Matemática Universitária, Nº 5, Junho de 1987, 33-60.

P = NP ou as Sutilezas da Complexidade Computacional

Alfredo Iusem

Introdução

Uma nova área de pesquisa surgiu nos últimos quinze anos, na fronteira entre a Matemática Aplicada, a Lógica e a Ciência da Computação. Ainda não possui um nome universalmente aceito, mas a sua denominação mais difundida é "Complexidade Computacional". Seu objetivo é determinar a existência ou não de algoritmos *eficientes* para certos problemas de combinatória.

Como muitas outras áreas da Matemática, esta se desenvolveu juntamente com os esforços para resolver um problema específico, cuja colocação, em 1971, coincide com o nascimento da própria área. É extremamente interessante observar como a formulação de problemas com enunciados concisos e compreensíveis para os não especialistas, porém altamente não triviais, acaba sendo um ótimo mecanismo para gerar novos campos de pesquisa. Sirvam como exemplo os 23 problemas que Hilbert propôs no Primeiro Congresso Internacional de Matemáticos em 1900. Em nosso caso, o enunciado conciso reduz-se à fórmula $P = NP$.

Para entender seu significado, devemos fazer uma breve excursão pelo domínio dos problemas de otimização combinatória, já que a "Complexidade Computacional" dos algoritmos para resolve-los está no nó da questão $P = NP$. Em vez de tentar uma definição precisa de "problema de otimização combinatória", começaremos com sete exemplos típicos. Como os quatro primeiros provém da Teoria dos Grafos, introduzimos a notação e definições básicas necessárias para falar de grafos.

Um *grafo* (finito) é um par de conjuntos finitos: (V, E). Os elementos de V chamam-se *vértices* ou *nós*, e os elementos de E, chamados *arestas* ou *arcos*, são pares (ordenados ou não) de nós. Normalmente, representamos

os nós como pontos no plano e uma aresta (u, v) como um segmento unindo os pontos que representam os nós u e v. Dois nós, u e v, são *adjacentes* se $(u, v) \in E$. Um *caminho* entre dois nós u_0, u_k é uma sequência de nós *diferentes* $u_1, \ldots, u_{k-1}$ tais que u_j e u_{j+1} são adjacentes $(0 \leq j \leq k-1)$. Se $u_0 = u_k$ o caminho é um *ciclo*. Um grafo é *conexo* se para todo par de nós existe um caminho entre eles. Uma *árvore* é um grafo conexo e sem ciclos. Uma *árvore espalhadora* do grafo $G = (V, E)$ é uma árvore da forma (V, E') com $E' \subset E$, ou seja, um subconjunto das arestas de G que atinge todos os nós, é conexo e não forma ciclos. Um grafo se diz *completo* se todo par de nós é uma aresta.

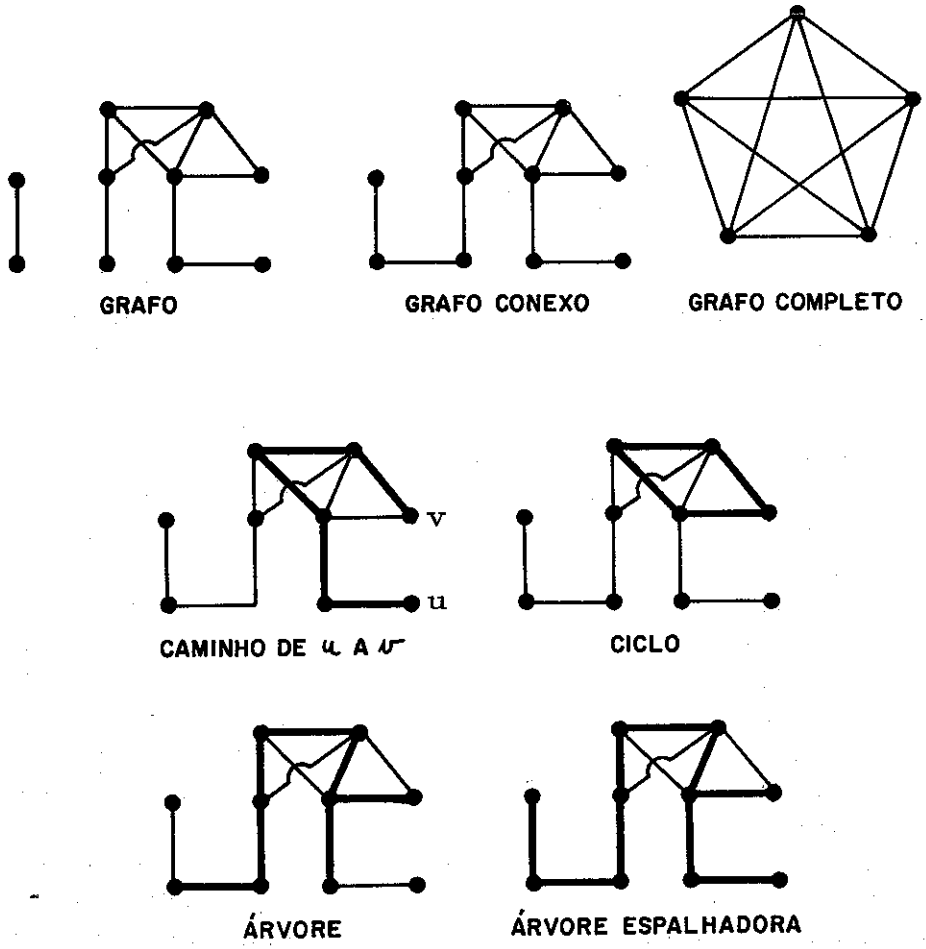


Figura 1

Exemplos de Problemas de Otimização Combinatória

Problema 1. *Caminho mais curto* (CMC): Dado um grafo $G = (V, E)$ e uma função $d : E \to \mathbf{Z}_{\geq 0}$, achar o caminho mais curto entre dois nós u, v ou verificar que não há caminho entre eles, sendo que o comprimento de um caminho $(u_0, u_1, \ldots, u_k)$ é definido como

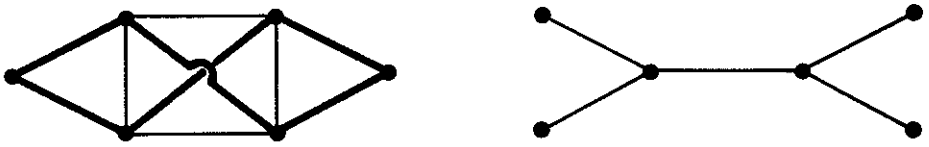
$$\sum_{j=0}^{k-1} d(u_j, u_{j+1}).$$

($d(u, v)$ é interpretado habitualemte como a *distância* entre u e v).

Problema 2. *Árvore Espalhadora Mínima* (AEM): Dado um grafo $G = (V, E)$ e uma função $d : E \to \mathbf{Z}_{\geq 0}$, achar uma árvore espalhadora de comprimento mínimo ou verificar que não há árvore espalhadora (O comprimento de uma árvore $A = (V', E')$ é igual a

$$\sum_{e \in E'} d(e)).$$

Problema 3. *Ciclo Hamiltoniano* (CH): Dado um grafo $G = (V, E)$, achar um ciclo que passe por todos os nós ou verificar que não existe tal ciclo.



CICLO HAMILTONIANO

GRAFO SEM CICLO HAMILTONIANO

Figura 2

Problema 4. *Caixeiro Viajante* (CV): Dado um grafo $G = (V, E)$ e uma função

$$d : E \to \mathbf{Z}_{\geq 0},$$

achar um ciclo hamiltoniano de comprimento mínimo ou verificar que não

existe ciclo hamiltoniano. Se os nós representam cidades e a função d distancia, o problema corresponde a achar o caminho mais curto que passa por todas as cidades (sem repetir nenhuma) voltando à cidade de partida.

