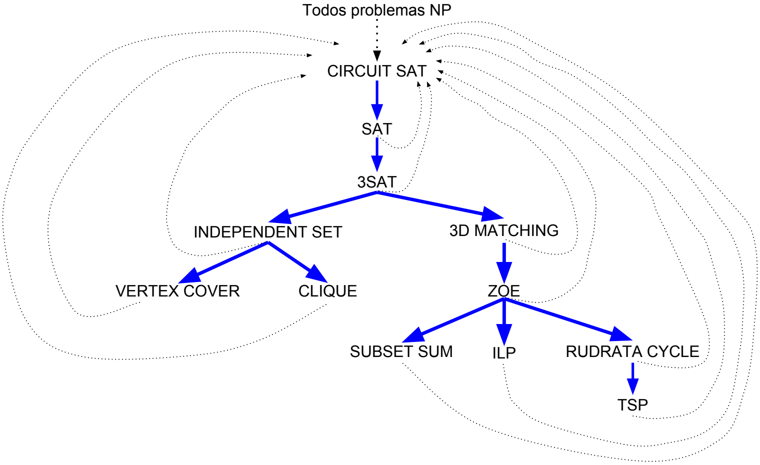
Redução de Cook-Levin e Considerações Finais

André Vignatti

DINF- UFPR

Fechando o Ciclo de Reduções

Nós reduzimos o SAT para diversos problemas de busca, como mostra a figura:



Fechando o Ciclo de Reduções

Agora vamos completar o ciclo e mostrar que **qualquer** dos problemas de busca apresentados – e, de fato, qualquer problema em **NP** – reduz para **SAT**.

Na verdade, vamos mostrar que todos os problemas em **NP** podem ser reduzidos a uma generalização do **SAT** a que chamamos de **CIRCUIT SAT**.

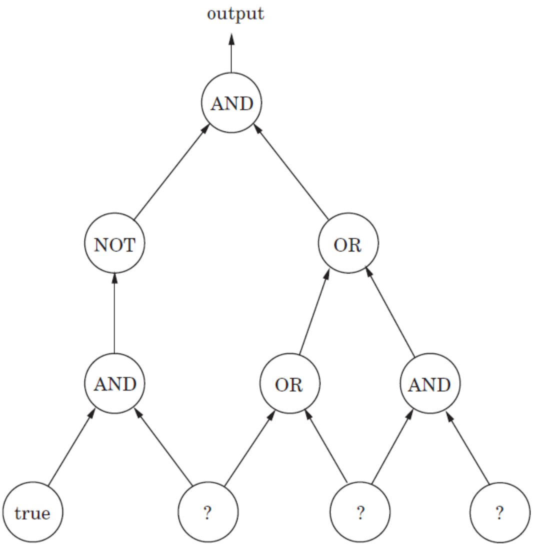
CIRCUIT SAT

No **CIRCUIT SAT** temos um grafo direcionado acíclico (dag) cujos vértices são portas lógica de **cinco tipos** diferentes:

- portas **AND** e portas **OR** (com duas entradas e uma saída).
- portas **NOT** (com uma entrada e uma saída).
- portas de **entrada conhecida** têm valores fixos **TRUE** ou **FALSE** (não tem entrada).
- portas de **entrada desconhecida** têm valores rotulados como "**?**" (não tem entrada).

Um dos vértices é designado como a porta de saída.

CIRCUIT SAT - Exemplo



CIRCUIT SAT

O problema de busca é definido como:

Problema CIRCUIT SAT:

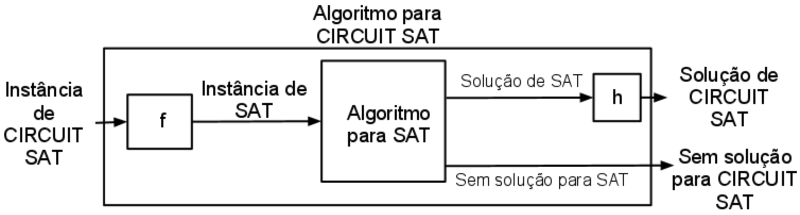
Dado um circuito, encontrar uma atribuição para as **entradas desconhecidas** tal que a porta de saída seja **TRUE**, ou dizer que tal atribuição não existe.

Na figura anterior, por exemplo,

- **(TRUE, FALSE, TRUE)** avaliaria em **FALSE**.
- **(FALSE, TRUE, TRUE)** avaliaria em **TRUE**.

