Projeto e Análise de Algoritmos

Flávio L. C. de Moura

06 de julho de 2023

Dizemos que um problema é *NP-completo*, i.e. que está na classe NPC, se está na classe NP e é tão difícil quanto qualquer problema em NP. Ao mostrarmos que um problema é NP-completo, não estamos tentando provar a existência de um algoritmo eficiente, mas concluir que a existência de um tal algoritmo é improvável. De fato, estamos concluindo sobre o quão difícil ele é, e não sobre quão fácil como fizemos até então. Portanto, um algoritmo eficiente provavelmente não existe para este problema.

Uma importante questão em aberto é quando P é ou não um subconjunto próprio de NP, o que corresponde ao problema "P vs NP" citado anteriormente. Problemas NP-completos normalmente são considerados intratáveis dada a grande quantidade de problemas NP-completos já estudados, e sem solução polinomial encontrada.

Propriedade interessante: Se algum problema NP-completo puder ser resolvido em tempo polinomial então qualquer problema em NP terá solução polinomial.

Seria uma surpresa encontrar uma solução polinomial para algum (e portanto para todos) destes problemas. Neste sentido, se um problema é NP-completo então isto pode ser visto como uma evidência da sua intratabilidade. Neste caso, algoritmos aproximados devem ser considerados, ao invés de soluções rápidas e exatas.

Por que até hoje ninguém conseguiu encontrar soluções polinomiais para estes problemas? Não sabemos, talvez porque elas simplesmente não existam, ou porque elas estejam baseadas em princípios ainda desconhecidos. Qualquer problema em P está também em NP porque pode ser verificada em tempo polinomial pelo mesmo algoritmo que a decide em P sem a necessidade de utilizar certificados.

Definição 1. *Uma linguagem $L \subseteq \{0,1\}^*$ é dita* ***NP-completa*** *se:*

1. $L \in NP$;
2. $L' \leq_P L, \forall L' \in NP$.

Denotamos por NPC a classe das linguagens NP-completas.

Se uma linguagem L satisfaz a propriedade 2 acima, mas não necessariamente a propriedade 1, dizemos que L é **NP-difícil**.

Como NP-completude consiste em mostrar quão difícil é um problema, utilizamos a redução polinomial na outra direção para mostrar que um problema é NP-completo:

Para mostrarmos que um problema B não possui solução polinomial, consideremos um problema A, para o qual sabemos não existir solução polinomial. Suponha também que temos um algoritmo de redução que transforma instâncias de A em instâncias de B em tempo polinomial. Agora, se existisse uma solução polinomial para B então poderíamos construir uma solução polinomial para A como acima, contradizendo a suposição de que A não possui solução polinomial.

Lema 2. *Se L é uma linguagem tal que $L' \leq_P L$ para algum $L' \in NPC$, então L é NP-difícil. Se adicionalmente, $L \in NP$ então $L \in NPC$.*

Prova. Como L' é NP-completo, então $\forall L'' \in$ NP, temos que $L'' \leq_P L'$, e por hipótese temos que $L' \leq_P L$. Logo. $L'' \leq_P L$, o que mostra que L é uma linguagem NP-difícil. Agora, se $L \in$ NP então, por definição temos que $L \in$ NPC. □

O lema anterior nos dá um método para mostrar que uma linguagem L é NP-completa:

1. Prove que $L \in$ NP;
2. Escolha uma linguagem L' que seja NP-completa;
3. Descreva um algoritmo que computa uma função f que mapeia cada instância x de L' em uma instância $f(x) \in L$;
4. Prove que $x \in L'$, se e somente se, $f(x) \in L$;
5. Prove que o algoritmo que computa f é polinomial

Os passos de 2-5 mostram que L é NP-difícil.

Exemplo 3. *$SAT = \{\langle\varphi\rangle : \varphi$ é uma fórmula booleana satisfatível$\}$.*

Podemos determinar em tempo exponencial se uma dada fórmula booleana φ contendo n variáveis é satisfatível: basta checar cada uma das 2^n possíveis valorações para as variáveis de φ. Nenhum algoritmo polinomial é conhecido para SAT. O teorema a seguir, mostra que é muito improvável que tal algoritmo exista.

O teorema a seguir é conhecido como o *teorema de Cook-Levin*:

Teorema 4. *$SAT \in NPC$.*

Teorema 5. *$3\text{-}SAT \in NPC$.*

Prova. 1. 3-SAT $\in$ NP: Dada uma designação de valores para as variáveis de uma 3-FNC fórmula (certificado), o algoritmo de verificação substitui cada variável pelo valor dado e avalia a expressão. Se a expressão resulta em 1 então o certificado é válido e a fórmula é satisfatível.

2. SAT $\leq_P$ 3-SAT.

□

Teorema 6. *$CLIQUE \in NPC$.*

Prova. 1. CLIQUE $\in$ NP;

2. 3-SAT $\leq_P$ CLIQUE

□

Uma cobertura de vértices de um grafo $G = (V, E)$ (não-dirigido) é um subconjunto $V' \subseteq V$ tal que $(u, v) \in E$ então $u \in V'$ ou $v \in V'$ (ou ambos!). O tamanho de uma cobertura de vértices é o seu número de vértices, ou seja, $|V'|$. O problema da cobertura de vértices consiste em encontrar uma cobertura de vértices de tamanho mínimo em um grafo. Reescrevendo este problema como um problema de decisão, queremos determinar se um grafo possui uma cobertura de vértices de tamanho dado k:

VERTEX-COVER $= \{\langle G, k\rangle : G$ possui uma cobertura de vértices de tamanho $k\}$

Teorema 7. *$VERTEX\text{-}COVER \in NPC$.*

Prova. 1. VERTEX-COVER $\in$ NP;

2. CLIQUE $\leq_P$ VERTEX-COVER.

□

Um caminho Hamiltoniano em um digrafo G é um caminho de um vértice u para um vértice v que visita todos os vértices de G exatamente uma vez. O problema de decisão HAMPATH consiste em, dado um digrafo G, responder se G possui um caminho Hamiltoniano ou não.

HAMPATH $= \{\langle G, u, v\rangle : G$ é um digrafo com um caminho Hamiltoniano de u para $v\}$

Teorema 8. *HAMPATH* $\in$ *NPC.*

Prova. 1. HAMPATH $\in$ NP;

2. 3-SAT $\leq_P$ HAMPATH

□