O Teorema de Cook-Levin

Prova da NP-completude do problema SAT.

Márcio Nicolau

2025-11-10

Table of contents

List of Figures

Objetivos da Aula

- Entender por que um "primeiro" problema NP-Completo é necessário.
- Apresentar o Problema da Satisfatibilidade Booleana (SAT) como o problema canônico.
- Desmistificar a estratégia de prova do Teorema de Cook-Levin: simular uma computação NP com uma fórmula booleana.
- Compreender a importância fundamental deste teorema como a "pedra de Roseta" da NP-Completude.

Conteúdo

A Necessidade de um Ponto de Partida

Nas aulas anteriores, estabelecemos a receita para provar que um problema é NP-Completo: 1. Mostrar que o problema está em NP. 2. Reduzir um problema **já conhecido** como NP-Completo a ele.

Isso nos leva a uma questão fundamental: como o *primeiro* problema foi provado NP-Completo? Não havia um problema conhecido para reduzir. Era preciso uma prova do zero, que demonstrasse que **qualquer** problema em NP poderia ser reduzido a este "problema primordial".

Este é o papel do **Teorema de Cook-Levin**, que estabelece a NP-Completude do Problema da Satisfatibilidade Booleana (SAT).

O Problema da Satisfatibilidade Booleana (SAT)

Antes de mergulhar na prova, vamos definir precisamente o nosso protagonista.

> **i** Definição: SAT
>
> **SAT** é o problema de decidir se uma dada fórmula booleana ϕ é **satisfazível**.
>
> - Uma **fórmula booleana** é uma expressão com variáveis booleanas (que podem ser V ou F), e os operadores $\wedge$ (E, conjunção), $\vee$ (OU, disjunção) e $\neg$ (NÃO, negação).
> - Uma fórmula é **satisfazível** se existe pelo menos uma atribuição de valores (Verdadeiro/Falso) para suas variáveis que torna a fórmula inteira Verdadeira.
> - A linguagem formal é: SAT $= \{\langle\phi\rangle \mid \phi$ é uma fórmula booleana satisfazível$\}$

Exemplo:

- $\phi_1 = (x \vee y) \wedge (\neg x \vee \neg y)$ é satisfazível. Atribuição: $x = V, y = F$.
- $\phi_2 = x \wedge \neg x$ é insatisfazível. Nenhuma atribuição pode torná-la verdadeira.

O Teorema de Cook-Levin

Este é um dos teoremas mais importantes da ciência da computação.

> **!** Teorema de Cook-Levin (1971)
>
> O problema **SAT é NP-Completo**. (Sipser, 2012, p. 312)

A prova consiste em duas partes, seguindo a definição de NP-Completude.

Parte 1: SAT está em NP

Esta é a parte fácil. Precisamos mostrar que, se alguém nos der um "certificado", podemos verificar a solução rapidamente.

- **Certificado**: Uma atribuição de valores Verdadeiro/Falso para todas as variáveis da fórmula ϕ.
- **Verificador**: Um algoritmo que recebe $\langle \phi, \text{atribuição} \rangle$ e faz o seguinte:
  1. Substitui cada variável em ϕ pelo seu valor na atribuição.
  2. Avalia a expressão booleana resultante.
  3. Se o resultado for Verdadeiro, aceita. Caso contrário, rejeita.
- **Análise de Tempo**: A avaliação de uma fórmula booleana é muito rápida, levando tempo linear no tamanho da fórmula. Portanto, a verificação é polinomial. **SAT** $\in$ **NP**.

Parte 2: SAT é NP-Difícil

Esta é a parte genial e o coração do teorema. Precisamos mostrar que **qualquer** linguagem L em NP pode ser reduzida em tempo polinomial a SAT ($L \leq_p$ SAT).

Esboço da Prova da NP-Dificuldade de SAT

A Grande Ideia: Vamos mostrar como transformar a **computação** de uma Máquina de Turing Não-Determinística (NTM) em uma **fórmula booleana gigante**. A fórmula será construída de tal forma que ela será satisfazível se, e somente se, a NTM aceitar sua entrada.

