Universidade Federal do Ceará
Campus Russas
Ciência da Computação

Trabalho Final
Lógica Para Computação

Matheus Nunes Lima
Luiz Felipe Rodrigues Nogueira

Russas
2025

Geração de cláusulas:
Algoritmo:

- Seleção de Variáveis: Para cada cláusula, selecionam-se k variáveis únicas (sem repetição) de um conjunto de n variáveis.
- Negações aleatórias: Cada variável é negada ou mantida positivamente com probabilidade de 50%.
- Unicidade de Cláusulas: Garante-se que não haja cláusulas duplicadas usando frozenset para armazenamento e verificação.

```python
def gerar_clausulas_validas(n_var, m_claus, k_lits):
    variaveis = list(range(1, n_var + 1))
    clausulas = set()
    while len(clausulas) < m_claus:
        vars_clausula = random.sample(variaveis, k_lits)
        clausula = [v * random.choice([1, -1]) for v in vars_clausula]
        if frozenset(clausula) not in clausulas:
            clausulas.add(frozenset(clausula))
    return [list(c) for c in clausulas]
```

Parâmetros:

- n: Número de variáveis (ex: 50, 100, 200).
- k: Tamanho das cláusulas (3 para 3-SAT, 5 para 5-SAT).
- α = m/n: Razão cláusulas/variáveis, variada em intervalos específicos (ex: 1.0 a 10.0 para 3-SAT).

```python
n = 200
k = 3
alpha_inicio = 1.0
alpha_fim = 10.0
alpha_passo = 0.1
num_instancias = 50
```

Resolução das Instâncias:

- Utilizou-se o Glucose3 através da biblioteca PySAT, um solver SAT eficiente e amplamente utilizado.
- Processo:
    - Para cada valor de α, geram-se m = α × n cláusulas.
    - Mede-se o tempo de execução e a taxa de sucesso (SAT/UNSAT).

```python
def executar_testes_alpha(n_var, m_claus, k_lits, num_testes):
    resultados = {'satisfazivel': 0, 'tempo_total': 0.0, 'tempos': []}
    for _ in range(num_testes):
        clausulas = gerar_clausulas_validas(n_var, m_claus, k_lits)
        inicio = time.perf_counter()
        with Glucose3(bootstrap_with=clausulas) as solver:
            resultado = solver.solve()
        tempo = time.perf_counter() - inicio
        resultados['tempo_total'] += tempo
        resultados['tempos'].append(tempo)
        if resultado:
            resultados['satisfazivel'] += 1
    return resultados
```

Geração de Logs:

Tomei a Liberdade de criar uma função para criar e armazenar os dados de execução que considerei mais importantes em formato de texto, sendo eles:

Alpha crítico calculado: 22.1

==========

Alpha 1.0 (m=20):

Probabilidade SAT: 100.00%

Tempo total: 0.00s

Tempo médio/instância: 0.0000s

Sendo esses dados acima apenas exemplos de uma execução do 5-SAT com alpha indo de 1-30 e com n = 20

Enviarei os documentos .txt dos logs de cada execução juntamente com o trabalho, eles servem para verificar o andamento do cálculo dos gráficos, tanto de tempo quanto do alpha crítico, e são realizados pela seguinte função:

```python
def registrar_log(nome_arquivo, dados, alpha_c):
    with open(nome_arquivo, 'w') as log:
        log.write("RELATÓRIO COMPLETO\n")
        log.write(f"Alpha crítico calculado: {alpha_c if alpha_c else 'N/A'}\n")
        log.write("="*50 + "\n")
        for alpha, info in dados.items():
            log.write(f"Alpha {alpha:.1f} (m={info['m_clausulas']}):\n")
            log.write(f"  Probabilidade SAT: {info['probabilidade']:.2%}\n")
            log.write(f"  Tempo total: {info['tempo_total']:.2f}s\n")
            log.write(f"  Tempo médio/instância: {info['tempo_medio_instancia']:.4f}s\n")
            log.write("-"*50 + "\n")
```

Cálculo do Alpha Crítico:

