Heurísticas e Metaheurísticas

Sebastián Urrutia
surrutia@dcc.ufmg.br

Departamento de Ciências da Computação
Universidade Federal de Minas Gerais

2008

Teoria da Complexidade

- Lei de Moore (1967): O poder de computo dos processadores dobra a cada 18 meses

Teoria da Complexidade

- Lei de Moore (1967): O poder de computo dos processadores dobra a cada 18 meses
- Isto significa que a cada 18 meses dobra-se o tamanho dos problemas que podem ser resolvidos?

Teoria da Complexidade

- Lei de Moore (1967): O poder de computo dos processadores dobra a cada 18 meses
- Isto significa que a cada 18 meses dobra-se o tamanho dos problemas que podem ser resolvidos?
- Depende da complexidade dos algoritmos disponíveis para resolver o problema

Teoria da Complexidade

- Lei de Moore (1967): O poder de computo dos processadores dobra a cada 18 meses
- Isto significa que a cada 18 meses dobra-se o tamanho dos problemas que podem ser resolvidos?
- Depende da complexidade dos algoritmos disponíveis para resolver o problema
- Essa propriedade se verifica se existe algoritmo linear (ou sub-linear) para resolver o problema

Teoria da Complexidade

- Teoria da Computação: Estuda quais problemas podem ser resolvidos em um modelo de computador e quais não

Teoria da Complexidade

- Teoria da Computação: Estuda quais problemas podem ser resolvidos em um modelo de computador e quais não
- Tese de Church: Todo problema computável pode ser computado por uma Maquina de Turing

Teoria da Complexidade

- Teoria da Computação: Estuda quais problemas podem ser resolvidos em um modelo de computador e quais não
- Tese de Church: Todo problema computável pode ser computado por uma Maquina de Turing
- Teoria da Complexidade: Estuda a complexidade inherente aos problemas computáveis

Teoria da Complexidade

- Teoria da Computação: Estuda quais problemas podem ser resolvidos em um modelo de computador e quais não
- Tese de Church: Todo problema computável pode ser computado por uma Maquina de Turing
- Teoria da Complexidade: Estuda a complexidade inherente aos problemas computáveis
  - Como se modificam os requerimentos de tempo e memória de determinado algoritmo quando o tamanho da instância do problema cresce?
  - Qual é a complexidade do algoritmo de menor complexidade que resolve determinado problema?

Complexidade de algoritmos

- Determinar o número de instruções básicas executadas (resp. a quantidade de memória necessaria) por um algoritmo em relação ao tamanho da instância do problema

Complexidade de algoritmos

- Determinar o número de instruções básicas executadas (resp. a quantidade de memória necessaria) por um algoritmo em relação ao tamanho da instância do problema
- Um algoritmo é da ordem de $O(f(n))$ se existem valores constantes n_0 e C tal que a quantidade de instruções executadas pelo algoritmo é menor ou igual a $C \cdot f(n)$ para todo $n \geq n_0$

Complexidade de algoritmos

- Determinar o número de instruções básicas executadas (resp. a quantidade de memória necessaria) por um algoritmo em relação ao tamanho da instância do problema
- Um algoritmo é da ordem de $O(f(n))$ se existem valores constantes n_0 e C tal que a quantidade de instruções executadas pelo algoritmo é menor ou igual a $C \cdot f(n)$ para todo $n \geq n_0$
- Se um algoritmo é $O(f(n))$ e também é $O(g(n))$ e $f(n) < g(n)$ para todo n maior a um n_0 dizer que o algoritmo é $O(f(n))$ é mais forte do que dizer que é $O(g(n))$

Complexidade de algoritmos

- Determinar o número de instruções básicas executadas (resp. a quantidade de memória necessaria) por um algoritmo em relação ao tamanho da instância do problema
- Um algoritmo é da ordem de $O(f(n))$ se existem valores constantes n_0 e C tal que a quantidade de instruções executadas pelo algoritmo é menor ou igual a $C \cdot f(n)$ para todo $n \geq n_0$
- Se um algoritmo é $O(f(n))$ e também é $O(g(n))$ e $f(n) < g(n)$ para todo n maior a um n_0 dizer que o algoritmo é $O(f(n))$ é mais forte do que dizer que é $O(g(n))$
- Exemplos: InsertionSort é $O(n^2)$, Busca seqüencial é $O(n)$, Busca binária é $O(\log n)$, Imprimir todos os subconjuntos de um conjunto por backtracking é $O(2^n)$

