A TEORIA DA COMPLEXIDADE COMPUTACIONAL

Nair Maria Maia de Abreu, D.C.

Este trabalho apresenta as noções básicas e fundamentais à Teoria da Complexidade Computacional, mostra sua situação atual, as principais questões em aberto e os novos rumos e tendências da área.

INTRODUÇÃO

Numerosos problemas são considerados "intratáveis" no sentido da não existência de algoritmos "eficientes" capazes de resolvê-los otimamente. Dentre estes, podemos citar os de natureza combinatória, cuja prova da otimalidade, muitas vezes, só se verifica pela enumeração de todas as possíveis soluções. É intuitivo perceber que, para um dado problema combinatório, o tempo de computação, em função da enumeração das soluções viáveis, cresce "muito rapidamente" com o "tamanho da entrada" dos dados que o definem. Porém, em muitos casos, provar a "intratabilidade" de um dado problema pode ser tão difícil quanto a descoberta de algoritmos "eficientes" para resolvê-lo.

A partir da década de 1970, a Teoria da Complexidade Computacional vem ocupando enorme espaço da literatura, com a publicação de inúmeros artigos e livros de pesquisadores e estudiosos do assunto: Karp,[11] Cook,[3,6] Aho, Hopcroft, Ullman[1], Garey e Johnson.[9]

O objetivo vem sendo a formalização da teoria, através de conceitos e definições consistentes (as palavras dispostas entre aspas nos parágrafos acima precisam estar bem definidas) para o estabelecimento e determinação desses "problemas intratáveis". A forma para um "possível tratamento" na busca de soluções "subótimas", pela determinação e análise de algoritmos e de técnicas heurísticas, constitui-se objeto importante no estudo e desenvolvimento desta teoria.

NOTAS HISTÓRICAS

Em 1937, Alan Turing introduziu sua famosa "máquina de Turing", a mais convincente formalização da noção de "função efetivamente computável", isto é, da noção de função (problema) executável por um algoritmo. A partir daí, tornou-se razoável a preocupação com que problemas são ou não computáveis e com a dificuldade computacional relativa de funções

computáveis. Em 1965, Hartmanis e Stearns escreveram o artigo intitulado "On the Computational Complexity of Algorithms",[10] que, amplamente lido, deu origem formal a este campo de estudo. Por ele, introduziu-se a importante noção de "medida de complexidade", em função do tempo de computação em máquinas de Turing. Nesta mesma época, Cobham[2] definiu e caracterizou a importante classe das funções computáveis em tempo polinomial e Edmonds[7] idealizou que a computabilidade em "tempo polinomial" está associada à noção de "algoritmos tratáveis". Em 1972, Karp[11] notou por P esta classe de problemas. Não é tão natural a correspondência entre "tempo polinomial" *versus* "problema tratável". Um algoritmo em tempo polinomial "na ordem de n^{1000}" é certamente inviável, enquanto um outro, na "ordem exponencial de $2^{0,00001n}$" pode não ser. No entanto, em 1979, Garey e Johnson[9] mostraram que situações como esta não ocorrem, com freqüência, nas aplicações práticas conhecidas.

Em 1971, Cook[3,4] chamou a atenção para a classe NP dos "problemas de decisão" que são resolvidos em tempo polinomial por "máquinas de Turing não-determinísticas", mostrando que a maioria dos "problemas intratáveis", na forma de "problemas de decisão", pertencem a esta classe. Nesta mesma ocasião, Cook mostrou que todo problema de NP pode ser "polinomialmente reduzido" ao "problema de satisfabilidade (SAT)".[3] Ele também sugeriu a existência de outros problemas em NP, com esta mesma propriedade do STA, assegurando a classe dos problemas NPC. Garey e Johnson,[9] em 1979, apresentaram mais de 300 problemas pertinentes a esta nova classe.

A questão de se os problemas NPC são "intratáveis" ou não até hoje não foi respondida, tratando-se de uma das mais famosas "questões em aberto" da área teórica da computação.