Essa função ainda não está 100% pronta, pois considera o ponto de alpha crítico apenas quando a probabilidade de SAT do alpha em questão for = exatamente 50%, porém, foi a melhor alternativa que encontrei, e ela costuma acertar bons pontos próximos aos da teoria, oque vai ser verificado no próximo ponto, da geração de gráficos, então, como estava acertando, decidi por manter, mas futuramente pretendo aprimorar essa função

```python
def calcular_alpha_critico(alphas, probabilidades):
    try:
        f = interp1d(probabilidades, alphas)
        return float(f(0.5))
    except:
        return None
```

Geração de gráficos:

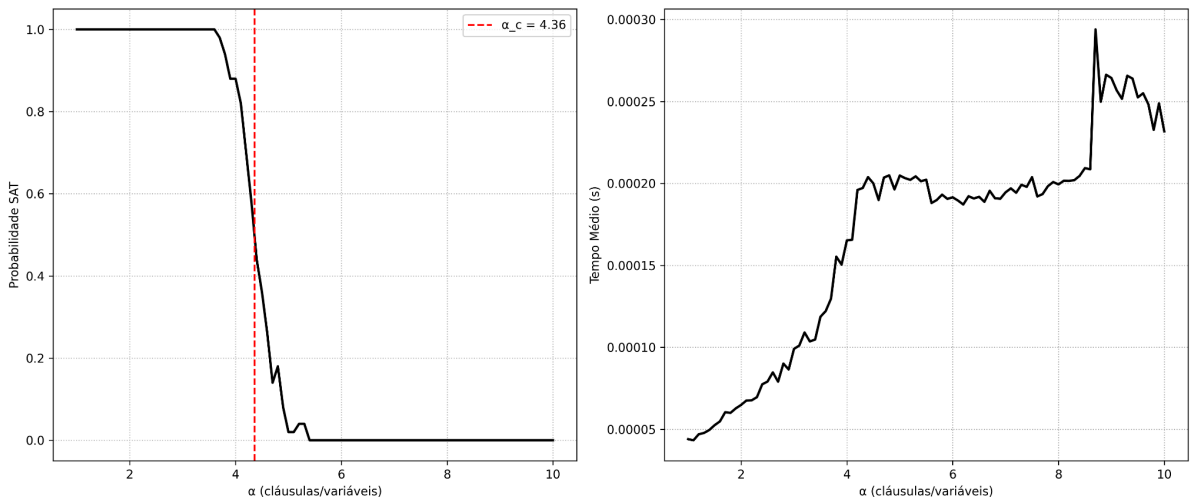
utilizei a biblioteca matplotlib para gerar os gráficos, tanto o de satisfazibilidade quanto o te tempo de execução, utilizando os dados armazenados anteriormente na função de executar testes, que armazena os dados em ‘resultados’ durante cada execução de alpha e que são transferidos para ‘dados_experimento’

que registra o dado de cada alpha e deixa guardado para o uso posterior nas funções geradoras de resultado:

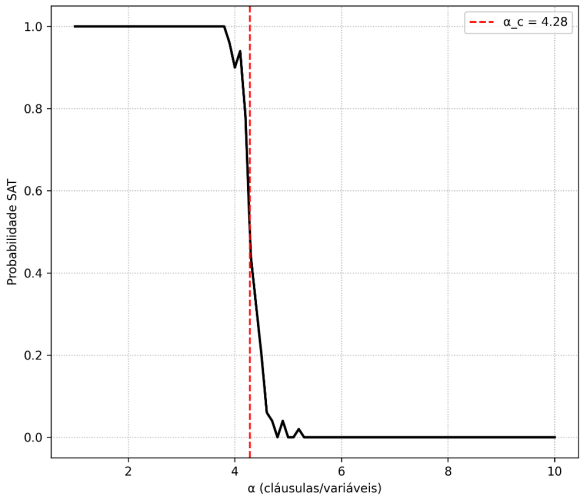
```python
def gerar_graficos(dados, alpha_c):
    plt.figure(figsize=(14,6))

    plt.subplot(1,2,1)
    alphas = list(dados.keys())
    probabilidades = [d['probabilidade'] for d in dados.values()]
    plt.plot(alphas, probabilidades, 'k-', linewidth=2)
    if alpha_c:
        plt.axvline(alpha_c, color='r', linestyle='--', label=f'α_c = {alpha_c:.2f}')
    plt.xlabel('α (cláusulas/variáveis)')
    plt.ylabel('Probabilidade SAT')
    plt.grid(linestyle=':')
    plt.legend()

    plt.subplot(1,2,2)
    tempos = [d['tempo_medio_instancia'] for d in dados.values()]
    plt.plot(alphas, tempos, 'k-', linewidth=2)
    plt.xlabel('α (cláusulas/variáveis)')
    plt.ylabel('Tempo Médio (s)')
    plt.grid(linestyle=':')

    plt.tight_layout()
    plt.savefig(nome_grafico, dpi=300)
    plt.close()
```