Problema 5. *Satisfatibilidade* (SAT): Saimos agora dos grafos para entrar na lógica matemática. Temos variáveis $x_1, \ldots, x_n$ que representam proposições. Cada variável pode tomar dois valores: V ou F (verdadeiro ou falso). Dado x_j, definimos $\overline{x}_j$ (não x_j) como a variável que toma o valor V se e só se x_j toma o valor F. Consideramos também os conectivos $\vee$ (ou) e $\wedge$(e). Uma *cláusula* é uma expressão da forma

$$C = y_1 \vee y_2 \vee \cdots \vee y_k,$$

onde y_j é igual a x_{i_j} ou a $\overline{x}_{i_j}$ (x_{i_j} é algum dos $x_1, \ldots, x_n$). Uma cláusula toma o valor V se *algum dos literais* y_j que a formam toma o valor V, e toma o valor F em caso contrário. Uma *fórmula* é uma expressão da forma

$$\Phi = C_1 \wedge C_2 \wedge \cdots \wedge C_\ell$$

onde

$$C_1, C_2, \ldots, C_\ell$$

são cláusulas. A fórmula Φ toma o valor V se *todas* as cláusulas C_j que a formam tomam o valor V, e F em caso contrário. Observe-se que uma variável x_j pode aparecer (afirmada ou negada) em várias cláusulas, e até várias vezes na mesma cláusula. O problema de satisfatibilidade é o seguinte: Dada uma fórmula Φ, existe uma correspondência (assignment) de valores de verdade (ou seja, uma função $A : \{x_1, \ldots, x_n\} \to \{V, F\}$) que dá à fórmula o valor V? Por exemplo, para a fórmula

$$\Phi = (x_1 \vee x_2) \wedge (x_1 \vee \overline{x}_2) \wedge (\overline{x}_1 \vee \overline{x}_2),$$

a correspondência $x_1 = V, x_2 = F$ produz $\Phi = V$, mas para a fórmula

$$\Phi = (x_1 \vee x_2) \wedge (\overline{x}_1) \wedge (\overline{x}_2)$$

não existe correspondência de valores de verdade para x_1 e x_2 que faça Φ verdadeira.

Problema 6. *Programação Linear* (PL): O problema consiste em minimizar uma função linear de n varjáveis reais na região definida, em $\mathbf{R}^n$, pelo octante positivo e mais m equações lineares, ou seja, em notação matricial:

$$\min c^T x \qquad (1)$$

sujeito a

$$Ax = b \tag{2}$$
$$x_j \geq 0 \qquad (1 \leq j \leq n) \tag{3}$$

onde $x, c \in \mathbf{R}^n$; $b \in \mathbf{R}^m$, $A \in \mathbf{R}^{m \times n}$. Em notação expandida:

$$\min \sum_{j=1}^{n} c_j x_j$$

sujeito a

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \qquad (1 \leq i \leq m)$$

$$x_j \geq 0 \qquad (1 \leq j \leq n).$$

Adicionando variáveis e/ou reescrevendo uma igualdade na forma de duas desigualdades contrárias, PL é equivalente a:

$$\min c^T x \tag{4}$$

sujeito a

$$\acute{A}x \leq b, \tag{5}$$

onde (5) significa

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad (1 \leq i \leq m).$$

Problema 7. *Programação Linear Inteira* (PLI): O problema PLI consiste do problema PL com a restrição adicional de integralidade, ou seja (1) – (3) mais

$$x_j \in \mathbf{Z} \qquad (1 \leq j \leq n) \tag{6}$$

Curiosamente, embora o domínio das variáveis seja menor, neste caso, o problema PLI é consideravelmente mais "difícil" que PL. Neste caso, como no anterior, é possível que o problema não tenha solução, ou seja, que não exista nenhum vetor satisfazendo (2), (3) e (6). Nessa situação, resolver o problema significa estabelecer a inexistência de soluções.

Algoritmos Eficientes

Observando os exemplos anteriores, nota-se que os 5 primeiros podem ser enquadrados no seguinte paradigma:

Dado um conjunto finito X, achar um elemento $x \in X$ que satisfaça uma propriedade π.

Nos casos de CMC, AEM e CV, a propriedade π é a de minimizar uma função $f : X \to \mathbf{Z}$ num subconjunto $U \subset X$. Nos problemas 1 a 4 X pode ser tomado como o conjunto de subconjuntos de E. Em CMC, U é o conjunto de caminhos de u a v e f é o comprimento de um conjunto de arestas. Em AEM, U seria o conjunto de árvores espalhadoras e f o mesmo de CMC. Em CV, U seria o conjunto de ciclos hamiltonianos e f novamente como nos dois casos anteriores. Em CH, a propriedade π é a de ser um ciclo hamiltoniano. Em SAT, $X = \{F,V\}^n$ (todas as sequências de n elementos, cada um dos quais é F ou V), e π é fazer verdadeira a fórmula.

PL e PLI parecem fugir a este padrão: o conjunto X em geral não é finito. Mas a anomalia é só aparente. No caso de PL, as restrições (2) e (3) definem um poliedro em $\mathbf{R}^n$. Da linearidade da função a minimizar, resulta que, se o problema tem solução, existe uma solução que é um vértice desse poliedro, no sentido geométrico usual ([12], pp. 29-44), e o conjunto desses vértices é finito. De fato existe uma identificação algébrica simples de tais vértices. Suponhamos, sem perda de generalidade, que, em PL, $m \leq n$ e o posto de A é m. Escolhamos m colunas linearmente independentes de $A : j_1, \ldots, j_m$ e fixemos o valor zero para x_j se j é diferente de $j_1, \ldots, j_m$. Seja B a matriz (quadrada e não singular) formada pelas colunas $j_1, \ldots, j_m$ de A e seja $x^B \in \mathbf{R}^n$ definido como $x^B_j = x_{j_t}$. Resolvemos $Bx^B = b$ obtemos $x^B = B^{-1}b$. Se $x^B \geq 0$, o vetor x resultante (x^B mais os componentes fixados em 0) é um vértice do poliedro, e todo vértice pode ser obtido dessa forma ([12], pp. 29-44). O número de vértices é então menor ou igual ao número de escolhas de m colunas linearmente independentes de A.

Para o problema PLI, demonstra-se que se o problema tem solução, então existe uma solução x com componentes x_j tal que

$$|x_j| \leq (n+m)(m\alpha)^{2m+4}$$

onde

$$\alpha = \max\{|a_{ij}|\ (1 \leq i \leq n, 1 \leq j \leq m),\ |b_i|(1 \leq i \leq m)\}$$

(ver [12] p. 321). Como os x_j devem ser inteiros, a busca do mínimo pode-se restringir a um conjunto U finito.

É conveniente distinguir agora entre um problema (por exemplo CMC, ou PL) e as suas *instâncias* (cada um dos casos particulares do problema, correspondentes a um dado grafo, para CMC, ou uma dada matriz A e dados vetores b e c, para PL).De fato, pensaremos que cada problema é o conjunto das suas instâncias.

Definimos, de forma propositadamente vaga, um *algoritmo* como *um procedimento que, dada qualquer instância de um certo problema, produz, após um número finito de operações sobre os dados da instância, seja a solução, seja a constatação de que não existe solução.*

As regras usuais para a divisão de inteiros e o método de Euclides para achar o máximo divisor comum de dois inteiros são algoritmos conhecidos por todos.

O objetivo da Otimização Combinatória é achar algoritmos eficientes para resolver este tipo de problema, e a palavra chave é *eficientes*. Isso porque sendo X finito, existe um algoritmo trivial para cada problema: examinem-se um a um os elementos de X até achar um que satisfaz π ou comprovar que nenhum deles satisfaz π. De fato, do ponto de vista da matemática do século passado, estes problemas, após verificada a sua finitude, careceriam absolutamente de interesse matemático (cheguei a ouvir essa opinião, no fim dos anos 60, emitida por um ilustre matemático).

Mas vejamos qual é o cardinal aproximado dos tais conjuntos X. Consideremos um grafo com n nós. Para CMC, a não repetição de nós num caminho diz que existem não mais de $(n-2)!$ caminhos entre dois nós dados. Para CH e CV, sendo que todos os nós devem ser utilizados, só fica por determinar a ordem: há exatamente $n!$ possibilidades, e para cada uma delas é necessário conferir se os nós nessa ordem resultam adjacentes. Para CV, em caso positivo, é preciso ainda calcular as distâncias e compará-las. Para AEM um teorema de Cayley (ver [1], p. 41) diz que um grafo completo com n nós tem n^{n-2} árvores espalhadoras. Para PL, existem $\binom{n}{m}$ formas de extrair m colunas da matriz A. Os algoritmos "ingênuos" mencionados antes requerem portanto um número gigantesco de operações, mesmo para instâncias de tamanho muito limitado.