CONCEITOS BÁSICOS

Para se ter uma idéia mínima sobre as classes dos problemas citados, torna-se evidente a necessidade do conhecimento de uma série de definições.* Em seguida, é apresentado um exemplo ilustrativo destes conceitos básicos e fundamentais.

- *Problema* — questão geral a ser respondida, usualmente possuindo diversos parâmetros ou variáveis livres, cujos valores não são especificados.

- *Caso do problema* — é obtido por atribuição de valores particulares a todos os parâmetros do problema.

- *Algoritmo* — procedimento "passo a passo" para resolver um dado problema (pode-se pensar em algoritmo como sendo um programa de computador escrito em alguma linguagem precisa de computador).

- *Passo de um algoritmo* — é a computação de uma instrução do programa que implementa o algoritmo.

- *Um algoritmo resolve um problema* π — se o algoritmo aplicado a qualquer que seja o caso I de π, produz uma solução para este caso.

- *Eficiência de um algoritmo* — esta noção envolve todos os diferentes recursos necessários para executar um algoritmo. Por "algoritmo mais eficiente" entende-se, normalmente, "algoritmo mais rápido". Aceita-se para definição de algoritmo eficiente, a existência de um polinômio $p(n)$, tal que para o "pior" caso I de π, cujo comprimento de entrada é n, o algoritmo executa $p(n)$ passos elementares de computação, para resolver o caso.

- *Tempo* — sugere uma unidade de medida da eficiência de um algoritmo. O tempo requerido por um algoritmo é medido em termos de uma única variável — o "comprimento da entrada" de um caso do problema.

- *Comprimento da entrada do caso do problema* — é o total de dados necessários para des-

*A maioria dos conceitos apresentados são noções intuitivas, não cabendo aqui a formalização de tais definições.

crever o caso do problema. O comprimento da entrada depende do "formato" ou "codificação" escolhida para representar os dados. Estes dados podem ser vistos como uma "seqüência finita de símbolos" pertinentes a um determinado alfabeto finito.

EXEMPLO

Seja o problema π definido por "O número natural X é primo?". Um caso I de π pode ser obtido, tomando-se $X = 5127$. Os dados necessários para descrever este caso I podem ser apresentados em pelo menos dois "formatos" diferentes. Na "codificação decimal" o alfabeto de entrada é dado pelos algarismos decimais, a seqüência finita de símbolos é 5127 e o comprimento da entrada do caso considerado é 4. Na "codificação binária" o alfabeto é constituído por 0 e 1 e a seqüência finita de símbolos que descreve o caso é dada por 1010000000111, cujo comprimento é 13. Um dado algoritmo resolve este problema, π, se, e somente se, ele determina se X é ou não primo, para qualquer que seja o número natural atribuído a X. Este algoritmo será considerado eficiente se o tempo descrito por uma função do comprimento da entrada para cada caso de π é limitado por um polinômio em função de n.

ALGORITMOS EM TEMPO POLINOMIAL E TEMPO EXPONENCIAL

Em 1953, Von Neumann estabeleceu uma distinção entre algoritmos em tempo polinomial e exponencial, associando-os, os primeiros, à idéia de "bons" algoritmos e cabendo aos demais a idéia de "algoritmos ruins".

Como foi dito no item 2, na metade da década de 1960, Cobham[2] definiu formalmente a classe P (notação devida a Karp [1972]) dos problemas resolvíveis por algoritmos limitados em "tempo polinomial" e mostrou que P estava bem definida e não dependia do modelo de computador escolhido.

Novamente é preciso apresentar outra série de definições, para que se possa acompanhar melhor o desenvolvimento do assunto.

Função complexidade de tempo de um algoritmo

Expressa o tempo máximo necessário, dentre os possíveis comprimentos de entrada que variam segundo os "tamanhos" dos casos do problema e segundo o "formato" fixado para medir o comprimento de entrada de cada caso.

Esta definição corresponde à medida de complexidade do "pior caso" e trata-se da medida mais usada pelos pesquisadores da área. No entanto, cabe registrar que, muitas vezes, é interessante medir a complexidade pela média ou pela determinação da medida do "melhor caso".[6]

Ordem de uma função

Sejam f e g funções de variáveis inteiras. Uma função $f(n)$ é dita ser da ordem de g, se, e somente se, $\forall n,\ n \mathbb{Z} 0,\ \exists c,\ c \in \mathbb{Z}$, tal que $|f(n)| \mathbb{Z} c|g(n)|$. Notação: $f(n)$ é $O(g(n))$.