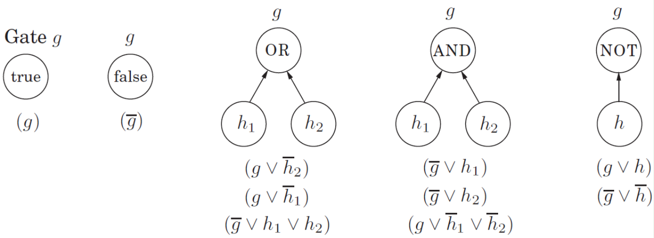
CIRCUIT SAT é uma **generalização** de SAT (fácil de ver, mas não é o que queremos)

Queremos ir na outra direção: mostrar que CIRCUIT SAT ⇒ SAT.



Redução:

1. Para cada porta g, criamos uma variável g, modelando o efeito da porta usando algumas cláusulas:

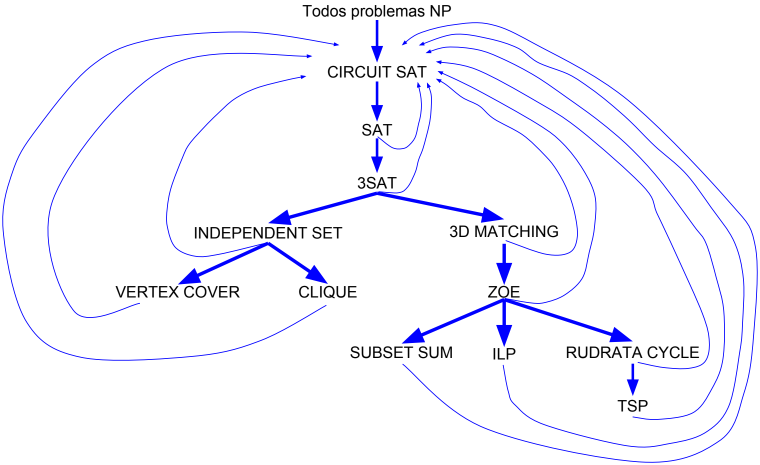


(Você percebe que essas cláusulas, de fato, forçam exatamente o efeito desejado?)

2. Se g é a porta de saída, força o valor **TRUE**, adicionando a cláusula (g).

(Exercício: mostre que a redução é válida (use os dois passos, como fizemos nas reduções anteriores))

Agora que sabemos **CIRCUIT SAT** reduz ao **SAT**, podemos voltamos ao nosso trabalho principal:



Qualquer Problema em NP $\Rightarrow$ SAT

Teorema (Cook-Levin 1971-1973):

Todos os problemas de busca (ou seja, problemas **NP**) se reduzem ao CIRCUIT SAT.

Suponha que A é um problema em **NP**.

Parece muito difícil fazer a redução para o CIRCUIT SAT: **não sabemos quase nada sobre A!**

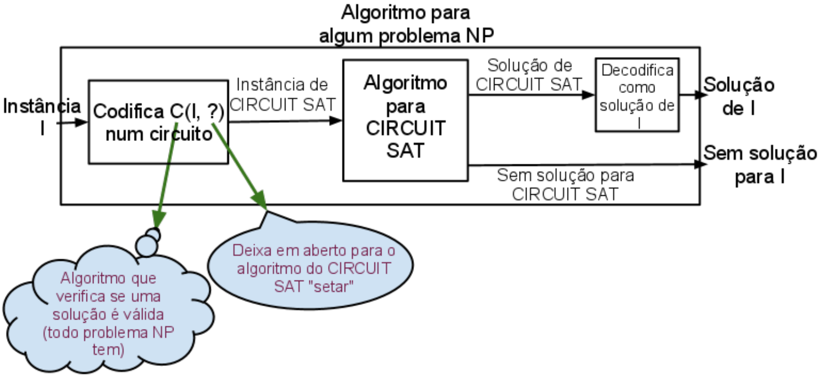
Qualquer Problema em NP $\Rightarrow$ SAT

Mas sabemos uma coisa!

Como A é um problema de busca, então podemos testar eficientemente uma solução:

- Sejam I e S, respectivamente, uma **instância** e **solução** de A
- **Existe** um algoritmo C, que recebe como entrada I e S e diz se S é ou não solução da instância I.
- Além disso, o algoritmo C é de tempo polinomial.

Então, nossa redução terá essa cara:



Qualquer Problema em NP $\Rightarrow$ SAT

Como transformar numa instância de CIRCUIT SAT?

Fácil! É só transformar a instância I, a solução S e o algoritmo C em **um circuito**!

É possível fazer isso?

SIM! Não mostraremos os detalhes, mas pense que no fundo todo computador é formado por portas lógicas.

Ou seja, todo algoritmo (polinomial) pode ser implementado por (um número polinomial de) portas lógicas (ou seja, pelo **CIRCUIT SAT**).

