- Uma técnica que, usualmente, produz boas soluções ou resolve um problema mais simples que contém ou intersecta a solução pretendida é denominada de Heurística(George Pólia, 1945, How to solve it);
- Heurística: *é o mesmo que a arte de inventar ou descobrir*, e tem por adjectivo heurístico, do grego, *heurískein* ($\varepsilon u\rho i\sigma k\omega$) que significa achar, descobrir ou encontrar, e que respeita à descoberta ou que serve para descobrir.
  (Dicionário da Porto Editora)
- As heurísticas são desenhadas de modo a melhorar, de modo, significativo, a eficiência à custa da precisão e/ou correcção

Aplicações das Heurísticas

- ★ Quando não existe alternativa: nenhum algoritmo ou método de aproximação convergente é conhecido

Aplicações das Heurísticas

- ★ Quando não existe alternativa: nenhum algoritmo ou método de aproximação convergente é conhecido
- ★ Quando existem algoritmos (ou métodos com uma boa “qualidade” de solução) mas onde as heurísticas aceleram o processo de procura da solução(e.g., procura de primeira solução admissível, ...)

Aplicações das Heurísticas

- ★ Quando não existe alternativa: nenhum algoritmo ou método de aproximação convergente é conhecido
- ★ Quando existem algoritmos (ou métodos com uma boa “qualidade” de solução) mas onde as heurísticas aceleram o processo de procura da solução(e.g., procura de primeira solução admissível, ...)
- ★ Muitas técnicas de procura incorporam heurísticas como *look-ahead* (como o *Branch-and-bound*)

Aplicações das Heurísticas

- ★ Quando não existe alternativa: nenhum algoritmo ou método de aproximação convergente é conhecido
- ★ Quando existem algoritmos (ou métodos com uma boa “qualidade” de solução) mas onde as heurísticas aceleram o processo de procura da solução(e.g., procura de primeira solução admissível, ...)
- ★ Muitas técnicas de procura incorporam heurísticas como *look-ahead* (como o *Branch-and-bound*)
- ★ Usadas como elementos particulares do tipo escolha de elemento pivotal

Meta-Heurísticas

- Definição 1: Conjunto de processos iterativos que conduz e modifica parâmetros de heurísticas subordinadas de modo a produzir eficientemente soluções de melhor qualidade;

Meta-Heurísticas

- ★ Definição 1: Conjunto de processos iterativos que conduz e modifica parâmetros de heurísticas subordinadas de modo a produzir eficientemente soluções de melhor qualidade;
- ★ Definição 2: Estratégia de alto nível capaz de guiar heurísticas subordinadas de modo a flexibilizar e adaptar estas a diferentes aplicações;

Meta-Heurísticas

- ★ Definição 1: Conjunto de processos iterativos que conduz e modifica parâmetros de heurísticas subordinadas de modo a produzir eficientemente soluções de melhor qualidade;
- ★ Definição 2: Estratégia de alto nível capaz de guiar heurísticas subordinadas de modo a flexibilizar e adaptar estas a diferentes aplicações;
- ★ Definição 3: Modelo geral (formal) de um dado método heurístico, cuja instanciação leva á concretização dos parâmetros mais adequados a aplicar sobre os dados específicos da instância a resolver;

Meta-Heurísticas

- ★ Definição 1: Conjunto de processos iterativos que conduz e modifica parâmetros de heurísticas subordinadas de modo a produzir eficientemente soluções de melhor qualidade;
- ★ Definição 2: Estratégia de alto nível capaz de guiar heurísticas subordinadas de modo a flexibilizar e adaptar estas a diferentes aplicações;
- ★ Definição 3: Modelo geral (formal) de um dado método heurístico, cuja instanciação leva á concretização dos parâmetros mais adequados a aplicar sobre os dados específicos da instância a resolver;
- ★ Exemplos: Construtivas, Procura Local, Simulação de Têmpera (*Simulated Annealing*), GRASP (Greedy Randomized Adaptive Search Procedure), Procura Tabu (Tabu Search), Colónia de Formigas, Algoritmos Evolutivos e Evolucionistas, etc;