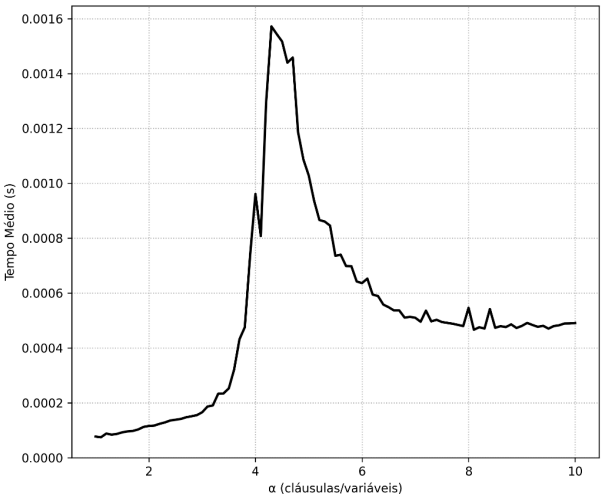
Gráficos gerados:

N= 50 3-SAT Alpha 1-10 variando +0.1 a cada alpha calculado

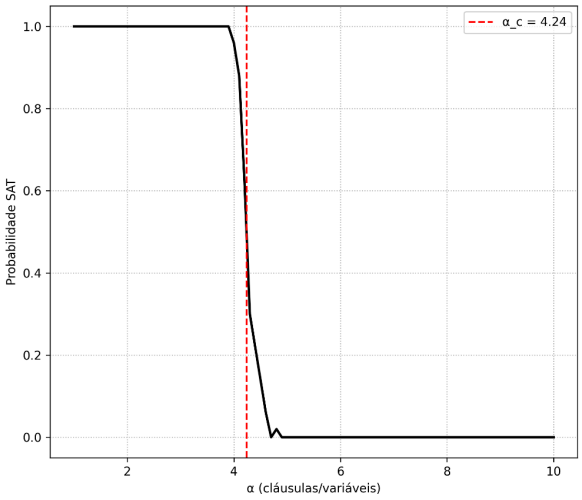


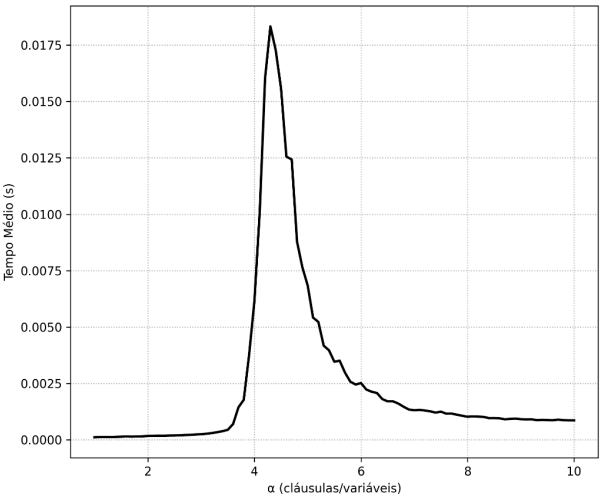
N= 100 3-SAT Alpha 1-10 variando +0.1 a cada alpha calculado



