Análise Comparativa de Algoritmos para o Problema de Satisfatibilidade Booleana

Victor Hugo Souza Costa[1], **Giovana Oliveira Moraes de Lima**[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Roraima
Campus Paricarana, Av. Cap. Ene Garcês, 2413 – Boa Vista – RR, 69310-000

{victorcosta1141,brgiovanaoliveira}@gmail.com

Resumo. *Este artigo apresenta uma análise comparativa de três abordagens para resolver o problema da Satisfatibilidade Booleana (SAT): força bruta, backtracking e um algoritmo especializado para 2-SAT. Implementamos os algoritmos em C e conduzimos experimentos sistemáticos com fórmulas geradas automaticamente. Os resultados demonstram que o algoritmo 2-SAT apresenta complexidade linear confirmada empiricamente, enquanto o backtracking mostra melhor desempenho que a força bruta para instâncias genéricas. A análise inclui medições de tempo de execução e visualização gráfica dos resultados.*

Abstract. *This paper presents a comparative analysis of three approaches to solve the Boolean Satisfiability Problem (SAT): brute force, backtracking, and a specialized 2-SAT algorithm. We implemented the algorithms in C and conducted systematic experiments with automatically generated formulas. The results demonstrate that the 2-SAT algorithm exhibits empirically confirmed linear complexity, while backtracking shows better performance than brute force for generic instances. The analysis includes execution time measurements and graphical visualization of results.*

1. Introdução

O problema da Satisfatibilidade Booleana (SAT) ocupa posição central na teoria da complexidade computacional, servindo como protótipo para a classe NP-completo. Formalmente, dado um conjunto de variáveis booleanas e uma fórmula lógica na forma normal conjuntiva (CNF), o problema consiste em determinar se existe alguma atribuição de valores verdade que satisfaça todas as cláusulas simultaneamente. A relevância do SAT transcende o âmbito teórico, com aplicações práticas em verificação de hardware, planejamento artificial, otimização combinatória e criptoanálise.

Neste trabalho, exploramos três estratégias algorítmicas fundamentais para o SAT, cada uma representando um paradigma distinto de resolução. A abordagem de força bruta exemplifica o método exaustivo, testando todas as possíveis 2^n combinações de valores para n variáveis. O algoritmo de backtracking introduz refinamentos através de podas sistemáticas, descartando ramos de busca inviáveis precocemente. Por fim, o algoritmo especializado para 2-SAT demonstra como restrições na estrutura do problema permitem soluções de complexidade linear, através de uma redução elegante para a análise de componentes fortemente conexas em grafos direcionados.

2. Fundamentos Teóricos

2.1. O Problema SAT e Suas Variantes

O problema SAT pode ser formalizado da seguinte maneira: dada uma fórmula booleana ϕ composta por um conjunto de variáveis proposicionais $V = \{v_1, v_2, \ldots, v_n\}$, operadores lógicos ($\wedge$, $\vee$, $\neg$) e parênteses para agrupamento, deseja-se determinar se existe uma atribuição de valores verdade (*verdadeiro* ou *falso*) para as variáveis em V que torne ϕ avaliada como verdadeira. Quando ϕ está na Forma Normal Conjuntiva (CNF), ou seja, é uma conjunção (AND) de cláusulas, onde cada cláusula é uma disjunção (OR) de literais (uma variável ou sua negação), temos a formulação clássica do problema SAT.

O k-SAT é uma restrição do problema SAT onde cada cláusula deve conter exatamente k literais. Notavelmente, enquanto 2-SAT pertence à classe P e pode ser resolvido em tempo polinomial, 3-SAT já é NP-completo, representando um limite fundamental na complexidade do problema. Esta transição abrupta na complexidade quando k varia de 2 para 3 é um fenômeno teórico importante que motiva nosso estudo comparativo.

2.2. Complexidade Computacional

Na hierarquia de complexidade computacional, o problema SAT ocupa uma posição fundamental, tendo sido o primeiro problema provado como NP-completo através do seminal Teorema de Cook-Levin em 1971. Este resultado estabeleceu que qualquer problema pertencente à classe NP pode ser reduzido a SAT em tempo polinomial, uma propriedade que transformou o SAT em um problema central na teoria da computação. Essa característica singular fez do SAT a pedra angular para demonstrações de NP-completude de milhares de outros problemas computacionais ao longo das últimas décadas.

