Limits to Computation

Problemas Tratáveis (P): podem ser rsolvidos em tempo polinomial como O(p(n))

Exemplo de Problemas Tratáveis:
- Ordenação de uma lista de números.
- Encontrar o menor caminho em um grafo usando o algoritmo de Dijkstra.

Problemas Intratáveis (NP-completo e pior): não podem ser resolvidos em tempo polinomial

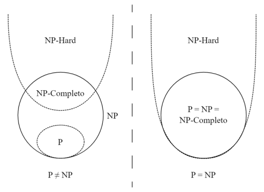
Exemplo de Problemas Intratáveis:
- O problema do Caixeiro Viajante (Traveling Salesman Problem): encontrar o caminho mais curto que visita uma série de cidades e retorna à cidade de origem.
- O problema de satisfatibilidade booleana (SAT): determinar se existe uma atribuição de valores verdade/falso que satisfaça uma dada expressão booleana.
- Problemas de coloração de grafos: determinar o número mínimo de cores necessárias para colorir um grafo de modo que nenhum vértice adjacente compartilhe a mesma cor.

Complexidade de redução
P -> a classe de problemas de decisão que podem ser resolvido em tempo polinomial por algoritmos determinísticos
Problemas de decisão: com respostas sim/não

Alguns problemas que não são de decisão, podem ser reduzidos a uma série de problemas de decisão
Original: qual é o caminho mais curto entre você e w?
Redução
Existe um caminho de peso ≤ d?
Existe um caminho de peso ≤ d − 1?

Algoritmos Não Determinísticos
- Classe NP: Conjunto de problemas de decisão verificáveis em tempo polinomial.
- Algoritmos Não Determinísticos: Operam em duas etapas - geração não determinística de uma solução e verificação determinística.
- Linguagem Não Determinística Hipotética: Inclui saltos não determinísticos para explorar diferentes caminhos de execução.

Problema do k-Clique
Descrição do Problema: Dado um grafo não direcionado G=(V,E) sem pesos, e um número k∈N tal que k≤ | V | , existe um subgrafo completo de G com k vértices?
- Etapa Não Determinística: Adivinhar um subconjunto de k vértices de G.
- Etapa Determinística: Verificar se o subconjunto adivinhado forma um subgrafo completo, ou seja, verificar se todos os pares de vértices no subconjunto estão conectados por arestas.



Para mostrar que um problema de decisão DDD é NP-completo, geralmente seguimos duas etapas principais:
1. Mostrar que D está em NP:
   - Isso significa que uma solução candidata para D pode ser verificada em tempo polinomial.
2. Mostrar que todo problema em NP é redutível a D em tempo polinomial:
   - Devido à transitividade da redução polinomial, isso geralmente é feito mostrando que um problema já conhecido como NP-completo pode ser reduzido a D em tempo polinomial.

Teoremas Importantes
- Teorema de Cook-Levin:
  - Afirma que o problema de satisfatibilidade booleana (SAT) é NP-completo.
- Teorema de Richard Karp:
  - Afirma que o problema 3-SAT (uma versão específica do SAT) é NP-completo.

Problema 3-SAT
- Descrição do Problema:
  - Dada uma fórmula lógica proposicional α\alphaα em forma normal conjuntiva (CNF), onde cada cláusula tem no máximo três literais, existe uma atribuição de valores verdadeiros/falsos que satisfaça α?

Redução Polinomial de 3-SAT para k-Clique
Para mostrar que 3-SAT é redutível ao problema do k-clique, onde k=k =k= número de cláusulas em α:
1. Representação da Fórmula α:
   - Suponha que $\alpha=c1\wedge c2\wedge\ldots\wedge cm$, onde cada cic_ici é uma cláusula com até três literais.
2. Construção do Grafo G=(V,E):
   - Vértices (V): Cada literal v_{i,j} representa um literal da cláusula cic_ici.
   - Arestas (E): Conecte os vértices v_{i,j} e v_{k,l}, l se i≠k e v_{i,j} ≠ no(v_{k,l}), não são literais contraditórios e vêm de cláusulas diferentes).

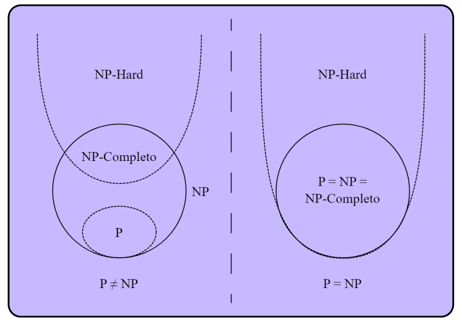
Exemplo de Redução
Considere a fórmula $\alpha=(x\vee y)\wedge(\neg x\vee y)\wedge(\neg x\vee\neg y)$.
1. Vértices do Grafo:
   - V={(c1,x),(c1,y),(c2,¬x),(c2,y),(c3,¬x),(c3,¬y)}
2. Arestas do Grafo:
   - Conecte (c1,x) com (c2,¬x) e (c2,y), mas não com (c3,¬x)
   - Similarmente, conecte outros vértices de acordo com as regras.

Reduções
Reduções são técnicas usadas para transformar um problema em outro. Elas são particularmente úteis na teoria da complexidade computacional para mostrar que a dificuldade de um problema é pelo menos tão grande quanto a de outro. Se um problema AAA pode ser reduzido a um problema BBB de forma eficiente (usando uma redução polinomial, por exemplo), então qualquer solução para BBB pode ser usada para resolver AAA.

The Theory of NP-Completeness
A teoria da NP-completude trata de problemas que são, de certa forma, "os mais difíceis" dentro da classe NP (Nondeterministic Polynomial time). Problemas em NP são aqueles para os quais uma solução proposta pode ser verificada em tempo polinomial. Um problema é NP-completo se é tanto em NP quanto tão difícil quanto qualquer outro problema em NP, no sentido de que qualquer problema em NP pode ser reduzido a ele em tempo polinomial.

NP-Completeness Proofs
Provas de NP-completude geralmente envolvem dois passos principais:
1. Mostrar que o problema está em NP, ou seja, que uma solução proposta pode ser verificada em tempo polinomial.
2. Mostrar que o problema é pelo menos tão difícil quanto qualquer outro problema em NP, geralmente reduzindo um problema conhecido NP-completo ao problema em questão.

17.2.3 Coping with NP-Complete Problems
Lidar com problemas NP-completos pode envolver várias abordagens, incluindo heurísticas, aproximações, algoritmos probabilísticos e redução do problema a instâncias menores ou mais simples que podem ser resolvidas eficientemente. Em muitos casos, soluções exatas não são práticas, então técnicas que encontram soluções suficientemente boas em um tempo razoável são usadas.



Impossible Problems (Problemas Impossíveis)
17.3.1 Uncountability
A incontabilidade refere-se à ideia de que certos conjuntos, como o conjunto de todas as funções computáveis, são tão grandes que não podem ser contados (enumerados) de forma completa usando números inteiros. Isso leva a implicações sobre os limites do que pode ser computado.
17.3.2 The Halting Problem Is Unsolvable
O problema da parada (Halting Problem) é um exemplo clássico de um problema não computável. Ele pergunta se é possível determinar, para um dado programa e uma entrada, se o programa irá parar ou executar indefinidamente. Alan Turing provou que não existe um algoritmo geral que resolve o problema da parada para todos os possíveis programas e entradas, demonstrando que alguns problemas estão além da capacidade dos algoritmos de resolverem.