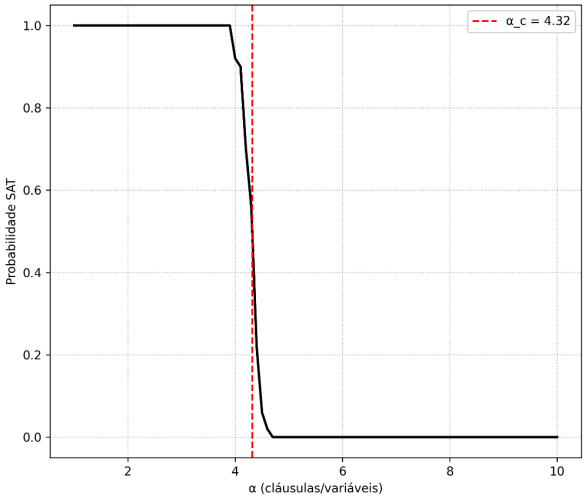


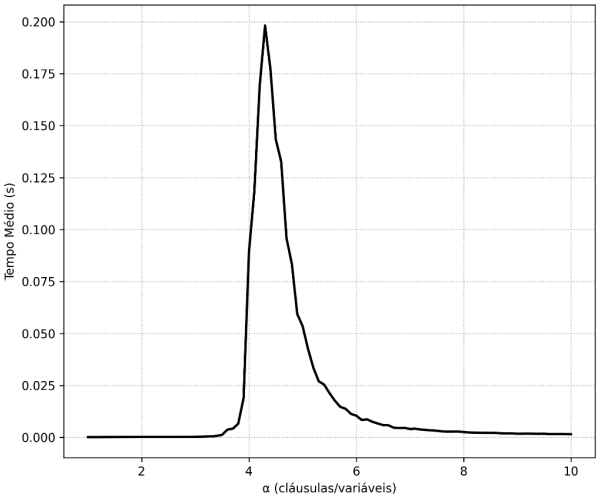
N= 150 3-SAT Alpha 1-10 variando +0.1 a cada alpha calculado



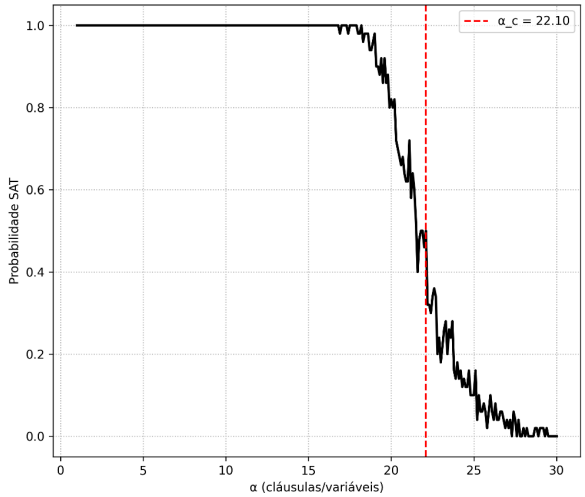


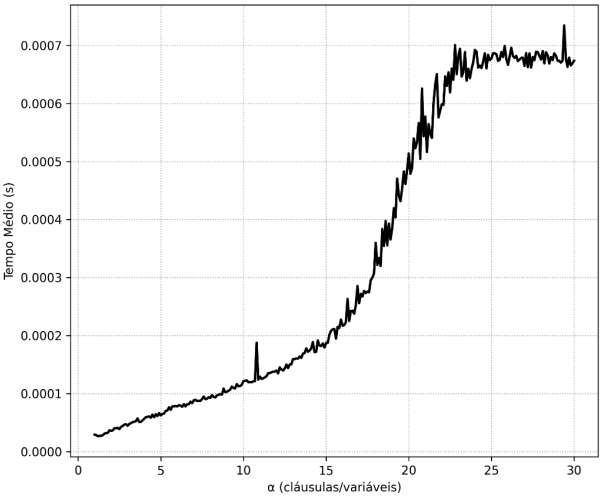
N= 200 3-SAT Alpha 1-10 variando +0.1 a cada alpha calculado