Qualquer Problema em NP $\Rightarrow$ SAT

Um problema:

Mas S **NÃO** é conhecido! Como é o circuito de algo que não conheço?

- Fácil! Usamos entradas desconhecidas (aquelas com "**?**") e deixamos que o **CIRCUIT SAT** descubra os valores dela.

- Ideia: dado $C(I, ?)$ (transformado num circuito), queremos saber qual $?$ satisfaz $C(I, ?) = TRUE$.

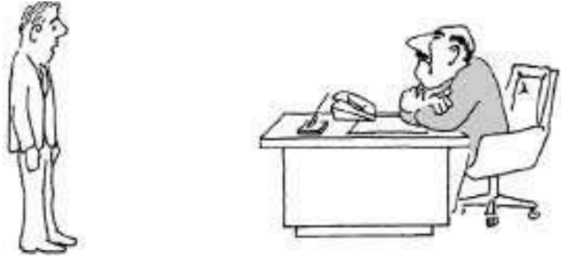
- Mas é exatamente isso que CIRCUIT SAT faz!

Resumindo: Para qualquer instância I do problema A,

1. Podemos **construir** em tempo polinomial um circuito que simula A.
2. As entradas conhecidas (fixas) são os bits de I.
3. As entradas desconhecidas ("?") são os bits de S.
4. A saída do circuito é **TRUE** $\Leftrightarrow$ as entradas "**?**" assumem o valor de uma solução S de I.

A redução está completa.

Cartoon Famoso do Garey & Johnson - 1979



"Não consigo encontrar um algoritmo eficiente, acho que sou muito burro..."

Cartoon Famoso do Garey & Johnson - 1979

"Não consigo encontrar um algoritmo eficiente, porque tal algoritmo não existe!"

Cartoon Famoso do Garey & Johnson - 1979

"Não consigo encontrar um algoritmo eficiente, mas todas essas pessoas famosas também não conseguem."

Na Prática...

No mundo corporativo e acadêmico existem milhões de problemas NP-Completos.

Mundo Corporativo

- Alguma hora um desses problemas aparece.
- Pela falta de conhecimento, geralmente não se sabe que o problema é difícil.
- Soluções heurísticas são usadas.

Na Prática...

Mundo Acadêmico

- Muitas áreas estão focadas em resolver problemas difíceis:
  - Redes e Sistemas Distribuídos
  - Banco de Dados
  - Inteligência Artificial
  - Processamento de Imagens
  - Sistemas Embarcados e Projeto de Hardware
- Diversas abordagens para achar soluções.
- Resultados teóricos e/ou experimentais.

Versão de Otimização

Na prática, muitas vezes queremos resolver a versão de otimização de um problema de busca.

Se P = NP, então é possível usar a versão de busca para resolver a versão de otimização (já vimos isso)

Mas como acredita-se que P $\neq$ NP, talvez seja melhor atacar direto a versão de otimização.

Versão de Otimização

NP-Difícil

A versão de otimização de um problema de busca NP-Completo está na classe NP-Difícil (não entraremos nos detalhes da definição de NP-Difícil).

Nos problemas NP-Completos, se encontrássemos uma solução para um, poderíamos usar para todos os outros.

Essa característica não é mantida nos problemas NP-Difícil.

Para otimizar um problema NP-Difícil, deve-se buscar **soluções específicas** para cada problema.

Métodos de Soluções de Problemas Difíceis

Alguns métodos para buscar soluções em problemas NP-Completos ou NP-Difíceis:

Heurísticas: Tenta achar soluções boas, mas sem garantir qualidade da solução e tempo de execução.

Algoritmos de Aproximação: Acha soluções boas rapidamente, garantindo qualidade da solução.

Backtracking e Branch-and-Bound: Acha soluções ótimas, mas só para instâncias pequenas ou médias (pior caso é exponencial).

Programação Linear Inteira: Acha soluções ótimas, mas só para instâncias pequenas ou médias (pior caso é exponencial).

Algoritmos de Parâmetro Fixo: Acha soluções ótimas, fixando um parâmetro pequeno da instância, e executando em tempo exponencial neste parâmetro.

Considerações Finais

Há muito mais para brincar:

- CI339 : Complexidade Computacional
- CI335 : Tópicos em Algoritmos
- CI801 : Tópicos em Inteligência Artificial (Metaheurísticas)
- MA??? : Pesquisa Operacional 1 e 2
- CIxxx : Algoritmos Aleatorizados???
- CIxxx : Algoritmos de Aproximação???
- ARG - Grupo de Pesquisa em Algoritmos