Um algoritmo *eficiente* seria algum que resolvesse, por exemplo, AEM em n^2 em lugar de n^{n-2} operações. Isto permite compreender melhor o desinteresse da matemática clássica pela otimização combinatória: procedendo manualmente, algoritmos que requerem 10^4 ou 10^{196} operações podem parecer igualmente impraticáveis, mas a perspectiva muda radicalmente na presença de computadores: 10^4 operações podem ser feitas em menos de um segundo; 10^{196} continuam sendo impraticáveis. De fato, a otimização com-

binatória nasceu e se desenvolveu simultaneamente com os computadores, e seus problemas estiveram entre os primeiros a serem atacados com os novos instrumentos.

Para introduzir uma noção mais rigorosa de "eficiência" começaremos definindo o *tamanho* de uma instância S ($|S|$) como *o número de símbolos de um alfabeto dado necessários para representar formalmente a instância* S e indicaremos como $\theta_A(S)$ *o número de operações necessárias para resolver a instância* S *com o algoritmo* A.

Note-se que tanto $|S|$ como $\theta_A(S)$ estão definidos com certa ambigüidade: $|S|$ depende tanto do alfabeto formal utilizado como da maneira de representar a instância. Assim, um grafo pode ser representado tando pela matriz de adjacência nó-nó (com tantas linhas e colunas quantos são os nós; o número um na posição ij se o nó i é adjacente ao nó j, e o número zero em caso contrário) como pela matriz de adjacência nó-aresta (tantas linhas quantos nós e colunas quantas arestas; o número um na posição ie se o nó i é um extremo da aresta e, e zero em caso contrário), como por uma sequência de listas de adjacência: para cada nó, listam-se os nós adjacentes a ele. Estas três representações dariam três tamanhos diferentes para o mesmo grafo.

Quanto ao número de operações, a ambigüidade está na noção de operação: a soma de dois inteiros de vários dígitos, deve ser considerada como uma ou várias operações? Veremos logo que essas diferenças são irrelevantes do ponto de vista da Complexidade Computacional, cujos resultados são invariantes, dentro de certos limites, por mudanças de alfabeto, representação ou computador.

Definimos então a *função de eficiência* g_A do algoritmo A, $g_A : \mathbf{N} \to \mathbf{N}$ como

$$g_A(n) = \max_{|S| \leq n} \theta_A(S) \tag{7}$$

ou seja, o número máximo de operações necessárias para resolver, com o algoritmo A, uma instância de tamanho não superior a n.

A Análise do Pior Caso

Observemos que a medida de eficiência dada por (7) considera, para cada tamanho n, o "pior caso", ou seja, a instância de tamanho n para a qual o algoritmo A é menos eficiente. Poder-se-ia esperar que se um algoritmo A_1 é mais eficiente, na análise do "pior caso", do que um algoritimo A_2, o

mesmo aconteceria na maioria das instâncias, ou seja, que se

$$g_{A_1}(n) < g_{A_2}(n) \qquad \text{para todo} \qquad n \geq n_0$$

então

$$\theta_{A_1}(S) < \theta_{A_2}(S)$$

para, pelo menos, um grande número de instâncias S. Embora isso seja verdade para muitos algoritmos, existe uma exceção muito importante, o problema PL, que é, de longe, o problema de otimização combinatória mais frequentemente usado nas aplicações.

Em 1947, G. B. Dantzig [3] introduziu o algoritmo **Simplex,** que é, até hoje, o mais utilizado para resolver PL. **Simplex** começa achando um vértice $\overline{x}$ do poliedro definido pelas restrições (2)-(3). Examinaram-se então os vértices x^j vizinhos (no sentido geométrico usual) de $\overline{x}$. Se $c^T\overline{x} \leq c^T x^j$ para todo j, $\overline{x}$ é a solução. Se algum x^j é "melhor" que $\overline{x}$, troca-se $\overline{x}$ por x^j e repete-se o procedimento até achar um vértice "melhor" (no sentido da função objetivo) que todos os seus vizinhos. A linearidade da função objetivo garante que um vértice que é "melhor" que todos os seus vizinhos é ótimo, ou seja, é também "melhor" que qualquer outro vértice.

Fica assim demonstrada a finitude do **Simplex.** No entanto, não há garantia de que não seja necessário percorrer todos os vértices até atingir o ótimo. Na experiência prática isso não acontece: 40 anos de uso do algoritmo em milhões de casos mostra que o número de vértices examinados é, quase sempre, da ordem de $m \log n$. Como a passagem de um vértice a outro requer aproximadamente m^2 operações, o resultado empírico indica que, em geral (ou seja, para todos os casos "da vida real") $\theta_A(S) \leq |S|^4$.

Mas (7) pede o máximo, não um valor médio. Em 1972 Klee e Minty [11] construiram uma família de instâncias de dimensão crescente cujo poliedro, em $\mathbf{R}^n$, é um 'hipercubo' ligeiramente deformado. Com $m = 2n$ hiperplanos, obtem-se um poliedro com 2^n vértices v^k $(1 \leq k \leq 2^n)$ onde v^k_j é igual a ϵ_{kj} ou a $1 - \epsilon_{kj}$, com ϵ_{kj} positivo e pequeno. A função objetivo é simplesmente $-x_1$, e o ótimo é o ponto $(1, 0, \ldots, 0)$. Começando em $(0, 0, \ldots, 0)$, uma escolha inteligente dos ϵ_{kj} leva o **Simplex** a passar por todos os vértices do poliedro até chegar ao ótimo. Isto significa mais de 2^n operações. Como o tamanho da instância em $\mathbf{R}^n$ e menor ou igual que βn para alguma constante β (o tamanho leva em conta os coeficientes das $2n$ equações que determinam os hiperplanos) chega-se a conclusão de que $g_A(n)$ é maior que qualquer polinômio em n, ou seja o **Simplex** é não polinomial.

Abriu-se assim a procura de um *algoritmo polinomial* para PL, que poucos acreditavam que existisse (ou seja *um algoritmo A para PL com*

$g_A(n) \leq p(n)$ *para um dado polinômio* p). Em 1979 o jovem matemático soviético Kachyan [8] supreendeu ao mundo com o seu agora célebre algoritmo "elipsoidal", baseado em trabalhos anteriores de Schor [13]. Pela sua importância para o problema '$P = NP$' vale a pena descrever mais detalhadamente o algoritmo, que se baseia em 4 fatos:

a) O problema PL pode ser 'reduzido' ao problema de resolver um sistema de desigualdades lineares. Consideremos o problema de programação linear (4)-(5) e estudemos o seguinte sistema de desigualdades em variáveis $(x, y) \in \mathbf{R}^{n+m}$ (A^T indica a transposta de A):

$$\begin{aligned} Ax &\leq b && (8)\\ -A^T y &\leq -c && (9)\\ -b^T x &+ c^T y \leq 0 && (10) \end{aligned}$$

O importante teorema de dualidade da programação linear ([4], Cap. 6) nos diz que, dada qualquer solução (x^*, y^*) de (8)-(10), x^* é a solução ótima de (4)-(5). Assim para resolver uma instância de PL basta achar um ponto x que satisfaz um sistema de desigualdades do tipo:

$$\overline{A}x \leq \overline{b} \qquad (11)$$

cujo tamanho, como é fácil ver, é o dobro do tamanho da instância (4)-(5) de PL.

b) O conjunto de soluções de (11), se não é vazio, é um poliedro. O seu volume (no sentido usual em $\mathbf{R}^n$) pode ser 0 ou maior que 0. É fácil, perturbando o lado direito $\overline{b}$ de (11) achar um problema

$$\overline{A}x \leq \widehat{b} \qquad (12)$$

tal que:

I) o conjunto de soluções de (12) ou é vazio ou tem volume maior que 0;

II) o conjunto de soluções de (12) é vazio se e só se o conjunto de soluções de (11) é vazio;

III) a partir de uma solução de (12) é possível construir uma solução de (11).