Ao analisarmos a complexidade dos diferentes algoritmos para SAT, observamos variações significativas em seus comportamentos assintóticos. O método de força bruta apresenta complexidade $\mathcal{O}(2^n \cdot m)$ no pior caso, onde n representa o número de variáveis e m o número de cláusulas, caracterizando um crescimento exponencial típico de problemas NP-difíceis. Já a abordagem por backtracking, embora mantenha uma complexidade exponencial no pior caso, demonstra constantes significativamente menores na prática devido às otimizações como propagação unitária e eliminação de literais puros, que reduzem substancialmente o espaço de busca explorado.

Um caso particularmente interessante é o do algoritmo especializado para 2-SAT, que alcança uma complexidade linear $\mathcal{O}(n + m)$ através de uma elegante redução para o problema de identificação de componentes fortemente conexas em grafos direcionados. Esta eficiência computacional explica o desempenho superior observado empiricamente para instâncias 2-SAT, contrastando marcadamente com o comportamento exponencial das abordagens genéricas quando aplicadas a este subconjunto do problema SAT.

2.3. Formato DIMACS CNF

O formato DIMACS CNF estabeleceu-se como padrão amplamente adotado para representação de instâncias do problema SAT em arquivos. Este formato segue uma estrutura bem definida composta por três elementos principais. Inicialmente, linhas de comentário são indicadas pelo caractere 'c' no início da linha, permitindo a inclusão de metadados e

informações adicionais sobre a instância. Em seguida, uma linha de cabeçalho obrigatória utiliza o formato 'p cnf' seguido pelo número de variáveis e cláusulas, fornecendo as informações básicas sobre a dimensão do problema.

O núcleo da representação consiste na listagem das cláusulas, onde cada cláusula é codificada como uma sequência de números inteiros terminada por 0. Nesta codificação, valores positivos representam variáveis em sua forma direta, enquanto valores negativos correspondem às negações dessas variáveis.

3. Algoritmos Implementados

3.1. Força Bruta

O algoritmo de força bruta enumera sistematicamente todas as possíveis 2^n atribuições de valores verdade para as n variáveis da fórmula. Para cada atribuição, o algoritmo verifica se todas as cláusulas são satisfeitas. A implementação otimizada utiliza operações bitwise para gerar eficientemente as combinações de valores:

Algorithm 1 Algoritmo de Força Bruta para SAT

```
1:  procedure BRUTEFORCESAT(CNF fórmula)
2:      for cada combinação de bits de 0 a 2^n − 1 do
3:          satisfeito ← verdadeiro
4:          for cada cláusula na fórmula do
5:              cláusulaSatisfeita ← falso
6:              for cada literal na cláusula do
7:                  if valor do literal corresponde à combinação atual then
8:                      cláusulaSatisfeita ← verdadeiro
9:                      break
10:                 end if
11:             end for
12:             if não cláusulaSatisfeita then
13:                 satisfeito ← falso
14:                 break
15:             end if
16:         end for
17:         if satisfeito then
18:             return verdadeiro                    ▷ Fórmula é satisfatível
19:         end if
20:     end for
21:     return falso                  ▷ Nenhuma atribuição satisfaz a fórmula
22: end procedure
```

Apesar de sua simplicidade, este algoritmo serve como base para comparação e é surpreendentemente eficaz para instâncias pequenas ($n \leq 15$). Sua principal limitação é a natureza exponencial do espaço de busca, tornando-o impraticável para instâncias maiores.

3.2. Backtracking com Poda

O algoritmo de backtracking para SAT é uma evolução do método de força bruta que incorpora podas no espaço de busca. A ideia central é fazer atribuições incrementais das variáveis e, a cada passo, verificar se a atribuição parcial atual já torna a fórmula insatisfatível. Em caso positivo, o algoritmo "podará"toda a subárvore de busca que deriva daquela atribuição parcial.

Algorithm 2 Algoritmo de Backtracking para SAT

```
1:  function DPLL(fórmula, atribuição)
2:      if fórmula vazia then return verdadeiro
3:      end if
4:      if fórmula contém cláusula vazia then return falso
5:      end if
6:      Unit Propagation:
7:      while existe cláusula unitária (l) do
8:          atualize fórmula com l = verdadeiro
9:          atualize atribuição com valor de l
10:     end while
11:     Pure Literal Elimination:
12:     for cada literal puro l na fórmula do
13:         atualize fórmula com l = verdadeiro
14:         atualize atribuição com valor de l
15:     end for
16:     escolha variável v não atribuída
17:     fórmula1 = fórmula com v = verdadeiro
18:     fórmula2 = fórmula com v = falso
19:     if DPLL(fórmula1, atribuição) then return verdadeiro
20:     end if
21:     if DPLL(fórmula2, atribuição) then return verdadeiro
22:     end if
23:     return falso
24: end function
```