Estratégia Geral:

1. Seja L uma linguagem qualquer em NP.
2. Por definição, existe uma NTM, N, que decide L em tempo polinomial, digamos $p(n) = n^k$.
3. Nosso objetivo é construir uma redução f que, para qualquer entrada w de tamanho n, produz uma fórmula booleana ϕ_w tal que:

$$w \in L \iff \phi_w \text{ é satisfazível}$$

   Além disso, a construção de ϕ_w deve levar tempo polinomial.

Simulando a Computação com Variáveis Booleanas

A computação de uma NTM pode ser visualizada como uma **tabela de computação** (ou *tableau*). Cada linha representa a configuração da máquina em um passo de tempo. A tabela tem tamanho $n^k \times n^k$.

Vamos criar variáveis booleanas para descrever esta tabela:

- `x_i,j,s`: É Verdadeiro se a célula j no passo de tempo i contém o símbolo s.
- `h_i,j`: É Verdadeiro se o cabeçote da TM está sobre a célula j no passo de tempo i.
- `q_i,k`: É Verdadeiro se a TM está no estado q_k no passo de tempo i.

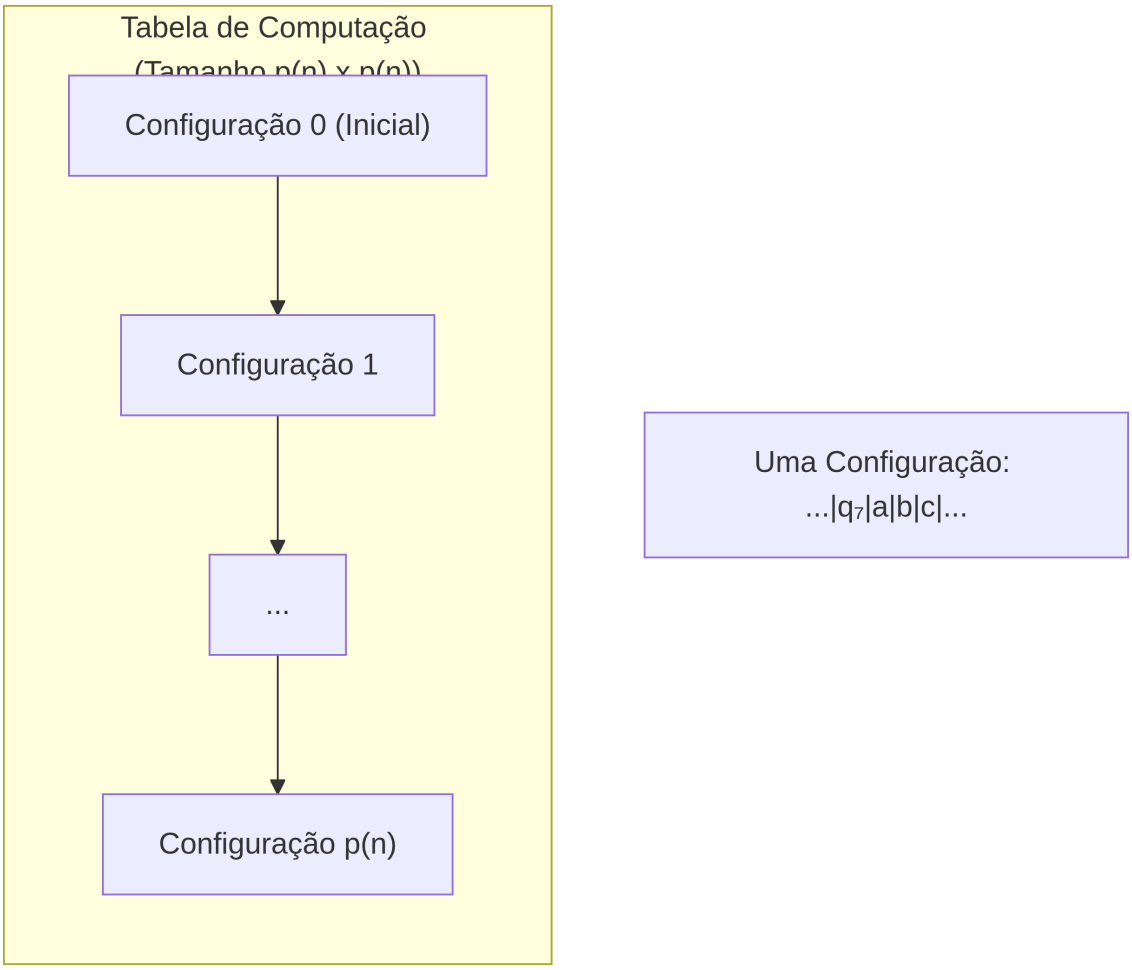

Figure 1: Uma tabela de computação. Cada linha é uma 'foto' da TM em um instante de tempo.

Construindo a Fórmula ϕ_w

A fórmula gigante ϕ_w será a conjunção (E) de quatro sub-fórmulas, cada uma garantindo uma propriedade da computação:

$\phi_w = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}$

1. ϕ_{cell} **(Consistência da Célula)**: Garante que cada célula da tabela contém exatamente um símbolo em cada passo de tempo.
   - *Para cada célula (i, j)*: "(a célula contém o símbolo s_1) OU (contém s_2) OU …" E "NÃO (contém s_1 E s_2)", etc.
2. ϕ_{start} **(Configuração Inicial)**: Garante que a primeira linha da tabela corresponde à configuração inicial da NTM com a entrada w.
   - "No tempo 0, o estado é q_0." E "No tempo 0, o cabeçote está na posição 1." E "No tempo 0, a célula 1 contém o primeiro símbolo de w," etc.
3. ϕ_{move} **(Transições Válidas)**: Esta é a parte mais complexa. Ela garante que cada linha da tabela (configuração) segue legalmente da linha anterior, de acordo com as regras de transição da NTM.