O tamanho da instância (12) é apenas maior que o tamanho de (11).

c) Dado um problema do tipo $Ax \leq b$ tal que o seu conjunto de soluções é ou vazio ou de volume positivo, podem-se determinar números V e v tais que:

I) se o conjunto de soluções do sistema é não vazio, seu volume é maior que v;

II) os vértices do poliedro estão numa esfera (n-dimensional) de volume V;

III) o número $\ell n(\frac{V}{v})$ está limitado superiormente por um múltiplo fixo do tamanho do problema (independente de instância).

Os números v e V dependem dos coeficientes da matriz A e do vetor b.

d) Dado um elipsoide E em $\mathbf{R}^n$ de volume $u(E)$ e um hiperplano que passa pelo seu centro, é possível construir um elipsoide E' que contém qualquer uma das metades de E definidas pelo hiperplano, tal que $u(E') \leq (1-\epsilon)u(E)$, onde ϵ é uma constante que só depende da dimensão n de forma que $[\ell n(\frac{1}{1-\epsilon})]^{-1}$ está limitado por um polinômio em n de grau 1.

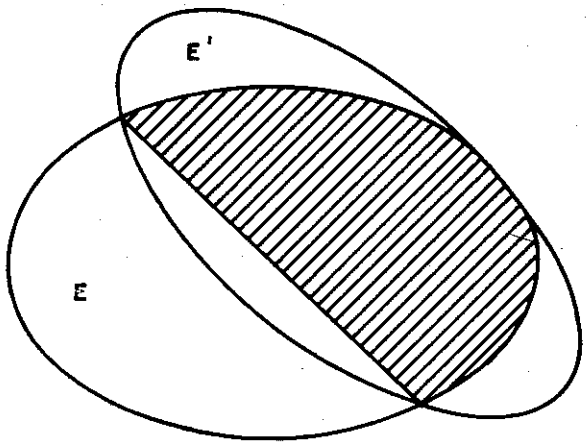


Figura 3

Os detalhes das demonstrações de a) - d) podem ser visto em [12], Cap. 8. Com estes fatos o algoritmo de Kachyan é extremamente simples. Dada a

instância de PL, realizam-se as reduções indicadas em a) e b), determinando v, V é a esfera E^1 de volume V. Se o centro α^1 de E^1 satisfaz (12) o problema acabou, e a solução de instância original de PL se obtêm de α^1 como indicam a) e b).

Se o centro α^1 não satisfaz (12), α^1 viola uma das restrições:

$$a^{iT}x \leq b_i$$

ou seja que os vértices do poliedro estão na região

$$R = E^1 \cap \{X \in \mathbf{R}^n : a^{iT}x \leq b_i\},$$

contida na região $R' = E^1 \cap \{x : a^{iT}x \leq a^{iT}\alpha^1\}$.

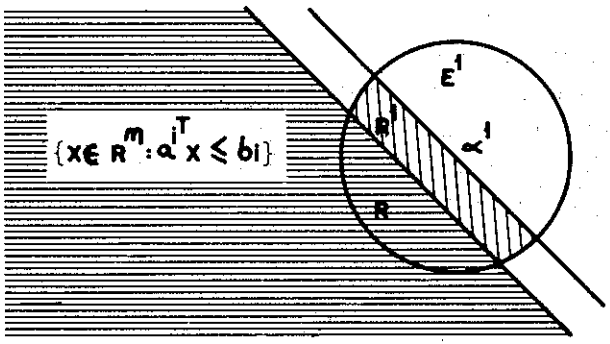


Figura 4

Usamos agora d) para construir um elipsoide E^2 que contém R' tal que

$$u(E^2) \leq (1-\epsilon)u(E^1) = (1-\epsilon)V.$$

E^2 contém todos os vértices do poliedro, evidentemente. O procedimento é repetido com E^2 e seu centro α^2: ou α^2 é a solução de (12) ou geramos E^3, que contém os vértices do poliedro, e tal que

$$u(E^3) \leq (1-\epsilon)u(E^2) \leq (1-\epsilon)^2V.$$

Após k passos, ou uma solução foi achada, ou temos o elipsoide E^{k+1} que contém os vértices e tal que $u(E^{k+1}) \leq (1-\epsilon)^kV$. Mas se $k \geq \ell n(V/v)/\ell n(1/1-\epsilon) = k^*$, resulta

$$u(E^{k+1}) \leq v.$$

Da definição de v e b) deduzimos que o poliedro é vazio, ou seja que a instância de PL não tem solução. Portanto o algoritmo faz no máximo k^* iterações. Cada iteração (construção do novo elipsoide) requer, aproximadamente n^2 operações. Segue-se que $\theta_A(S) \leq |S|^4$, tendo em conta que k^* é linear, segundo vimos, no tamanho do problema. Logo $g_A(n) \leq n^4$ para o algoritmo de Kachyan.

A publicação deste algoritmo provocou grandes expectativas, no sentido de que ele fosse substituir o **Simplex**. A prática mostrou, no entando, que ele era muito mais lento que o **Simplex** em *todos* os problemas *reais* (naturalmente, ele é melhor que o **Simplex** para o exemplo de Klee e Minty com n grande). Que aconteceu? O algoritmo elipsoidal é polinomial de grau menor ou igual que quatro. O **Simplex**, embora não polinomial em geral, funciona de fato, fora os contraexemplos patológicos, como um algoritmo polinomial de grau menor que 4, melhor que o elipsoidal. Ou seja, para quase todas as instâncias o **Simplex** tem uma "performance" enormemente melhor que o que indica o seu limite teórico para o "pior caso". O algoritmo elipsoidal tem um limite teórico melhor, mas todas as instâncias ficam muito perto do 'pior caso'.

Tudo isto mostra a fraqueza de $g_A(n)$ como indicador de eficiência. Pareceria melhor, por exemplo, estudar o 'caso médio' em vez do 'pior caso' (por exemplo com:

$$g'_A(n) = \frac{1}{r(n)} \sum_{|S| \leq n} \theta_A(S)$$

onde $r(n)$ e o número de instâncias de tamanho menor ou igual a n). O motivo pelo qual a teoria continua se baseando em $g_A(n)$ é puramente prática: $g'_A(n)$ é impossível de estimar para quase todos os algoritmos. Pelo contrário, como veremos logo, é relativamente fácil achar bons limites superiores para $g_A(n)$, como ja fizemos para o algoritmo elipsoidal.

Estimação da Eficiência

Aceitando então, por falta de melhor opção, a medida de eficiência dada pelo 'pior caso', vejamos como estimar $g_A(n)$ para alguns algoritmos em problemas de grafos.

a) **Caminho mais curto.** Para simplificar, vamos supor que o grafo é completo, adicionando as arestas faltantes com um valor $d(e)$ suficientemente

grande como para que nunca sejam usadas (basta tomar

$$d(i,j) \geq \sum_{e \in E} d(e)$$

para os pares de nós $(i,j) \in E$). Para um grafo com n nós d é então uma matriz $D = \{d_{ij} \in \mathbf{Z}^{n \times n}\}$, definindo também $d_{ii} = 0$. Definamos agora $D^1 = D$, observando que d_{ij} pode ser interpretado como a longitude do caminho mais curto de i a j num passo só, ou seja sem usar nós intermediários. Seja então D^k a matriz que contém, no lugar d^k_{ij}, a longitude do caminho mais curto para ir de i a j usando não mais de $k-1$ nós intermediários. E fácil observar que vale a seguinte fórmula:

$$d^{k+1}_{i,j} = \min_{\ell \in V \; \ell \neq j} \{d^1_{i\ell} + d^k_{\ell j}\} \tag{13}$$

ou seja, o caminho mais curto de i a j em não mais de k passos é obtido escolhendo um nó ℓ como primeiro intermediário, usando o caminho mais curto de ℓ a j em não mais de $k-1$ passos, e tomando a menor das distâncias resultantes. D^{n-1} da então a longitude dos caminhos mais curtos que usam não mais de $n-2$ nós intermediários. Como um caminho não pode ter nós repetidos, D^{n-1} da a longitude dos caminhos mais curtos sem mais. Se

$$d^{n-1}_{ij} > \sum_{e \in E} d(e),$$

então não existe caminho entre i e j no grafo original.