Algoritmo de tempo polinomial

É aquele cuja função complexidade de tempo é $O(p(n))$ para alguma função polinomial $p(n)$, quando n é o comprimento da entrada.

Algoritmo de tempo exponencial

É aquele cuja função complexidade de tempo não pode ser limitada por um polinômio.

A distinção entre estes dois tipos de algoritmos é bastante significativa, quando são consideradas soluções de grandes casos de problemas. O Quadro 1, adaptado de Garey e Johnson,[9] ilustra bem este fato. Com o crescimento do comprimento da entrada n de casos de um dado problema π, o tempo computacional, para funções exponenciais, cresce "muito mais rapidamente" que para as polinomiais.

A maioria dos algoritmos de tempo exponencial são meras variações da "busca exaustiva" ou da "enumeração total", enquanto os algoritmos de tempo polinomial são geralmente construídos, analisando-se a natureza do problema e descobrindo-se propriedades que auxiliem na sua construção.

Quadro 1 – *Comparação entre os tempos de complexidade-funções polinomiais X funções exponenciais.*

Função complexidade Tempo	Comprimento n					
	10	20	30	40	50	60
n	0,00001 seg	0,00002 seg	0,00003 seg	0,00004 seg	0,00005 seg	0,00006 seg
n^2	0,0001 seg	0,0004 seg	0,0009 seg	0,0016 seg	0,0025 seg	0,0036 seg
n^3	0,001 seg	0,008 seg	0,027 seg	0,064 seg	0,125 seg	0,216 seg
n^5	1 seg	3,2 seg	24,3 seg	1,7 min	5,2 min	13,0 min
2^n	0,001 seg	1,0 seg	17,9 min	12,7 dias	35,7 anos	366 séc.
3^n	0,059 seg	58 min	6,5 anos	3855 séc.	2×10^8 séc.	$1{,}3 \times 10^{13}$ séc.

Cabe registrar a existência de algoritmos de tempo exponencial, que na prática são considerados eficientes. É o caso do método *simplex* para resolver o problema de programação linear. Isto se justifica porque a medida de complexidade de tempo é estabelecida sobre o "pior caso", enquanto se poderia tomá-la pelo "caso médio". A este respeito, muito recentemente, em 1982, Smale[12] mostrou que o tempo médio do *simplex* é "muito rápido". Com base nestas observações, é possível admitir que a tese de *P* ser a classe dos problemas tratáveis ainda não foi violada.[6]

PROBLEMAS NP-COMPLETOS

Entende-se por problemas intratáveis aqueles "tão difíceis" que não se conhecem algoritmos de tempo polinomial capazes de resolvê-los. Os primeiros "problemas intratáveis" surgiram, na literatura, via Alan Turing, com seus resultados sobre "problemas indecidíveis", no sentido de que nenhum algoritmo poderia resolvê-los. No entanto, inúmeros "problemas intratáveis", encontrados na prática, são "decidíveis". Isto significa que é possível resolvê-los, em tempo polinomial, com o auxílio de um computador "não-determinístico". Estes problemas constituem-se na classe dos problemas NP.

Para melhor compreensão do que está sendo dito, é necessário o entendimento de dois conceitos: problemas de decisão (PD) e modelos não-determinísticos de computador.

Problemas de decisão

São aqueles cuja resposta é "Sim" ou "Não". A maioria dos problemas se apresentam sob esta forma. No entanto, dado um problema, é sempre possível aproximá-lo a um PD. Por exemplo considere π o "problema da decomposição de um número n em fatores primos". É possível substituir π por vários PD, da forma "Seja $n \in \mathbb{Z}$. Para $a, b \in \mathbb{Z}$, existe um fator primo f de n, tal que $a \leqslant f \leqslant b$?"

