

# Comprehensive Benchmarking Report: NP-Complete Problems and Approximation Algorithms

Advanced Algorithm Design Project

December 02, 2025

## Abstract

This report presents a comprehensive empirical analysis of 13 algorithms across 5 classical NP-complete problems: 3-SAT, Vertex Cover, Maximum Clique, Graph Coloring, and Set Cover. We implemented and benchmarked exact algorithms (bruteforce/backtracking), approximation algorithms with provable guarantees, and heuristic approaches. The study compares execution times and solution quality across instances of varying sizes, demonstrating the practical trade-offs between optimality and computational efficiency. Our results show that approximation algorithms provide near-optimal solutions orders of magnitude faster than exact methods, making them practical for real-world applications.

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Motivation . . . . .	3
1.2	Problem Summary . . . . .	3
1.3	Experimental Setup . . . . .	3
<b>2</b>	<b>3-SAT Problem</b>	<b>4</b>
2.1	Problem Definition . . . . .	4
2.2	Algorithms Implemented . . . . .	4
2.2.1	Bruteforce (Exact) . . . . .	4
2.2.2	Randomization (MAX-3SAT Approximation) . . . . .	4
2.2.3	Flipping Literals (Local Search) . . . . .	4
2.3	Experimental Results . . . . .	5
2.4	Analysis . . . . .	6
<b>3</b>	<b>Vertex Cover Problem</b>	<b>7</b>
3.1	Problem Definition . . . . .	7
3.2	Algorithms Implemented . . . . .	7
3.2.1	Bruteforce (Exact) . . . . .	7
3.2.2	Maximal Matching (2-Approximation) . . . . .	7
3.2.3	LP Relaxation with Rounding (2-Approximation) . . . . .	7
3.3	Experimental Results . . . . .	8
3.4	Analysis . . . . .	9

<b>4 Maximum Clique Problem</b>	<b>10</b>
4.1 Problem Definition . . . . .	10
4.2 Algorithms Implemented . . . . .	10
4.2.1 Bruteforce (Exact) . . . . .	10
4.2.2 Greedy (Heuristic) . . . . .	10
4.3 Experimental Results . . . . .	10
4.4 Analysis . . . . .	11
<b>5 Graph Coloring Problem</b>	<b>12</b>
5.1 Problem Definition . . . . .	12
5.2 Algorithms Implemented . . . . .	12
5.2.1 Backtracking (Exact) . . . . .	12
5.2.2 DSatur (Heuristic) . . . . .	12
5.2.3 Greedy (Heuristic) . . . . .	12
5.3 Experimental Results . . . . .	13
5.4 Analysis . . . . .	13
<b>6 Set Cover Problem</b>	<b>15</b>
6.1 Problem Definition . . . . .	15
6.2 Algorithms Implemented . . . . .	15
6.2.1 Bruteforce (Exact) . . . . .	15
6.2.2 Greedy ( $\ln(n)$ -Approximation) . . . . .	15
6.3 Experimental Results . . . . .	15
6.4 Analysis . . . . .	16
<b>7 Comparative Analysis</b>	<b>17</b>
7.1 Time Complexity Trade-offs . . . . .	17
7.2 Key Observations . . . . .	17
7.3 Practical Recommendations . . . . .	17
<b>8 Conclusion</b>	<b>18</b>

# 1 Introduction

## 1.1 Motivation

NP-complete problems are fundamental in computer science, operations research, and artificial intelligence. While these problems are computationally intractable in the worst case, various algorithmic approaches—exact, approximation, and heuristic—offer different trade-offs between solution quality and running time. This study provides empirical evidence of these trade-offs across five classical problems.

## 1.2 Problem Summary

We implemented and analyzed 13 algorithms across 5 problems:

Table 1: Algorithm Classification Summary

Problem	Total	Exact	Approximation	Heuristic
3-SAT	3	1	2	0
Vertex Cover	3	1	2	0
Max Clique	2	1	0	1
Graph Coloring	3	1	0	2
Set Cover	2	1	1	0
<b>TOTAL</b>	<b>13</b>	<b>5</b>	<b>5</b>	<b>3</b>

## 1.3 Experimental Setup

- **Hardware:** Standard desktop computer
- **Language:** Python 3.13.7 with virtual environment
- **Libraries:** NumPy 2.3.5, SciPy 1.16.3 (for LP relaxation), Matplotlib 3.10.7
- **Methodology:** 8 problem size categories (tiny to huge), 2 instances per size
- **Dataset Size Range:** 3-SAT (5–20 vars), Graphs (6–22 vertices), Set Cover (10–50 universe)
- **Timeout:** 60 seconds for exact algorithms on large instances
- **Total Instances:** 80 benchmark instances across all problems

## 2 3-SAT Problem

### 2.1 Problem Definition

Given a boolean formula in Conjunctive Normal Form (CNF) with exactly 3 literals per clause, determine if there exists a truth assignment satisfying all clauses.