O algoritmo requer o cálculo de $n-1$ matrizes de n^2 elementos cada uma, e para calcular cada elemento é preciso o mínimo de $n-1$ somas, ou seja, precisam-se $n^2(n-1)$ comparações e $n^2(n-1)^2$ somas. Em consequência

$$\theta_A(S) \leq \beta |S|^4 \quad \text{e} \quad g_A(n) \leq \gamma n^4$$

para certas constantes β, γ. Notamos que é necessário guardar também numa matriz E^k_{ij} o vértice onde se atinge o mínimo em (13) para poder então reconstruir o caminho mais curto.

Este não é o melhor algoritmo para CMC: o de Floyd-Warshall ([6], [16]), que modifica o algoritmo dado substituindo (13) por um mínimo entre só dois números, é de ordem n^3; o algoritmo de Dijkstra [5] é de ordem n^2.

b) **Árvore espalhadora mínima**. Tomemos o seguinte algoritmo: a árvore espalhadora vai sendo construida adicionando um arco cada vez. O

primeiro é o arco mais curto do grafo. E quando já temos k arcos, cujos nós formam o conjunto S^k, adicionamos o arco mais curto entre os que têm um nó em S^k e ou outro fora de S^k. O algoritmo acaba quando não há mais arcos a serem adicionados. Se o S^k final não contém todos os nós, não há árvore espalhadora (o grafo não é conexo). Se acabamos com $S^k = V$, o conjunto de arcos formado é uma árvore espalhadora mínima: o conjunto de arcos gerado é conexo, por que todo arco adicionado está ligado ao conjunto anterior (ele tem um nó em S^k); não tem ciclos, porque se os tivesse ao adicionar o último arco de um ciclo, tal arco teria *seus dois nós* no S^k já formado, e espalha todos os nós porque acaba com $S^k = V$. Só falta ver que sua longitude é mínima. A prova, bastante simples, pode ser vista em [12], pp. 272-274. Num grafo com n nós, o algoritmo requer $n-1$ passos, cada um dos quais consite de não mais que n comparações. Portanto $g_A(n) \leq \beta n^2$, para alguma constante β.

Novamente, este não é o melhor algoritmo disponível; o de Yao [17] requer $\gamma m \log\log n$ operações para um grafo com m arcos e n nós (γ é uma constante). Estes exemplos, mostram que para certos problemas, como CMC e AEM, existem algoritmos eficientes, com $g_A(n) \leq n^k$, k pequeno. A discussão do algoritmo de Kachyan nos diz que PL entra também nesta categoria. Para os outros quatro exemplos, no entanto, a situação é bem pior. Mais de quarenta anos de pesquisas não consequiram algoritmos que melhorassem substancialmente os algoritmos 'ingênuos' antes mencionados. De fato, para nenhum desses quatro problemas (CH, CV, SAT e PLI) conhecemos hoje algoritmos com $g_A(n)$ limitados por um polinômio em n (a possível exceção seria o trabalho de Swart, discutido no final do artigo).

A Classe de Problemas P

Tentaremos agora uma classificação, não de algoritmos, mas de problemas: *Um problema Q pertence à classe P se existe um algoritmo polinomial A para resolve-lo, ou seja tal que $g_A(n)$ está limitado por um polinômio em n.*

A existência deve ser entendida aqui no sentido matemático não construtivo, ou seja incluíndo algoritmos ainda não descobertos. Assim, da discussão anterior resulta que CMC, AEM e PL estão em P, mas não sabemos ainda se CH, CV, SAT e PLI estão ou não em P, pois não conhecemos nenhum algoritmo polinomial para eles.

O estudo da classe P é um objetivo básico da teoria da Complexidade Computacional. Em geral, asssocia-se P com a classe de problemas 'eficien-

temente resolúveis', isto é, considera-se que os algoritmos polinômiais são eficientes e os não polinômiais são ineficientes.

Se nos mantemos no critério de eficiência dado pelo 'pior caso', com as ressalvas discutidas, é evidente que os algoritmos não polinomiais são ineficientes, mas será que podemos considerar eficientes *todos* os algoritmos polinomiais? Não é claro que um algoritmo com $g_A(n) = n^{80}$ ou $g_A(n) = 10^{100}n^2$ possa ser chamado de eficiente., Não seria melhor estudar, por exemplo, a classe P' de problemas com algoritmos A tais que $g_A(n) \leq p(n)$ com p um polinômio de grau menor que 5 e coeficientes menores que 1000?

Do ponto de vista prático certamente sim, mas existem pelo menos três razões para fazer de P o alvo principal da teoria. As duas primeiras são empiricas:

a) Até agora, os algoritmos polinomiais descobertos tem grau e coeficientes 'razoáveis'.

b) Mesmo nos casos em que o primeiro algoritmo polinomial achado não foi tão eficiente como seria de desejar, a experiência mostra que rapidamente surgiram para o mesmo problema algoritmos progressivamente melhores. No caso de PL o algoritmo polinomial mas não muito eficiente de Kachyan foi sucedido em 1984 pelo de Karmarkar [9] que é também polinomial mas aparentemente comparável em eficiência com o **Simplex** nos problemas 'reais' (ou seja longe do 'pior caso'). O mesmo aconteceu com CMC e AEM.

A terceira razão é de índole teórica, e mais importante.

c) A classe P possui certas propriedades de 'fecho' que a fazem passível de tratamento matemático. Isso não acontece com a classe P' sugerida, sobre a qual provavelmente não seja possível provar nunhum teorema interessante. É mais um exemplo (como a questão do 'pior caso') onde é preferível uma definição que, embora não modelizando da melhor forma possível o objeto conceitual (neste caso 'eficiência'), permite no entanto construir uma teoria rigorosa e elegante, com teoremas, demonstrações, etc.

A primeira dessas propriedades é a invariância a respeito de alfabetos, representações e 'computadores' (ou seja critérios para contar operações). Para quaisquer alfabetos, representações e computadores 'razoáveis', um algoritmo será polinomial para todos eles ou não o será para nenhum. No entanto, um algoritmo poderia ter $g_A(n) = n^4$ num computador e $g_A(n) = n^5$ noutro. Isso justifica a ambigüidade nas definições de 'tamanho' e 'número de operações' e faz a teoria mais robusta e livre de 'impurezas'.

Outra propriedade decorre do fato de que o conjunto de polinômios é fechado por composição de funções (polinômio de polinômio é polinômio).

A consequência disso no nosso caso é a seguinte: Suponhamos que para o problema Q (com conjunto de instâncias T) temos o algoritmo polinomial A com $g_A(n) \leq p(n)$. Agora consideramos outro problema Q' (com conjunto de instâncias T') e achamos uma função $\varphi : T' \to T$ bijetiva tal que:

I) da solução de uma instância $S' \in T'$ é possível obter a solução de $\varphi(S') \in T$ e reciprocamente;

II) existe um polinômio p' tal que $|\varphi(S')| \leq p'(|S'|)$ para todo $S' \in T'$. (Uma tal função φ define uma *redução polinomial* de Q' a Q).

Então Q' também é polinomial: para cada instância S' aplicamos A a $\varphi(S')$, e o algoritmo resultante A' é polinomial com $g_{A'}(n) \leq p(p'(n))$.

É evidente que a redução polinomial, ferramenta básica da Complexidade Computacional, não funciona se nos limitamos o algoritmos polinomiais com grau previamente limitado.

A Classe de Problemas NP e a Questão $P = NP$

Definida a classe P, podemos nos perguntar quais os problemas que poderiamos esperar que pertençam a P. Isto nos leva a definir, informalmente, a classe NP como a formada por *aqueles problemas para os quais, dada uma solução, existe um algoritmo polinomial B que confirma que o objeto dado é uma solução.*

Intuitivamente, $P \subset NP$, porque se somos capazes de achar uma solução x em tempo polinomial com um algoritmo A, esse mesmo algoritmo confere que x é solução e fornece o algoritmo B requerido na definição da classe NP. Ou seja, é mais difícil achar uma solução que confirmar que uma solução é mesmo solução do problema. Para problemas com solução única o argumento dado é rigoroso. Num caso com soluções múltiplas a questão é mais delicada e necessitamos uma definição mais formal da classe NP para provar que $P \subset NP$.