Complexidade de algoritmos

- Os algoritmos cuja complexidade está limitada por un polinómio são chamados de polinomiais

Complexidade de algoritmos

- Os algoritmos cuja complexidade está limitada por un polinómio são chamados de polinomiais
- Os algoritmos cuja complexidade não está limitada por un polinómio são chamados de exponenciais

Complexidade de problemas

- Problema de decisão: Existe uma solução que satisfaça certas propriedades?
  - Resposta: sim ou não
- Problema de otimização: Encontrar uma solução que satisfaça certas propriedades e otimize uma certa função objetivo?
  - Resposta: Solução viável e ótima

Complexidade de problemas

Exemplo: Problema do caixeiro viajante

- Problema de decisão: Dado um valor inteiro L, existe um ciclo hamiltoniano de custo menor ou igual que L?
- Problema de otimização: Encontrar um ciclo hamiltoniano do menor custo possível

Complexidade de problemas

Exemplo: Problema do caixeiro viajante

- Problema de decisão: Dado um valor inteiro L, existe um ciclo hamiltoniano de custo menor ou igual que L?
- Problema de otimização: Encontrar um ciclo hamiltoniano do menor custo possível

Exemplo: Problema da mochila 0-1

- Problema de decisão: Dado um valor inteiro L, existe um subconjunto dos objetos cujo peso no ultrapasse a capacidade da mochila e a soma dos seus custos seja maior ou igaul a que L?
- Problema de otimização: Encontrar um subconjunto de objetos que respeite a capacidade da mochila e maximize o custo

Complexidade de problemas

Grande parte da Teoria da Complexidade baseia-se no estudo de problemas de decisão

- Classe P: Problemas de decisão para os quais existem algoritmos polinomiais
  - Grafos (decisão): caminho mínimo, arvore geradora mínima, emparelhamento perfeito a custo mínimo, etc.
  - Matemática: programação linear (Leonid Khachiyan (1979)), determinar se um número é primo (Manindra Agrawal, Neeraj Kayal, Nitin Saxena (2004))
  - Lógica: 2-SAT

Complexidade de problemas

Algoritmos não-deterministicos:

- Ante uma instrução dada executam duas ações diferentes simultaneamente

Complexidade de problemas

Algoritmos não-deterministicos:

- Ante uma instrução dada executam duas ações diferentes simultaneamente
- Algumas instruções criam dois ou mais 'threads' de execução

Complexidade de problemas

Algoritmos não-deterministicos:

- Ante uma instrução dada executam duas ações diferentes simultaneamente
- Algumas instruções criam dois ou mais 'threads' de execução
- Cada um dos threads roda em um processador diferente

Complexidade de problemas

Exemplo de algoritmo não-deterministico: Mochila 0-1

1. Para j desde 1 até n
2. Goto A,B
3. A: $x_j = 1$
4. Goto C
5. B: $x_j = 0$
6. C:
7. Fim Para
8. Se $x.a \leq b$ e $x.c \geq L$ então responda sim

Complexidade de problemas

Algoritmos não-deterministicos:

- Um algoritmo não-determinístico é polinomial se ele dá uma resposta em tempo polinomial em relação ao tamanho da instância

Complexidade de problemas

Algoritmos não-deterministicos:

- Classe NP: Problemas de decisão para os quais existe um algoritmo não-determinístico polinomial

Complexidade de problemas

Algoritmos não-deterministicos:

- Classe NP: Problemas de decisão para os quais existe um algoritmo não-determinístico polinomial
- São problemas para os quais existe um certificado de solução que pode ser verificado em tempo polinomial

Complexidade de problemas

Algoritmos não-deterministicos:

- Classe NP: Problemas de decisão para os quais existe um algoritmo não-determinístico polinomial
- São problemas para os quais existe um certificado de solução que pode ser verificado em tempo polinomial
- Se dada uma solução é possível verificar em tempo polinomial que ela realmente é solução do problema então o problema está em NP

Complexidade de problemas

Algoritmos não-deterministicos:

- Classe NP: Problemas de decisão para os quais existe um algoritmo não-determinístico polinomial
- São problemas para os quais existe um certificado de solução que pode ser verificado em tempo polinomial
- Se dada uma solução é possível verificar em tempo polinomial que ela realmente é solução do problema então o problema está em NP
- Exemplo: Dada uma solução para o TSP é possível verificar em tempo polinomial que ela é um ciclo hamiltoniano e que o custo das arestas é menor que L

Complexidade de problemas

Algoritmos não-deterministicos:

- Classe NP: Problemas de decisão para os quais existe um algoritmo não-determinístico polinomial
- São problemas para os quais existe um certificado de solução que pode ser verificado em tempo polinomial
- Se dada uma solução é possível verificar em tempo polinomial que ela realmente é solução do problema então o problema está em NP
- Exemplo: Dada uma solução para o TSP é possível verificar em tempo polinomial que ela é um ciclo hamiltoniano e que o custo das arestas é menor que L
  - Grafos: TSP, STP, VRP, Coloração, Máximo conjunto independente, etc.
  - Matemática: Programação Linear Inteira, Programação Quadrática
  - Lógica: SAT, 3-SAT

Complexidade de problemas

- Dado que um algoritmo determinístico é um caso particular de um algoritmo não-determinístico temos que $P \subseteq NP$
- A questão em aberto é $P = NP$ ou $P \subset NP$

Complexidade de problemas

- Transformação polinomial: Um problema de decisão A se transforma polinomialmente em um outro problema de decisão B se sempre é possível a partir de uma instância de A, I_A, construir uma instância de B, I_B em tempo polinomial em relação ao tamanho da instância de A tal que I_A é uma instância **sim** de A se e somente se I_B é uma instância **sim** de B
- Se A se transforma polinomialmente (se reduz) a B então, se existe um algoritmo polinomial para resolver B também existirá um algoritmo polinomial para resolver A

Complexidade de problemas

- Problemas NP-Completos: Um problema de decisão $A \in NP$ é NP-Completo se todos os outros problemas em NP se reduzem a ele
- Se um algoritmo polinomial resolve um problema NP-Completo então todos os problemas em NP podem ser resolvidos em forma polinomial. Isto é, $P = NP$

Complexidade de problemas

- SAT: Dado um conjunto C de cláusulas onde cada uma está formada pela conjunção de literais (variáveis de decisão ou sua negação) do conjunto de variáveis $U = \{u_1, u_2, \ldots u_n\}$, existe uma atribuição de valores às variáveis de forma tal que todas as cláusulas sejam satisfeitas?

Complexidade de problemas

- SAT: Dado um conjunto C de cláusulas onde cada uma está formada pela conjunção de literais (variáveis de decisão ou sua negação) do conjunto de variáveis $U = \{u_1, u_2, \ldots u_n\}$, existe uma atribuição de valores às variáveis de forma tal que todas as cláusulas sejam satisfeitas?
- Exemplo:
  $c_1 = \{\overline{u_1} \vee u_2 \vee u_4\}, c_2 = \{\overline{u_2} \vee u_3\}, c_3 = \{u_1 \vee \overline{u_2} \vee u_3 \vee \overline{u_4}\}$

Complexidade de problemas

- SAT: Dado um conjunto C de cláusulas onde cada uma está formada pela conjunção de literais (variáveis de decisão ou sua negação) do conjunto de variáveis $U = \{u_1, u_2, \ldots u_n\}$, existe uma atribuição de valores às variáveis de forma tal que todas as cláusulas sejam satisfeitas?
- Exemplo:
  $c_1 = \{\overline{u_1} \vee u_2 \vee u_4\}, c_2 = \{\overline{u_2} \vee u_3\}, c_3 = \{u_1 \vee \overline{u_2} \vee u_3 \vee \overline{u_4}\}$
- Resposta: sim. Por exemplo com todas as variáveis em falso