Para numerosos PD, não se conhecem algoritmos polinomiais (máquinas de Turing) capazes de resolvê-los. Porém, se alguma informação adicional (*guess*) é acrescentada, torna-se possível resolver um PD por um algoritmo polinomial.

Máquinas de Turing não-determinísticas (MTND)

O último parágrafo do item anterior motiva a conceituação de MTND. Dado que a finalidade deste artigo não é o estudo das máquinas de Turing (MT), convém pensar nelas como a representação teórica do computador e nas MTND como simples generalizações das determinísticas (MTD), onde se permite o acréscimo das "suposições". O nome "não-determinístico" se justifica pela não exigência de um "método determinístico" para o cálculo das "suposições". A conversão natural de uma MTND em um algoritmo (MTD) se realiza quando, através da MTND, se obtém uma resposta para o PD a cada suposição apresentada. Infelizmente, esta conversão é feita em "tempo exponencial".

Dado $N \in \mathbb{Z}$, considere o problema de decisão "É verdade ser N não-primo?" Se, adicionalmente, é dado um fator F de N, é fácil verificar em "tempo polinomial" ser N não-primo, simplesmente constatando ser F seu fator. O *guess* é apenas a especificação do fator F. A máquina divide N por F e verifica se o resto é zero. Em caso afirmativo, ela responde SIM; em caso contrário, ela responde NÃO e pára. Neste último caso, a resposta não determina ser N primo, apenas constata-se que F não é fator de N. O certificado de "ser N primo" só se faria depois do teste de todas as possíveis suposições. Tal procedimento pode ser exponencial. Observa-se, portanto, que a MTND é uma ferramenta útil e eficiente na obtenção de respostas positivas, o que significa "sorte" no *guess*. Para respostas negativas, torna-se uma ferramenta insuficiente na resolução do problema decisório.

A classe NPC

A classe P é constituída dos problemas de decisão, resolvíveis por alguma máquina de Turing determinística, o equivalente à existência de um algoritmo com limite de tempo polinomial $p(n)$, que resolve o PD na MTD. A classe NP é constituída pelos problemas de decisão, resolvíveis por uma máquina de Turing não-determinística, com um limite de tempo polinomial.

É imediato das definições acima que $P \subset NP$. A questão $P = NP$ é um dos clássicos problemas em aberto na teoria da complexidade computacional. Acredita-se ser $P \neq NP$, Cook,[6] significando a existência de problemas não-resolvíveis deterministicamente em tempo polinomial. Para a definição da classe NPC, ainda é necessário a noção apresentada a seguir.

Redução polinomial de problemas de decisão

Sejam dois problemas de decisão, D_1 e D_2. Diz-se que existe uma redução polinomial de D_1 a D_2 ($D_1 \propto D_2$), se, e somente se, forem satisfeitas as seguintes condições:

i. Existe uma função $f(I_1)$ do conjunto de entradas de D_1 ao de D_2, tal que a resposta de I_1 é "positiva", com respeito a D_1, sempre que a resposta de $f(I_1)$ for "positiva", com respeito a D_2 e vice-versa.

ii. Existe um algoritmo limitado polinomialmente que calcula $f(I_1)$.

A Fig. 2 ilustra este conceito.

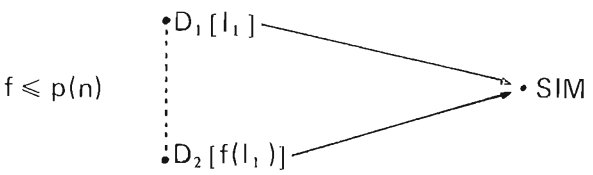

Figura 2 – $D_1 \propto D_2$.

A definição de NPC, segundo Karp[8]

Um problema de decisão D é dito ser NP-completo (NPC), se, e somente se,

i. $D \in NP$

ii. $\forall D', D' \in NP \rightarrow D' \propto D$.

Segundo esta definição, se D ϵ NPC e se for possível construir um algoritmo polinomial que resolve D então $\forall D', D' \epsilon NP \rightarrow D' \epsilon P$. Isto provaria $NP = P$. Um motivo como este é o bastante para justificar o estudo dos problemas NPC.