Além do apelo intuitivo, a classe NP tem a vantagem de conter quase todos os problemas interessantes de otimização combinatória. Aliás, a demonstração de que um problema está em NP é geralmente simples. Para CH, o algoritmo B limita-se a conferir que o conjunto dado de arestas é um ciclo que passa por todos os nós, o que claramente pode ser feito em tempo polinomial. Para SAT, B simplesmente verifica que a correspondência dada da à fórmula Φ o valor V. Para os outros exemplos é preciso refinar tecnicamente a definição de NP, mas feito isso é fácil provar que eles estão em

NP. Claro que, uma vez que $P \subset NP$, podemos afirmar sem necessidade de uma prova direta que CMC, AEM e PL estão em NP. Também estão em NP CH, PLI e quase todos os problemas de interesse (as letras NP segnificam 'nondeterministically polinomial', em referência a uma definição anterior da classe, hoje abandonada).

Podemos, agora sim, dar sentido a questão do título do artigo: $P = NP$, que afirma simplesmente que a classe NP, em princípio maior que P, é igual a P, ou seja que *para todo problema que possui um algoritmo polinomial para verificar soluções é possível encontrar um algoritmo polinomial que acha as soluções.*

As consequências de uma resposta afirmativa à conjetura $P = NP$ são significativas: existiria um algoritmo eficiente (com as ressalvas anteriores sobre a identificação de algoritmo polinômial com algoritmo eficiente) para todo problema em NP, por exemplo para o Caixeiro Viajante.

Em princípio, pareceria que provar a conjetura é uma tarefa quase impossível: deveriamos encontrar (ou demonstrar a existência de) algoritmos polinomiais para cada problema da classe NP!

Felizmente, um teorema de Cook de 1971 [2] faz a tarefa muito mais simples (em termos). O teorema diz:

Todo problema da classe NP pode ser reduzido polinomialmente ao problema de **Satisfatibilidade**.

Ou seja toda instância S de qualquer problema em NP pode ser convertida numa fórmula lógica (de tamanho menor ou igual que um polinômio em $|S|$) de forma que a fórmula é satisfatível se e só se a instância tem solução, e de uma correspondência de valores de verdade que faz a fórmula verdadeira é possível reconstruir uma solução de S.

Antes de dar uma idéia da prova do teorema, vejamos a sua principal conseqüência: SAT é "maximalmente difícil" em NP porque se achássemos um algoritmo polinômial para SAT, mediante a redução do teorema ele serviria também para todo problema em NP, ou seja: se $SAT \in P$ então $P = NP$.

Quanto à prova (ver [12] pp. 356-358), ela requer uma formalização da noção de computador, por exemplo a de Turing [15]. A ideia é descrever com uma fórmula lógica todos os passos do algoritmo verificador de soluções B (que existe porque o problema está em NP) usando variáveis lógicas $x_{it\alpha}$ com significados de tipo: "a posição i da memória no instante t contém o símbolo α do alfabeto", ou "a posição i da memória passou a conter o símbolo α no instante t". Supõe-se que as primeiras posições contêm no instante 0 a codificação da solução a ser checada e que quando o algoritmo

B confirma que a solução é correta escreve um símbolo particular em certa posição. A fórmula só é verdadeira se o algoritmo foi corretamente executado e acabou escrevendo o símbolo especial. Como o algoritmo B é, por hipótese, polinomial no tamanho de S, o número de posições de memória e de instantes usados é também polinomial em $|S|$, do jeito que o número de variáveis lógicas é polinomial em $|S|$ (sendo que o alfabeto tem tamanho préfixado). Em definitivo o tamanho da fórmula é polinomial em $|S|$. Se achamos uma correspondência de valores de verdade que faz a fórmula verdadeira, olhando o valor das variáveis lógicas associadas com as primeiras posições de memória no instante 0 podemos reconstruir os símbolos que as ocuparam. Como sabemos que a execução acabou usando o símbolo especial, esses simbolos representam uma solução da instância S. Reciprocamente, dada uma solução de instância S, damos às variàveis lógicas os valores que resultam de fazer funcionar o algoritmo B a partir dessa solução e obtemos uma correspondência de valores de verdade que faz a fórmula verdadeira.

Que acontece agora se achamos que pela sua vez SAT se reduz polinomialmente a outro problema $Q \in NP$? Nesse caso resulta que se $Q \in P$ então $P = NP$: tomamos um Q' qualquer em NP, reduzimos Q' polinomialmente a SAT via o teorema de Cook, logo SAT a Q, e sob a hipótese $Q \in P$ temos o algoritmo polinomial A para Q que podemos usar para resolver Q' em tempo polinomial. Vemos que um tal Q partilha a posição privilegiada de SAT em NP: ele é "maximalmente difícil" e todo problema em NP se reduz polinomialmente a ele.

A Classe de Problemas NP-Completos

Surge então naturalmente a idéia de definir uma nova classe: a dos problemas NP-completos. Q é NP-completo se:

I) $Q \in NP$;

II) *todo* $Q' \in NP$ *se reduz polinomialmente a* Q.

Obtemos assim a seguinte representação gráfica de NP:

Existem outros problemas NP-completos além de SAT? Pouco tempo depois do teorema de Cook, Karp [10] demonstrou que CH é NP-completo. Para isso, dada qualquer fórmula lógica Φ, ele construi um grafo cujos ciclos hamiltonianos estão em correspondência biunívoca com as correspondências de valores de verdade que fazem Φ verdadeira.

Sabendo que CH é NP-completo é evidente que CV é NP-completo: a cada instância de CH num grafo G fazemos corresponder a instância de CV no grafo completo G' com os mesmos nós que G, e definimos os comprimentos como 1 para os arcos que estão em G e um número grande (maior que o número de nós) para os que não estão em G. Se a instância de CV tem solução de comprimento $n-1$ o ciclo mais curto em G' corresponde a um ciclo hamiltoniano em G. Reciprocamente, um ciclo hamiltoniano em G produz um ciclo de comprimento mínimo em G'.

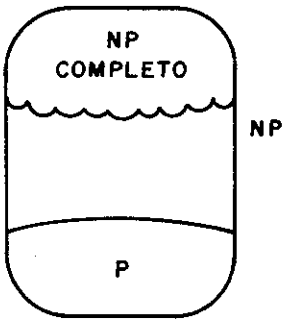


Figura 5

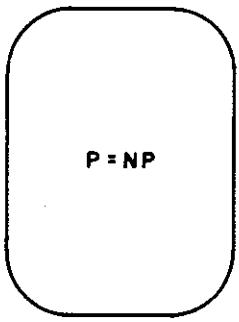


Figura 6

Compreende-se então a importância da invariância de P por reduções polinomiais: se fazemos uma sequência de reduções polinomiais a partir de um problema em NP e acabamos num problema em P, o problema de partida está em P. Se aliás o problema de partida é NP-completo, então $P = NP$.

Veremos logo que CH se reduz polinomialmente a PLI, ou seja que PLI é também NP-completo.

O método das reduções polinomiais produziu uma verdadeira orgia de demonstrações de NP-completeza, cujo número cresceu exponencialmente a partir do teorema de Cook. O livro de Garey e Johnson [7] dá em 1979 uma lista de mais de 600 problemas NP-completos, e aparentemente a lista passa hoje de 4000.

Isto se deveu não somente a euforia dos otimizadores combinatórios (que depois de 40 anos de procurar sem sucesso algoritmos eficientes para esses 4000 problemas tinham a opção de engrossar seus "curriculum" não resolvendo-os mas provando que eles são "maximalmente difíceis") se não

também à propriedade mencionada: como para ver que um problema é NP-completo basta reduzi-lo polinomialmente a qualquer um que já está na lista, quanto maior a lista mais chances de achar nova redução e mais "papers" com provas de NP-completeza.

De fato a Figura 5 ficou com a faixa intermediária cada vez mais magra: para quase todos os problemas interessantes ou bem achou-se um algoritmo polinômial, e acabaram em P, ou bem demonstrou-se que eram "maximalmente difíceis", e foram parar a NP-completo. Como vimos, PL ingressou em P em 1979 graças a Kachyan.

Tudo isto pareceria simplificar notavelmente a prova da conjetura $P = NP$. Bastaria encontrar um algoritmo polinomial para qualquer um dos 4000 problemas da lista de NP-completos, e a Figura 5 colapsaria a Fig. 6.