Complexidade de problemas

- SAT: Dado um conjunto C de cláusulas onde cada uma está formada pela conjunção de literais (variáveis de decisão ou sua negação) do conjunto de variáveis $U = \{u_1, u_2, \ldots u_n\}$, existe uma atribuição de valores às variáveis de forma tal que todas as cláusulas sejam satisfeitas?
- Exemplo:
  $c_1 = \{\overline{u_1} \vee u_2 \vee u_4\}, c_2 = \{\overline{u_2} \vee u_3\}, c_3 = \{u_1 \vee \overline{u_2} \vee u_3 \vee \overline{u_4}\}$
- Resposta: sim. Por exemplo com todas as variáveis em falso
- SAT é NP-Completo (Cook 1971)

Complexidade de problemas

- 3-SAT: Caso particular de SAT no qual todas as cláusulas tem 3 literais

Complexidade de problemas

- 3-SAT: Caso particular de SAT no qual todas as cláusulas tem 3 literais
- Teorema: 3-SAT é NP-Completo

Complexidade de problemas

- 3-SAT: Caso particular de SAT no qual todas as cláusulas tem 3 literais
- Teorema: 3-SAT é NP-Completo
- (1) 3-SAT está em NP: Facilmente pode-se determinar um algoritmo não-determinístico para resolver o problema

Complexidade de problemas

- (2) SAT se reduze a 3-SAT

Complexidade de problemas

- (2) SAT se reduze a 3-SAT
  - Para toda cláusula $c_i = \{z_1\}$ com um literal criar quatro cláusulas usando duas variáveis artificiais $y_{i,1}$ e $y_{i,2}$ da seguinte forma: $\{z_1, y_{i,1}, y_{i,2}\}, \{z_1, \overline{y_{i,1}}, y_{i,2}\}, \{z_1, y_{i,1}, \overline{y_{i,2}}\}, \{z_1, \overline{y_{i,1}}, \overline{y_{i,2}}\}$

Complexidade de problemas

- (2) SAT se reduze a 3-SAT
  - Para toda cláusula $c_i = \{z_1\}$ com um literal criar quatro cláusulas usando duas variáveis artificiais $y_{i,1}$ e $y_{i,2}$ da seguinte forma: $\{z_1, y_{i,1}, y_{i,2}\}, \{z_1, \overline{y_{i,1}}, y_{i,2}\}, \{z_1, y_{i,1}, \overline{y_{i,2}}\}, \{z_1, \overline{y_{i,1}}, \overline{y_{i,2}}\}$
  - Para toda cláusula $c_i = \{z_1, z_2\}$ com dois literais criar duas cláusulas usando uma variável artificial $y_{i,1}$ da seguinte forma: $\{z_1, z_2, y_{i,1}\}, \{z_1, z_2, \overline{y_{i,1}}\}$

Complexidade de problemas

- (2) SAT se reduze a 3-SAT
  - Para toda cláusula $c_i = \{z_1\}$ com um literal criar quatro cláusulas usando duas variáveis artificiais $y_{i,1}$ e $y_{i,2}$ da seguinte forma: $\{z_1, y_{i,1}, y_{i,2}\}, \{z_1, \overline{y_{i,1}}, y_{i,2}\}, \{z_1, y_{i,1}, \overline{y_{i,2}}\}, \{z_1, \overline{y_{i,1}}, \overline{y_{i,2}}\}$
  - Para toda cláusula $c_i = \{z_1, z_2\}$ com dois literais criar duas cláusulas usando uma variável artificial $y_{i,1}$ da seguinte forma: $\{z_1, z_2, y_{i,1}\}, \{z_1, z_2, \overline{y_{i,1}}\}$
  - Para toda cláusula $c_i = \{z_1, z_2, z_3\}$ com três literais criar essa mesma cláusula