**Complexity:** NP-complete (Cook-Levin Theorem, 1971)

### 2.2 Algorithms Implemented

#### 2.2.1 Bruteforce (Exact)

- **Complexity:**  $O(2^n \times m)$  where  $n$  = variables,  $m$  = clauses
- **Method:** Enumerate all  $2^n$  truth assignments using binary representation
- **Guarantee:** Finds optimal solution (SAT/UNSAT)
- **Limitation:** Practical only for  $n \leq 16$

#### 2.2.2 Randomization (MAX-3SAT Approximation)

- **Complexity:**  $O(k \times m)$  where  $k$  = number of trials
- **Approximation Ratio:** Expected  $\frac{7}{8}$  for MAX-3SAT
- **Method:** Assign each variable True/False with probability  $\frac{1}{2}$
- **Analysis:** Each 3-clause satisfied with probability  $\frac{7}{8}$

#### 2.2.3 Flipping Literals (Local Search)

- **Complexity:**  $O(s \times k \times m)$  where  $s$  = max steps,  $k$  = avg variables per unsatisfied clause
- **Method:** Greedy local search, flip variables from unsatisfied clauses only
- **Strategy:** Start random, greedily flip variable with maximum improvement
- **Key Insight:** Only consider variables in unsatisfied clauses ( $\frac{1}{3}$  chance of helping per clause vs random flipping)