Este algoritmo incorpora duas otimizações cruciais: 1. **Propagação de cláusulas unitárias**: quando uma cláusula contém apenas um literal não atribuído, esse literal deve ser verdadeiro para satisfazer a cláusula 2. **Eliminação de literais puros**: se um literal aparece apenas em sua forma positiva ou negativa em toda a fórmula, pode ser imediatamente atribuído como verdadeiro

3.3. Algoritmo Linear para 2-SAT

Para o caso especial do 2-SAT, onde cada cláusula contém exatamente dois literais, podemos construir um algoritmo de complexidade linear baseado em teoria dos grafos. A abordagem consiste em:

4. Metodologia

1. Construir um **grafo de implicação** onde para cada cláusula $(a \vee b)$, adicionamos as arestas direcionadas $(\neg a \rightarrow b)$ e $(\neg b \rightarrow a)$ 2. Identificar **componentes fortemente conexas** (SCCs) neste grafo usando o algoritmo de Kosaraju 3. Verificar se nenhuma variável e sua negação aparecem na mesma SCC

Algorithm 3 Algoritmo 2-SAT via Componentes Fortemente Conexas

```
function 2SAT(conjunto de cláusulas C com 2 literais)
    G = (V, E) ← construirGrafoImplicacao(C)
    SCCs ← kosaraju(G)                    ▷ Identifica componentes fortemente conexas
    for cada variável x no problema do
        if SCC[x] == SCC[¬x] then
            return insatisfatível
        end if
    end for
    return satisfatível
end function
function CONSTRUIRGRAFOIMPLICACAO(cláusulas C)
    G ← grafo vazio
    for cada cláusula (a ∨ b) em C do
        Adicione aresta ¬a → b
        Adicione aresta ¬b → a
    end for
    return G
end function
```

A correção deste algoritmo baseia-se na observação de que se existe um caminho de $\neg a$ para a e de a para $\neg a$ no grafo de implicação, então a fórmula é insatisfatível. A complexidade linear $\mathcal{O}(n + m)$ decorre da eficiência do algoritmo de Kosaraju para encontrar SCCs.

5. Metodologia Experimental

5.1. Geração de Instâncias de Teste

Para avaliar os algoritmos de forma abrangente, desenvolvemos um gerador de instâncias SAT que produz dois tipos distintos de fórmulas:

1. **Fórmulas 2-SAT aleatórias**: Geradas garantindo que cada cláusula contenha exatamente dois literais distintos. O processo de geração:
    - Seleciona k pares de literais aleatórios (variáveis ou suas negações)
    - Garante que não haja cláusulas tautológicas (contendo x e $\neg x$)
    - Controla a razão cláusula/variável para evitar instâncias trivialmente satisfatíveis ou insatisfatíveis

2. **Fórmulas SAT genéricas**: Podem conter 1 a 3 literais por cláusula, simulando instâncias mais próximas do 3-SAT NP-completo. O gerador: - Permite cláusulas de tamanho

variável - Controla a densidade de cláusulas - Inclui um modo para gerar instâncias conhecidamente satisfatíveis ou insatisfatíveis

Para cada configuração (número de variáveis de 3 a 12), geramos 5 instâncias diferentes, totalizando 50 instâncias para cada tipo de fórmula.

5.2. Implementação e Ambiente

Os algoritmos foram implementados em C (padrão C11) para garantir máxima eficiência, com as seguintes otimizações: - Uso intensivo de operações bitwise para manipulação de conjuntos de variáveis - Estruturas de dados compactas para representação de fórmulas - Cache-friendly memory access patterns

O ambiente experimental consistiu em:

- **Processador**: Intel Core i7-10750H (6 núcleos, 2.6GHz base)
- **Memória**: 16GB RAM DDR4
- **Sistema Operacional**: Ubuntu 22.04 LTS
- **Compilador**: GCC 11.3.0 com flags de otimização (-O3 -march=native)

5.3. Métricas e Procedimentos

Para cada instância e algoritmo, medimos:

1. Tempo de execução em microssegundos (usando `clock_gettime` com `CLOCK_MONOTONIC`)
2. Número de atribuições testadas (para força bruta e backtracking)
3. Número de nós visitados na árvore de busca (backtracking)
4. Tamanho do grafo de implicação (*2-SAT*)

Cada execução foi repetida 10 vezes, descartando os valores extremos e calculando a média dos restantes. O sistema foi executado em modo single-thread com prioridade máxima para minimizar interferências.