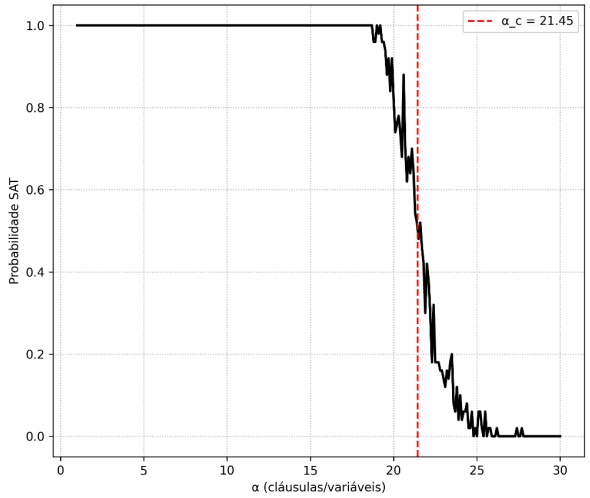


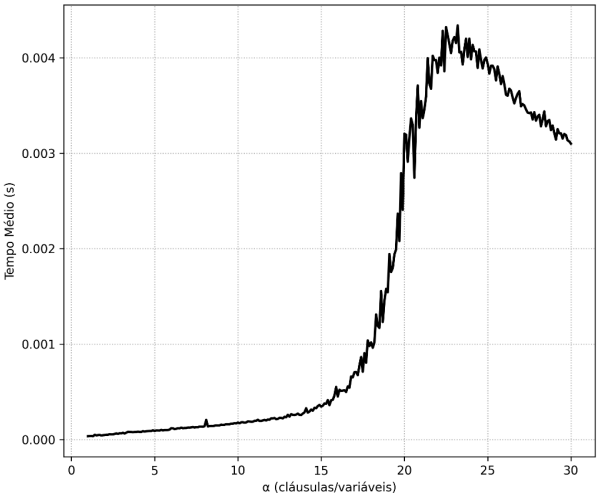
N= 20 5-SAT Alpha 1-30 variando +0.1 a cada alpha calculado



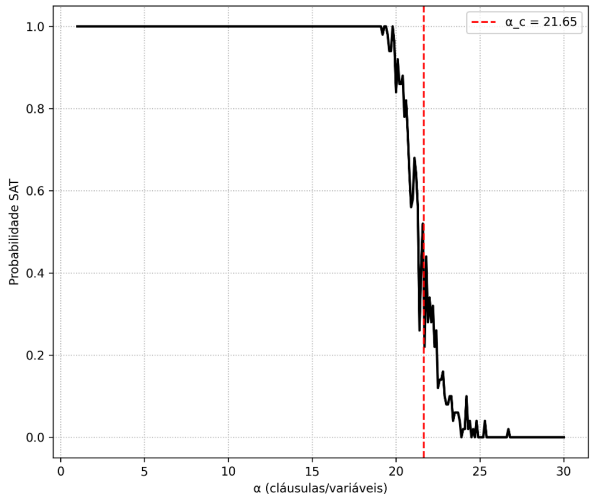


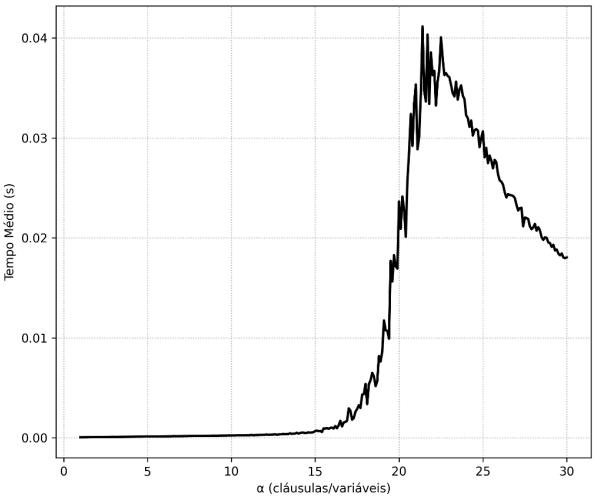
N= 30 5-SAT Alpha 1-30 variando +0.1 a cada alpha calculado