Complexidade de problemas

- (2) SAT se reduze a 3-SAT
  - Para toda cláusula $c_i = \{z_1\}$ com um literal criar quatro cláusulas usando duas variáveis artificiais $y_{i,1}$ e $y_{i,2}$ da seguinte forma: $\{z_1, y_{i,1}, y_{i,2}\}, \{z_1, \overline{y_{i,1}}, y_{i,2}\}, \{z_1, y_{i,1}, \overline{y_{i,2}}\}, \{z_1, \overline{y_{i,1}}, \overline{y_{i,2}}\}$
  - Para toda cláusula $c_i = \{z_1, z_2\}$ com dois literais criar duas cláusulas usando uma variável artificial $y_{i,1}$ da seguinte forma: $\{z_1, z_2, y_{i,1}\}, \{z_1, z_2, \overline{y_{i,1}}\}$
  - Para toda cláusula $c_i = \{z_1, z_2, z_3\}$ com três literais criar essa mesma cláusula
  - Para toda cláusula $c_i = \{z_1, z2, \ldots, z_k\}$ com $k > 3$ literais criar $k - 2$ cláusulas usando $k - 3$ variáveis artificiais $y_{i,1}, y_{i,2} \ldots y_{i,k-3}$ da seguinte forma: $\{z_1, z_2, y_{i,1}\}, \{\overline{y_{i,1}}, z_3, y_{i,2}\}, \{\overline{y_{i,2}}, z_4, y_{i,3}\}, \ldots \{\overline{y_{i,k-3}}, z_{k-1}, z_k\}$

Complexidade de problemas

- (2) SAT se reduze a 3-SAT
  - A transformação é polinomial: Observar que o número de cláusulas que se criam é linear em relação a $n \cdot m$ sendo n o número de cláusulas e m o número de variáveis do problema original

Complexidade de problemas

- (2) SAT se reduze a 3-SAT
  - A transformação é polinomial: Observar que o número de cláusulas que se criam é linear em relação a $n \cdot m$ sendo n o número de cláusulas e m o número de variáveis do problema original
  - O problema criado (3-SAT) tem uma solução **sim** se e soamente se o problema original (SAT) tem uma solução **sim**

Complexidade de problemas

- (2) SAT se reduze a 3-SAT
  - A transformação é polinomial: Observar que o número de cláusulas que se criam é linear em relação a $n \cdot m$ sendo n o número de cláusulas e m o número de variáveis do problema original
  - O problema criado (3-SAT) tem uma solução **sim** se e soamente se o problema original (SAT) tem uma solução **sim**
  - Exemplo: Ver quadro

Complexidade de problemas

- (2) SAT se reduze a 3-SAT
  - A transformação é polinomial: Observar que o número de cláusulas que se criam é linear em relação a $n \cdot m$ sendo n o número de cláusulas e m o número de variáveis do problema original
  - O problema criado (3-SAT) tem uma solução **sim** se e soamente se o problema original (SAT) tem uma solução **sim**
  - Exemplo: Ver quadro
  - 3-SAT é NP-Completo

Complexidade de problemas

- (2) SAT se reduze a 3-SAT
  - A transformação é polinomial: Observar que o número de cláusulas que se criam é linear em relação a $n \cdot m$ sendo n o número de cláusulas e m o número de variáveis do problema original
  - O problema criado (3-SAT) tem uma solução **sim** se e soamente se o problema original (SAT) tem uma solução **sim**
  - Exemplo: Ver quadro
  - 3-SAT é NP-Completo
- Pode-se mostrar que 3-SAT se reduze a máximo conjunto independente pelo que máximo CI também é NP-Completo

Complexidade de problemas

- (2) SAT se reduze a 3-SAT
  - A transformação é polinomial: Observar que o número de cláusulas que se criam é linear em relação a $n \cdot m$ sendo n o número de cláusulas e m o número de variáveis do problema original
  - O problema criado (3-SAT) tem uma solução **sim** se e soamente se o problema original (SAT) tem uma solução **sim**
  - Exemplo: Ver quadro
  - 3-SAT é NP-Completo
- Pode-se mostrar que 3-SAT se reduze a máximo conjunto independente pelo que máximo CI também é NP-Completo
- Da mesma forma se prova que muitos outros problemas são NP-Completos