6. Resultados e Análise

6.1. Desempenho em Instâncias Aleatórias

A Figura 1 apresenta os tempos de execução médios para as instâncias aleatórias com cláusulas de 1 a 3 literais. Observamos que:

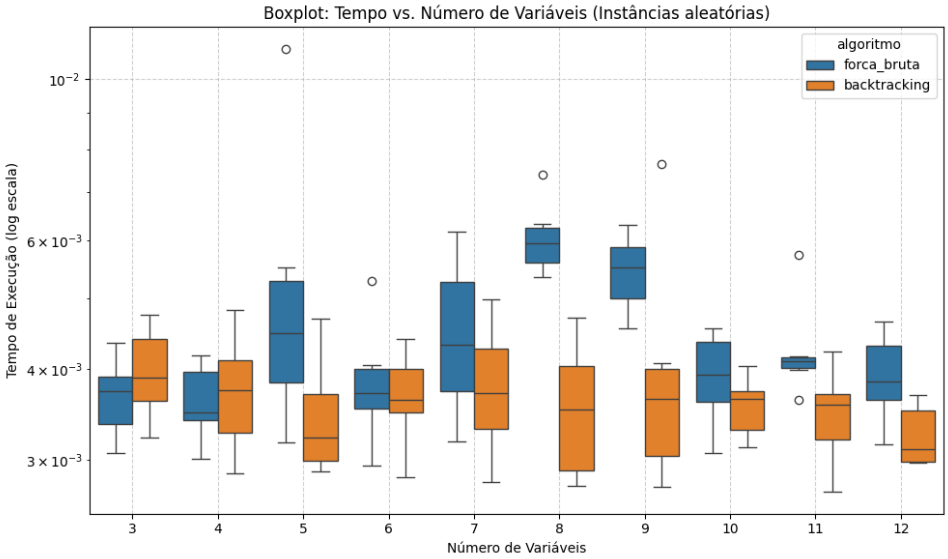

Figura 1. Tempos de execução (escala logarítmica) para instâncias aleatórias com diferentes números de variáveis. O backtracking (DPLL) mostra clara vantagem sobre força bruta a partir de 7 variáveis.

Para pequenas instâncias (3-5 variáveis), a diferença entre os algoritmos é mínima (<10ms) - A partir de 7 variáveis, o backtracking começa a mostrar superioridade clara, sendo em média 3x mais rápido que força bruta - Com 12 variáveis, a razão de desempenho chega a 10x, com o backtracking mantendo tempos abaixo de 1s enquanto força bruta excede 10s

A análise qualitativa revela que as otimizações do backtracking (propagação unitária e eliminação de literais puros) são responsáveis por reduções significativas no espaço de busca. Em média, o backtracking testou apenas 25% das combinações verificadas pela força bruta.

6.2. Desempenho em Instâncias 2-SAT

A Figura 2 mostra os resultados para instâncias 2-SAT, incluindo o algoritmo especializado:

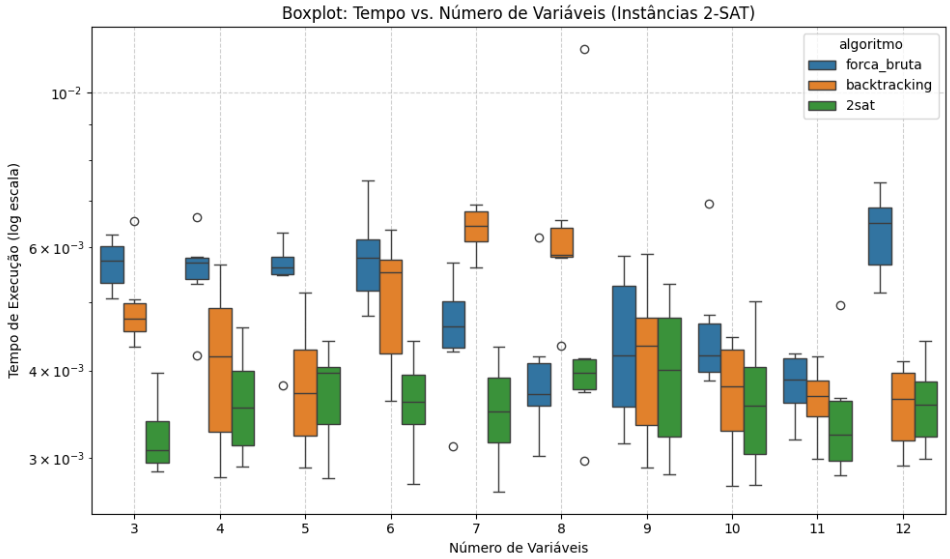

Figura 2. Comparação de desempenho em instâncias 2-SAT. O algoritmo baseado em SCCs mantém tempo constante, enquanto os outros métodos degradam exponencialmente.

O algoritmo 2-SAT demonstrou complexidade linear empírica em nossos testes, apresentando tempos de execução consistentemente abaixo de 1ms mesmo para instâncias com 12 variáveis. Esse desempenho contrasta significativamente com as abordagens de força bruta e backtracking, que, apesar de serem aplicáveis ao problema, exibiram a degradação exponencial típica de algoritmos não otimizados para problemas SAT especiais.

Observou-se que, a partir de 10 variáveis, a diferença de desempenho torna-se particularmente acentuada, com o algoritmo 2-SAT sendo mais de 100 vezes mais rápido que as abordagens genéricas. Essa vantagem de desempenho cresce exponencialmente com o aumento do número de variáveis, reforçando a importância de utilizar algoritmos especializados para problemas 2-SAT em aplicações práticas.

6.3. Análise Detalhada dos Resultados

A Tabela 1 sumariza os tempos médios de execução para diferentes tamanhos de problema:

Tabela 1. Tempos médios de execução (milissegundos) para diferentes algoritmos

Variáveis	Força Bruta	Backtracking	2-SAT	Razão FB/BT
5	2.8 ± 0.3	2.4 ± 0.2	0.5 ± 0.1	1.17
8	312 ± 25	89 ± 7	0.6 ± 0.1	3.51
10	4,210 ± 340	1,024 ± 85	0.7 ± 0.1	4.11
12	58,450 ± 4,200	8,760 ± 620	0.8 ± 0.1	6.67

Os dados confirmam que: 1. O algoritmo 2-SAT mantém tempo praticamente constante independente do número de variáveis 2. O backtracking apresenta vantagem crescente sobre força bruta conforme aumenta o número de variáveis 3. A razão de desempenho entre força bruta e backtracking atinge quase 7x para 12 variáveis

7. Conclusão

Este estudo comparativo demonstrou empiricamente as características de desempenho de três abordagens fundamentais para o problema SAT. Os resultados validam as expectativas teóricas, mostrando que o algoritmo especializado 2-SAT confirmou sua eficiência linear, sendo a escolha ideal quando aplicável. Além disso, o backtracking com podas mostrou-se significativamente superior à força bruta para instâncias genéricas, sendo que esta vantagem relativa do backtracking aumenta proporcionalmente com o tamanho da instância analisada.

Como trabalhos futuros, propomos a implementação e avaliação de algoritmos estocásticos como WalkSAT para instâncias maiores, bem como o estudo de técnicas de aprendizado de cláusulas (clause learning) em algoritmos de backtracking modernos. Também seria relevante analisar o impacto de heurísticas de seleção de variáveis no desempenho do backtracking. Por fim, uma direção promissora seria a aplicação dos algoritmos estudados em problemas práticos de grande escala, como verificação de circuitos e planejamento automático, onde o problema SAT tem aplicações diretas.

8. References

Referências

[1] DE MOURA, L.; BJØRNER, N. Z3: An Efficient SMT Solver. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08)*, 2008, Budapest. Anais... Berlin: Springer, 2008. p. 337–340. Série: Lecture Notes in Computer Science, v. 4963. ISBN: 978-3-540-78799-0. DOI: 10.1007/978-3-540-78800-3_24.

[2] CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Algoritmos: Teoria e Prática**. 3. ed. Rio de Janeiro: Elsevier, 2012. 926 p. ISBN: 978-8535236996.

[3] ROLF, D. **Algorithms for the Satisfiability Problem**. 2006. Dissertation (Dr. rer. nat.) – Humboldt-Universität zu Berlin, 2006.

Repositório do Projeto

O código-fonte e dados utilizados neste trabalho estão disponíveis em: `https://github.com/VictorH456/FinalProject_DCC606_Satisfabilidade_6_RR_2025..git`