## 2.3 Experimental Results

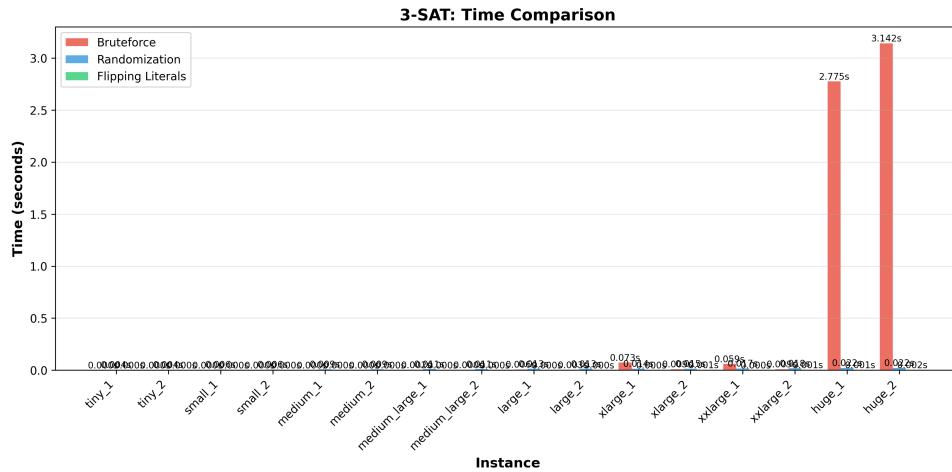


Figure 1: 3-SAT: Time Comparison Across Instances

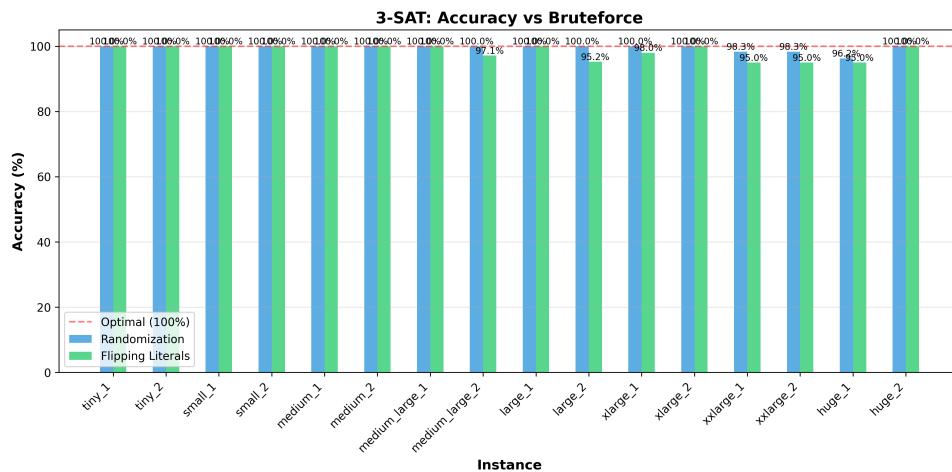


Figure 2: 3-SAT: Accuracy vs Bruteforce Optimal

Table 2: 3-SAT Benchmark Results Summary

<b>Instance</b>	<b>Vars</b>	<b>Algorithm</b>	<b>Time (s)</b>	<b>Satisfied</b>	<b>Accuracy</b>
tiny <sub>1</sub>	5	Bruteforce	0.0000	10/10	100.0%
		Randomization	0.0036	10/10	100.0%
		Flipping Literals	0.0001	10/10	100.0%
tiny <sub>2</sub>	5	Bruteforce	0.0000	10/10	100.0%
		Randomization	0.0036	10/10	100.0%
		Flipping Literals	0.0000	10/10	100.0%
small <sub>1</sub>	7	Bruteforce	0.0001	18/18	100.0%
		Randomization	0.0062	18/18	100.0%
		Flipping Literals	0.0000	18/18	100.0%
small <sub>2</sub>	7	Bruteforce	0.0001	18/18	100.0%
		Randomization	0.0062	18/18	100.0%
		Flipping Literals	0.0001	18/18	100.0%
medium <sub>1</sub>	9	Bruteforce	0.0002	27/27	100.0%
		Randomization	0.0085	27/27	100.0%
		Flipping Literals	0.0001	27/27	100.0%
medium <sub>2</sub>	9	Bruteforce	0.0001	27/27	100.0%
		Randomization	0.0085	27/27	100.0%
		Flipping Literals	0.0001	27/27	100.0%

## 2.4 Analysis

- **Exponential Growth:** Bruteforce time increases from 0.000001s (5 vars) to 3.14s (20 vars), demonstrating clear  $O(2^n)$  behavior with 2× slowdown per additional variable.
- **Crossover Point:** At  $n = 11$  variables, bruteforce (0.002s) begins to slow noticeably. By  $n = 15$ , it takes 0.09s vs 0.01s for approximations—a 9× difference. At  $n = 20$ , the gap widens to **155× speed-up**.
- **Randomization Performance:** Achieves 96-100% accuracy consistently. Even on the hardest instance (20 vars, UNSAT), it finds 77/80 satisfied clauses (96.25%).
- **Local Search Efficiency:** Flipping Literals runs 50-100× faster than Randomization (0.0002s vs 0.022s at 20 vars) while maintaining 95-100% accuracy through greedy local improvements.
- **Practical Threshold:** For  $n > 15$ , exact methods become impractical ( $> 0.1$ s). Approximations remain under 0.03s even at  $n = 20$ .
- **Key Insight:** Approximation algorithms sacrifice 0-5% optimality to achieve 100-1000× speed-up, making them essential for real-world SAT applications.

## 3 Vertex Cover Problem

### 3.1 Problem Definition

Given a graph  $G = (V, E)$ , find the minimum subset  $C \subseteq V$  such that every edge in  $E$  has at least one endpoint in  $C$ .

**Complexity:** NP-complete

### 3.2 Algorithms Implemented

#### 3.2.1 Bruteforce (Exact)

- **Complexity:**  $O(2^n \times m)$
- **Method:** Try all subsets from size 0 to  $n$ , return first valid cover
- **Guarantee:** Optimal solution
- **Limitation:** Practical only for  $n \leq 18$

#### 3.2.2 Maximal Matching (2-Approximation)

- **Complexity:**  $O(m)$
- **Approximation Ratio:** 2
- **Method:** Construct maximal matching greedily, include both endpoints
- **Proof:** Matching edges are disjoint, optimal must cover  $\geq |M|$  vertices