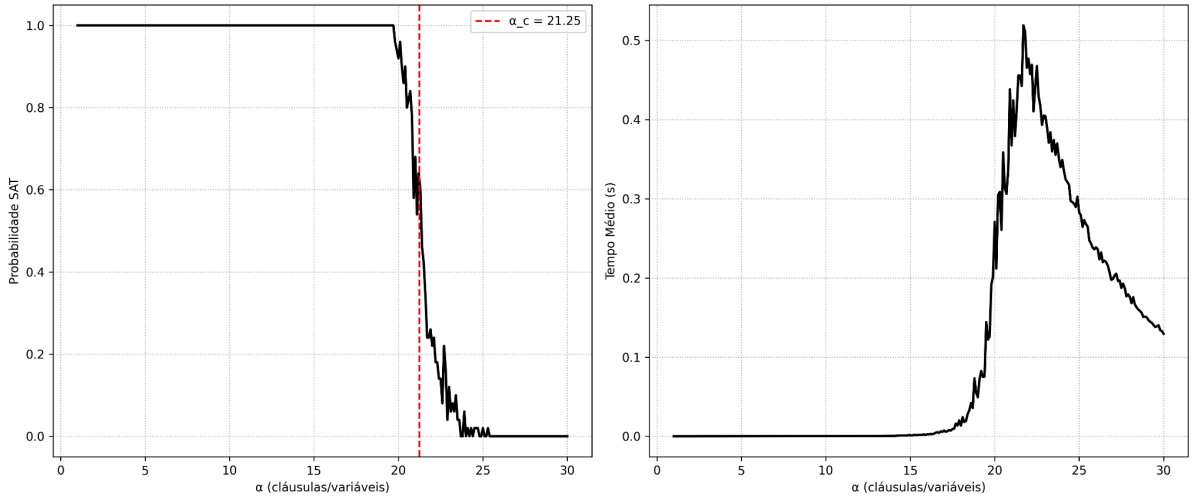


N= 40 5-SAT Alpha 1-30 variando +0.1 a cada alpha calculado





N= 50 5-SAT Alpha 1-30 variando +0.1 a cada alpha calculado



Considerações sobre os resultados:
Estimativa do Ponto Crítico (α_c):

3-SAT:

Teórico: α_c ≈ 4.26.

Experimental: Observou-se queda abrupta da probabilidade de SAT para α entre 4.2 e 4.5, compatível com a literatura.

Exemplo: Para n = 100, a probabilidade cai de ~100% (α=3.8) para ~20% (α=4.5).

5-SAT:

Teórico: α_c ≈ 21.12.

Experimental: A transição ocorre de forma mais gradual, com queda significativa em α ≈ 20.0–22.0.

Exemplo: Para n = 50 a probabilidade cai de ~100% (α=19.8) para ~22.2% (α=22.2).

Esses dados podem ser observados nos logs que enviarei junto ao relatório

Tempo de execução:

Como pode ser observado nos gráficos acima, o tempo de execução cresce quase que linearmente até chegar ao pico de complexidade no alpha crítico ou muito próximo dele, depois desse ponto costuma desce e estabilizar-se em uma complexidade mais baixa, porém o comportamento muda um pouco de acordo com os dados utilizados no experimento e a cada execução, visto que as instâncias são geradas aleatoriamente novamente toda vez que o programa é rodado.

Reflexão sobre Diferenças entre 3-SAT e 5-SAT:

- Transição de Fase:

3-SAT: Transição abrupta, com queda quase vertical na probabilidade de SAT.

5-SAT: Transição um pouco mais suave.

- Complexidade Computacional:

3-SAT: Rápida execução, tornando mais fácil fazer experimentos com um conjunto maior de variáveis.

5-SAT: Tempos de execução significativamente maiores devido ao maior número de literais por cláusula.

- Ponto Crítico:

3-SAT: um pico de complexidade bastante definido para o ponto crítico, assim como sua fase de transição.

5-SAT: uma subida mais “tranquila” quando se trata da complexidade, também chegando ao seu pico de tempo de execução praticamente rente a ele, porém, tem números próximos disso em seus arredores, oque um pouco mais difícil de identificar ele

Conclusão:

A declaração anterior sobre cada SAT faz com que deixe explicita a relação direta entre ponto de transição e tempo de execução, oque pode ser novamente comprovado nos arquivos de logs que eu enviarei, caso assim deseje!

Referências:

BONA,Glauber de. Satisfazibilidade Probabilística. 2011. 101 f. Tese (Mestrado em Ciência da Computação) – Universidade de São Paulo. Disponível em: <https://www.teses.usp.br/teses/disponiveis/45/45134/tde-02062011-181639/publico/tese.pdf>. Acesso em: 15 fev. 2025