Complexidade de problemas

- Problemas NP-Difíceis: Problemas aos quais todo problema em NP se 'reduze' mas não necessariamente está em NP
- A reduzão neste caso não é necessariamente entre dois problemas de decisão. Por isso usa-se ourto tipo de reduzão (Turing reduction)
- Problemas de otimização correspondentes aos problemas de decisão NP-Completos

Complexidade de problemas

- Dois cenários possíveis

Complexidade de problemas

- Dois cenários possíveis
- $P = NP$

Complexidade de problemas

- Dois cenários possíveis
- $P = NP$
  - Existem algoritmos polinomiais para todo problema em NP

Complexidade de problemas

- Dois cenários possíveis
- $P = NP$
  - Existem algoritmos polinomiais para todo problema em NP
  - Este seria o caso em que se desenvolvesse um algoritmo polinomial para qualquer problema NP-Completo

Complexidade de problemas

- Dois cenários possíveis
- $P = NP$
  - Existem algoritmos polinomiais para todo problema em NP
  - Este seria o caso em que se desenvolvesse um algoritmo polinomial para qualquer problema NP-Completo
- $P \neq NP$

Complexidade de problemas

- Dois cenários possíveis
- $P = NP$
  - Existem algoritmos polinomiais para todo problema em NP
  - Este seria o caso em que se desenvolvesse um algoritmo polinomial para qualquer problema NP-Completo
- $P \neq NP$
  - Existem problemas em NP para os quais há algoritmos polinomiais e están em P

Complexidade de problemas

- Dois cenários possíveis
- $P = NP$
  - Existem algoritmos polinomiais para todo problema em NP
  - Este seria o caso em que se desenvolvesse um algoritmo polinomial para qualquer problema NP-Completo
- $P \neq NP$
  - Existem problemas em NP para os quais há algoritmos polinomiais e están em P
  - Existem outros problemas em NP para os quais não há algoritmos polinomiais e están em $NP - P$

Complexidade de problemas

- Dois cenários possíveis
- $P = NP$
  - Existem algoritmos polinomiais para todo problema em NP
  - Este seria o caso em que se desenvolvesse um algoritmo polinomial para qualquer problema NP-Completo
- $P \neq NP$
  - Existem problemas em NP para os quais há algoritmos polinomiais e están em P
  - Existem outros problemas em NP para os quais não há algoritmos polinomiais e están em $NP - P$
  - Neste caso há problemas em $NP - P$ que não são NP-Completos (Ladner 1975)

Complexidade de problemas

- Dois cenários possíveis
- $P = NP$
  - Existem algoritmos polinomiais para todo problema em NP
  - Este seria o caso em que se desenvolvesse um algoritmo polinomial para qualquer problema NP-Completo
- $P \neq NP$
  - Existem problemas em NP para os quais há algoritmos polinomiais e están em P
  - Existem outros problemas em NP para os quais não há algoritmos polinomiais e están em $NP - P$
  - Neste caso há problemas em $NP - P$ que não são NP-Completos (Ladner 1975)
  - A única forma de afirmar que este é o verdadeiro cenário é provando a inexistência de um algoritmo polinomial para qualquer problema em NP

Complexidade de problemas

- Dois cenários possíveis
- $P = NP$
  - Existem algoritmos polinomiais para todo problema em NP
  - Este seria o caso em que se desenvolvesse um algoritmo polinomial para qualquer problema NP-Completo
- $P \neq NP$
  - Existem problemas em NP para os quais há algoritmos polinomiais e están em P
  - Existem outros problemas em NP para os quais não há algoritmos polinomiais e están em $NP - P$
  - Neste caso há problemas em $NP - P$ que não são NP-Completos (Ladner 1975)
  - A única forma de afirmar que este é o verdadeiro cenário é provando a inexistência de um algoritmo polinomial para qualquer problema em NP
  - Este cenário é usado de fato por causa da atual inexistência de algortimos polinomiais para muitos problemas em NP