Infelizmente nunhum problema NP-completo parecia estar em P, de modo que a quase totalidade dos especialistas acreditaram que a conjectura era falsa. Para refutá-la seria suficiente provar que algum dos 4000 problemas NP-completos não está em P (do qual resultaria imediatamente que não existe algoritmo polinomial para nenhum deles) mas até hoje não temos ferramentas para demonstrar, dado um problema, que *não* existe um algoritmo de certo tipo para resolve-lo.

De fato, em outubro de 1986 a situação era a seguinte: a lista de problemas NP-completos seguia crescendo e os poucos audaciosos dispostos a atacar frontalmente a conjetura tentavem refuta-la.

P = NP?

Alors Malherbe vint! E. R. Swart, um pesquisador não muito conhecido (embora com vários trabalhos na área) da ainda menos conhecida Universidade de Guelph (Ontario, Canadá), anunciou ao mundo que $P = NP$. O seu trabalho [14] foi recebido com ceticismo, não somente por ir contra a intuição da maioria, se não também porque o caminho escolhido por Swart parecia condenado ao fracasso, após ter sido muito explorado nos anos 50 e 60. Swart afirma ter reduzido polinomialmente CH a PL. Como $PL \in P$ (via Kachyan ou Karmarkar), $CH \in P$, logo $P = NP$, já que sabemos que CH é NP-completo. Observe-se que esta forma de redução é inversa à das demonstrações de NP-completeza: acha-se um algoritmo eficiente para um problema reduzindo-o a outro conhecidamente mais fácil.

As consequências espetaculares do resultado de Swart, se verdadeiro

(tanto as boas, como a existência de algoritmos eficientes para 4000 problemas até agora intratáveis, quanto as ruins, porque se $P = NP$ a classe "NP-completo" perde todo o seu sentido e as demonstrações de NP-completeza tornam-se absolutamente irrelevantes, sendo que todos os problemas de NP ficam igualmente fáceis, ou difíceis) levaram uma multidão de especialistas a se debruçarem sobre o "technical report" de Swart. Os resultados ainda não são definitivos e entramos no terreno do boato. Diz-se que o "paper" foi enviado a 12 "referees". Dois de eles teriam-no considerado correto, outro teria achado um erro rapidamente corrigido pelo próprio Swart. Os outros dez, até janeiro de 1987, não teriam se pronunciado. Em março, novo boato: a prova teria um erro irremediável. Enquanto assenta-se a poeira (e pode demorar) tentemos dar uma descrição não detalhada do caminho seguido por Swart.

Queremos encontrar um ciclo hamiltoniano no grafo G com n vértices. Damos aos arcos (i,j) de G longitude 1, e para os pares de nós (i,j) não adjacentes em G criamos um arco fictício com longitude $d_{ij} > n$, obtendo o grafo completo G'. É claro que um ciclo hamiltoniano de longitude n em G' é um ciclo hamiltoniano em G. Agora imaginamos que o problema CH consiste em transportar um objeto ao longo do ciclo hamiltoniano. Definimos, para cada par de nós (i,j) uma variável x_{ij} que toma só valores 0 ou 1. Damos a x_{ij} a seguinte interpretação: se $x_{ij} = 1$ o objeto vai de i a j pelo arco (i,j), caso contrário $x_{ij} = 0$. Consideramos a seguinte instância S_1 do problema PLI:

$$S_1 : \min f(x) = \sum_{j=1}^{n} \sum_{i=1}^{n} d_{ij} x_{ij} \tag{14}$$

sujeito a

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad (1 \leq j \leq n) \tag{15}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad (1 \leq i \leq n) \tag{16}$$

$$x_{ij} \geq 0 \qquad (1 \leq i, j \leq n) \tag{17}$$

$$x_{ij} \in \mathbf{Z} \qquad (1 \leq i, j \leq n). \tag{18}$$

Segundo (15) o objeto transportado entra exatamente uma vez em cada nó, e segundo (16) sai exatamente uma vez de cada nó. Isto porque de (15), (16) e (17) resulta que cada variável x_{ij} fica entre 0 e 1, e usando (18) ela

deve ser 0 ou 1. Então para cada j um dos x_{ij} é 1 e os outros são 0, e para cada i um dos x_{ij} é 1 e os outros são 0. Representa corretamente esta instância de PLI o problema de CH em G?

É claro que dado um ciclo hamiltoniano em G, dando valor 1 às variáveis x_{ij} associadas com arcos do ciclo é 0 às restantes obtemos um vetor x que satisfaz (15)-(18) tal que $f(x) = n$, ou seja uma solução de S_1. Infelizmente a recíproca não funciona, como mostra o seguinte exemplo, onde os arcos marcados são os que tem $x_{ij} = 1$.

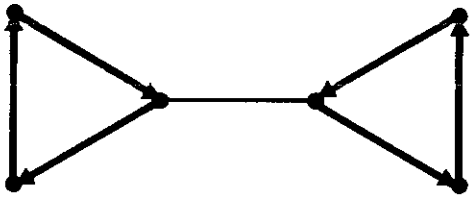


Figura 7

O objeto entra e sai uma vez de cada nó, mas não fazendo um só ciclo, se não dois assim chamados "sub-tours". O grafo da Figura 7 não possui ciclos hamiltonianos mais o vetor x resultante dos arcos marcados é uma solução de S_1. Para remediar o problema consideramos variáveis x_{ijk}, interpretando $x_{ijk} = 1$ como: "o k-ésimo arco do ciclo hamiltoniano va de i a j'. Formulamos S_2:

$$\max \sum_{j=1}^{n} \sum_{i=1}^{n} d_{ij} \sum_{k=1}^{n} x_{ijk} \tag{19}$$

sujeito a

$$\sum_{j=1}^{n} \sum_{i=1}^{n} x_{ijk} = 1 \qquad (1 \leq k \leq n) \tag{20}$$

$$\sum_{k=1}^{n} \sum_{j=1}^{n} x_{ijk} = 1 \qquad (1 \leq i \leq n) \tag{21}$$

$$\sum_{k=1}^{n} \sum_{i=1}^{n} x_{ijk} = 1 \qquad (1 \leq j \leq n) \tag{22}$$

$$\sum_{i=1}^{n} (x_{ijk} - x_{j,i,k+1}) = 0 \qquad (1 \leq j, k \leq n) \tag{23}$$

$$x_{ijk} \geq 0 \qquad (1 \leq i, j, k \leq n) \tag{24}$$

$$x_{ijk} \in \mathbf{Z} \qquad (1 \leq i, j, k \leq n) \tag{25}$$

(para $k = n$ identificamos, nas restrições (23), $x_{j,i,k+1}$ com $x_{j,i,1}$). Novamente um ciclo hamiltoniano em G produz uma solução x de S_2 mas agora, graças as restrições (23), toda solução x de S_2 poduz um ciclo hamiltoniano em G. As restrições (23), chamadas "sub-tour breaking constraints" impedem os "sub-tours" como os da Figura 7, exigindo que o ciclo passe por n arcos.

Portanto, temos uma correta redução de CH a PLI. (Incidentalmente, isto prova que PLI é NP-completo). Mas não ganhamos muito, porque não se conhece nenhum algoritmo polinomial para PLI. Que acontece se passamos a PL, para o qual temos os algoritmos polinomiais de Kachyan e Karmarkar? Basta para isso eliminar as restrições (18) em S_1, e (25) em S_2 ou seja aceitar valores reais para as variáveis.

Em S_1 nada muda. Devido a estrutura particular da matriz das restrições (15)-(16) (é uma matriz *unimodular*, ver [12] pp. 316- 318) todos os vértices do poliedro definido por (15)-(17) tem componentes inteiros. Então, procurando uma solução real de (14)-(17) que seja um vértice do poliedro, recebemos, de graça, uma solução inteira (já vimos ao discutir PL que se existe solução, existe uma que é vértice do poliedro). Infelizmente, como mostra a Figura 7, tal solução inteira pode não representar um ciclo hamiltoniano em G.