   - Para cada "janela" de 2×3 células na tabela, a configuração na linha de baixo deve ser uma das possibilidades permitidas pela função de transição da NTM, dado o estado e os símbolos na linha de cima. Isso é codificado como uma grande disjunção (OU) de todas as transições válidas.
4. ϕ_{accept} **(Condição de Aceitação)**: Garante que, em algum momento, a máquina entra no estado de aceitação.
   - "(No tempo 0, o estado é q_{accept}) OU (No tempo 1, o estado é q_{accept}) OU … OU (No tempo $p(n)$, o estado é q_{accept})."

A Conexão Final:

- Se a NTM N **aceita** w, existe uma sequência de configurações válidas (um caminho na árvore de computação) que leva à aceitação. Essa sequência corresponde a uma atribuição de Verdadeiro/Falso para as variáveis que **satisfaz** todas as quatro partes da fórmula.
- Se a NTM N **não aceita** w, não existe tal sequência. Qualquer tentativa de preencher a tabela violará pelo menos uma das regras (ϕ_{start}, ϕ_{move} ou ϕ_{accept}), tornando a fórmula **insatisfazível**.

A construção da fórmula em si é um procedimento mecânico que leva tempo polinomial no tamanho da tabela, que é $O((n^k)^2) = O(n^{2k})$. Isso completa a prova.

Exercícios de Verificação

Atividade Prática

1. **Conceitual**: Por que a prova do Teorema de Cook-Levin precisa usar uma Máquina de Turing **Não-Determinística** como base para a simulação, em vez de uma determinística?
2. **A Importância das Sub-fórmulas**: O que aconteceria se a fórmula ϕ_w não incluísse a parte ϕ_{move}? Que tipo de "computação inválida" uma atribuição satisfatória poderia representar?
3. **Python e SAT Solvers**: Na prática, problemas SAT são resolvidos por programas chamados "SAT Solvers". Embora o problema seja NP-Completo no pior caso, esses solvers usam heurísticas incrivelmente eficientes. Pesquise brevemente sobre um SAT Solver moderno (ex: MiniSat, Z3, Glucose) e descreva em uma ou duas frases qual a principal técnica que eles usam para evitar a busca por força bruta de 2^n.

Referências Bibliográficas

SIPSER, Michael. **Introdução à Teoria da Computação**. 3. ed. São Paulo, Brasil: Cengage Learning, 2012.