Os fundamentos básicos da Teoria da NP-Complexidade se encontra no artigo de Cook (1971).[3] Neste artigo Cook apresenta diversos e importantes resultados sobre a teoria da complexidade e exibe o primeiro problema pertinente à classe NPC. Trata-se do "problema da satisfabilidade (SAT)", que tem a propriedade de todo problema em NP ser, polinomialmente, reduzido a ele.

PERSPECTIVAS FUTURAS

Além dos inúmeros problemas em aberto, há vários aspectos nesta teoria que estão sendo desenvolvidos com vigor. Dentre eles é possível citar:

a) *Algoritmos probabilísticos* — são aqueles que determinam com precisão as respostas "positivas", enquanto as respostas "negativas" apresentam um grau de precisão superior a 50%.

b) *A computação paralela simultânea* — a alocação de diversos processadores em *chips* muito pequenos, constituídos por centenas de processadores, trabalhando simultaneamente na resolução de um único problema, sugere uma nova medida de complexidade, em função do número de processadores. Alguns trabalhos já vêm sendo desenvolvidos neste sentido e a preocupação com problemas que podem ser resolvidos significativamente mais rápido tem sido motivação de alguns estudiosos.[6]

c) *O Problema dos limites inferiores e superiores* — é um assunto muito importante e que, apesar de aqui não ser abordado, é fundamental para a avaliação de técnicas heurísticas.

d) *A Teoria da informação computacional* — estudada recentemente por Yao[13] promete apresentar novos rumos à teoria da complexidade, com o envolvimento de noções probabilísticas e aplicações na área da criptografia.

REFERÊNCIAS BIBLIOGRÁFICAS

1. AHO, A.V., Hopcroft, J. E. e ULLMAN, J.D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.
2. COBHAM, A., *The intrinsic computational difficulty of functions.* Proc 1964 International Congress for Logic, Methodology and Philosophy of Sciences. Y. Bar Hellel, Ed., North Holland, Amsterdam, 1965, p. 24-30.
3. COOK, S.A., *The complexity of theorem proving procedures.* Proc. 3rd ACM Symp. on Theory of Computing. Shaker Heights, Ohio, (maio 3-5, 1971), p. 151-158.
4. _____, *Linear time simulation of deterministic two-way pushdown automata.* Proc. IFIP Congress 71, (Theoretical Foundations), North Holland, Amsterdam, 1972, p. 75-80.
5. _____ *Towards, complexity, theory of synchronous parallel computations,* L'Enseignement Mathematique XXVII (1981), p. 99-124.
6. _____, An Overview of Computacional Complexity, Communications of the ACM, junho de 1983, v. 26, nº 6, p. 400-408.
7. EDMONDS, J., Paths, trees, flowers. *Canad. J. Math.* 17 (1965), p. 449-67.
8. EVEN, S., *Graph Algorithms,* Prentice Hall, (1979).
9. GAREY, M.R. e JOHNSON, D.S., *Computer and Intractability. A Guide to the Theory of NP-Completeness,* W. H. Freeman, San Francisco, 1979.
10. HARTMANIS, J. e STEARNS, R.E., On the Computational Complexity of Algorithms, *Trans. AMS* 117(1965), 285-306.
11. KARP, R.M., *Reducibility among combinatorial problems. In: Complexity of Computer Computations,* R.E.Miller e J.W. Thatcher, Eds., Plenum Press, Nova York, 1972, p. 85-104.
12. SMALE, S., *On the average speed of the simplex method of linear programming,* Preprint, 1982.
13. YAO, A.C., *Theory and applications of trapdoor functions (Extended abstract)* Proc. 23rd IEEE Symp. on Foundations of Computer Science. IEEE Computer Society, Los Angeles (1982),.p. 80-91.

Profª Nair Maria Maia de Abreu

Possui graduação em Matemática (UFF, 1974), mestrado em Pesquisa Operacional e Matemática Aplicada (IME, 1977) e doutorado em Engenharia de Produção (COPPE/UFRJ, 1984).

Professora Titular e Coordenadora do Programa de Pesquisa Operacional do IME, Professora Visitante da UFF e Diretora de Publicações da SOBRAPO.