As soluções de (19)-(24), se são inteiras, representam sim um ciclo hamiltonianos em G, mas a matriz de (20)-(23) não é unimodular. É possível que certos vértices do poliedro dado por (20)-(24) tenham componentes não inteiros. Então a solução da versão PL de S_2 (ou seja, sem (25)) pode não ser inteira, incluindo valores fracionários entre 0 e 1. Isso não nos permite reconstruir um ciclo porque não saberiamos interpretar $x_{ijk} = 1/2$, por exemplo.

Tudo isto era bem conhecido faz 25 anos (ver [4] pp. 545-547). A primeira grande idéia de Swart consistiu em estudar as soluções fracionárias de S_2. Ele provou que qualquer solução x^* de S_2 que é vértice do poliedro dado por (20)-(24), é uma média de soluções inteiras, ou seja

$$x^* = \sum_{\ell=1}^{r} \alpha_\ell y^\ell$$

com

$$\alpha_\ell \geq 0, \qquad \sum_{\ell=1}^{r} \alpha_\ell = 1$$

e

$$y^\ell_{ijk} = 0 \qquad \text{ou} \qquad 1.$$

Os y^ℓ podem ser identificados com caminhos no grafo G, mas como as restrições (20)-(24) operam sobre x^* e não sobre y^ℓ, não é claro que todos os y^ℓ devam ser ciclos hamiltonianos. De fato Swart dá um exemplo onde a solução x^* de S_2 escreve-se como

$$x^* = \frac{1}{2}y^1 + \frac{1}{2}y^2,$$

mas y^1 é um ciclo em G com $n-1$ nós (ou seja um a menos) e y^2 não é ciclo porque repete um nó. Swart recorre ao mesmo mecanismo usado para passar de S_1 a S_2: introduzir mais equações e variáveis com mais sub-índices. (Ele toma variáveis de tipo x_{ijkrs} que vale 1 se (i,j) é o k-ésimo arco de um ciclo hamiltoniano que começa no nó 1 é o r-ésimo arco num ciclo hamiltoniano que começa no nó s). Novo contra-exemplo, mais equações, mais sub-índices até chegar a uma formulação S_5 com variáveis com até 8 sub-índices e 22 grupos de equações, no lugar dos quatro grupos (20)-(23) de S_2.

Ele prova então seu teorema crucial:

Se a instância S_5 de Programação Linear não tem solução, então o grafo G não possui caminhos hamiltonianos. Se S_5 tem solução, então toda solução que é vértice do poliedro escreve-se como uma média de soluções inteiras, uma das quais, pelo menos, representa um ciclo hamiltoniano em G.

O tamanho de S_5, embora monstruoso, é ainda polinomial em n. Temos então uma redução polinomial de CH a PL. Dada uma instância de CH, escrevemos a sua instância associada S_5 de PL, e resolvendo S_5 com um algoritmo polinomial para PL, resulta que $CH \in P$. Como CH é NP-completo, fica provado que $P = NP$.

Assim para resolver em tempo polinomial uma instância de um problema qualquer de NP (por exemplo CV) reduzimos o problema a SAT (via a fórmula lógica do teorema de Cook), SAT a CH, CH a PL via a formulação S_5 de Swart, e resolvemos S_5 em tempo polinomial com o algoritmo de Kachyan ou o de Karmarkar.

É bom notar que embora estes algoritmos, a diferença do **Simplex**, não produzem necessariamente soluções que são vértices do poliedro (como o

teorema de Swart requer) eles podem sera modificados para produzir vértices sem perder a polinomialidade. Acontece também a feliz circunstância de ser o teorema de Swart construtivo: a partir da solução de S_5 em variáveis reais é possível achar a solução inteira que representa o ciclo hamiltoniano em G.

A discussão sobre a validade do resultado de Swart centra-se na prova do seu teorema crucial, que é inevitavelmente bastante complicada porque depende da presença de cada um dos 22 grupos de equações na formulação S_5.

Mas, quais são as conseqüências para a otimização combinatória do resultado de Swart? Se ele é verdadeiro, como já explicamos, desaparece a distinção entre P e NP, arrastando no caminho a categoria intermediária "NP-completo". Agora todo problema é P. Mas aqui nos confrontamos com a ambigüidade da identificação "algoritmo polinomial-algoritmo eficiente". Se fazemos a redução de um problema NP via SAT, CH e PL como indicamos, podemos acabar com uma instância de PL de tamanho polinomial, mas gigantesco (S_5 sozinho já é grande de mais) e com algoritmos com $g_A(n) = n^{80}$ ou coisas semelhantes. A tarefa então, é a de encontar reduções melhores, talvez diretamente de cada problema a PL, baixando aos poucos o grau do polinômio, como já foi feito para outros problemas em P. O próprio Swart conjetura que a formulação S_5 é altamente redundante; muitos dos 22 grupos de equações, que foram adicionados para facilitar a prova do teorema, poderiam ser eliminados sem alterar o resultado.

A direção do trabalho em otimização combinatória viraria 180^0, passando de procurar resultados "limitativos" (provas de NP-completeza, que dão limites *inferiores* à dificuldade do problema) à busca de resultados "construtivos" (introdução de novos algoritmos eficientes).

Se pelo contrário, como muitos acreditam, a prova do teorema de Swart está errada, fica a tarefa de corrigi-la, ou tentar outra por caminhos semelhantes. De fato, a plausibilidade de "$P = NP$", descartada até Swart por quase todos os especialistas, ganhou muitos pontos, e sem dúvida muitos dos que antes tentavam refutá-la preferirão daqui em diante procurar prová-la.

Observação. [12] apresenta magnificamente todo o material sobre o tema até 1981. Também é recomendável [7], de leitura muito agradável. O autor pode fornecer aos interessados o "preprint" de Swart [14], ainda não publicado, sem a página 16.

Referências

[1] BERGE, C.: *Graphes et hypergraphes.* Dunod (1970).

[2] COOK, S. A.: The complexity of Theorem Proving Procedures. *Proc. 3rd. ACM Symp. on the Theory of Computing 151-158* (1971).

[3] DANTZIG, G. B.: Maximization of a Linear Function of Variables Subject to Linear Inequalities. In *Activity Analysis of Production and Allocation* (Ed. T.C. Koopmans) 339-347. John Wiley (1951).

[4] DANTZIG, G. B.: *Linear Programming and Extensions.* Princeton Univ. Press (1962).

[5] DIJKSTRA, E. W.: A Note on Two Problems in Connexion with Graphs. Numerische Mathematik 1: 269-271 (1959).

[6] FLOYD, R. W.: Algorithm 97: Shortest Path. Comm. A.C.M. 5, 6: 345 (1962).

[7] GAREY, M. R., JOHNSON, D. S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Co. (1979).

[8] KACHYAN, L. G.: A Polynomial Algorithm for Linear Programming. Doklady Akad. Nauk. USSR 5, 244: 1093-1096 (1979).

[9] KARMARKAR, N.: A New Polynomial Algorithm for Linear Programming. Combinatorica 4: 375-395 (1984).

[10] KARP, R. M.: Reducibility among Combinatorial Problems. In *Complexity of Computer Calculations* (Ed. R.E. Miller e J.W. Thatcher) 85-103. Plenum Press (1972).

[11] KLEE, V., MINTY, G. J.: How Good is the **Simplex** Algorithm? In *Inequalities III* (Ed. O. Shisha) 159-175. Academic Press (1972).

[12] PAPADIMITRIOU, C. H., STEIGLITZ, K.: *Combinatorial Optimization: Algorithms and Complexity:* Prentice-Hall (1982).

[13] SCHOR, N. Z.: Utilization of the Operation of Space Dilatation in the Minimization of Convex Functions. Kibernetica 6: 6-12 (1970).

[14] SWART, E. R.: P = NP Preprint. University of Guelph (1986).

[15] TURING, A. M.: On Computable Numbers, with an Application to the Entscheidungs Problem. Proc. London Math. Soc. 2, 42: 230-265 (1936)

[16] WARSHALL, S.: A Theorem on Boolean Matrices. Journal A.C.M. 9, 1: 11-12 (1962).

[17] YAO, A. C. C.: An $0(|E|\log\log|V|)$ Algorithm for Finding Minimum Spanning Trees. Inf. Proc. Letters 4: 21-25 (1975).

Instituto de Matemática Pura e Aplicada — CNPq
Estrada Dona Castorina 110 — Jardim Botânico
22.460 Rio de Janeiro, RJ