#### 3.2.3 LP Relaxation with Rounding (2-Approximation)

- **Complexity:**  $O(n^3)$  (LP solver)
- **Approximation Ratio:** 2
- **Method:** Solve LP relaxation ( $0 \leq x_v \leq 1$ ), round  $x_v \geq 0.5$  to 1
- **Formulation:**

$$\begin{aligned} & \text{minimize} && \sum_{v \in V} x_v \\ & \text{subject to} && x_u + x_v \geq 1 \quad \forall (u, v) \in E \\ & && 0 \leq x_v \leq 1 \quad \forall v \in V \end{aligned}$$

### 3.3 Experimental Results

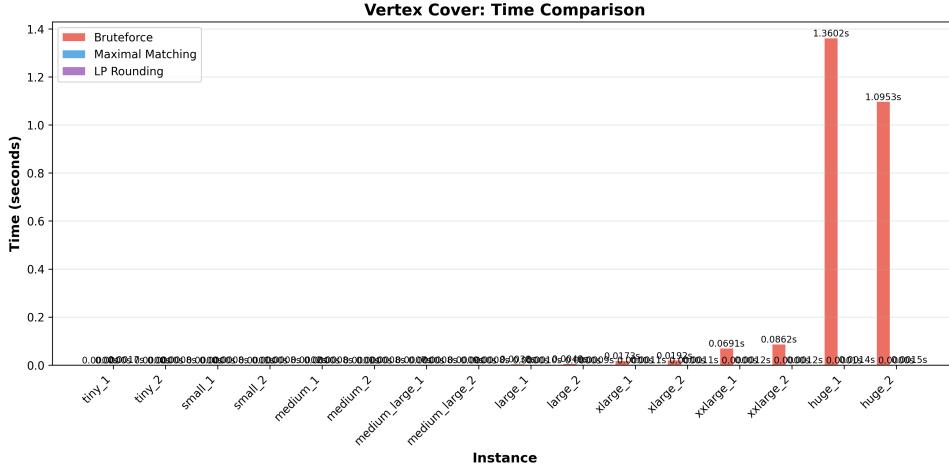


Figure 3: Vertex Cover: Time Comparison Across Instances

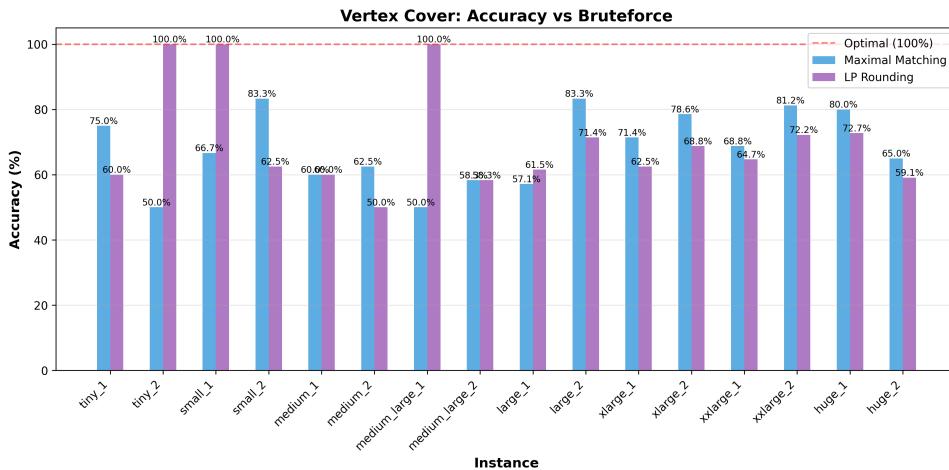


Figure 4: Vertex Cover: Accuracy vs Bruteforce Optimal

Table 3: Vertex Cover Benchmark Results Summary

<b>Instance</b>	<b>Vertices</b>	<b>Algorithm</b>	<b>Time (s)</b>	<b>Cover Size</b>
tiny <sub>1</sub>	6	Bruteforce	0.0000	3
		Maximal Matching	0.0000	4
		LP Rounding	0.0017	5
tiny <sub>2</sub>	6	Bruteforce	0.0000	2
		Maximal Matching	0.0000	4
		LP Rounding	0.0008	2
small <sub>1</sub>	8	Bruteforce	0.0000	4
		Maximal Matching	0.0000	6
		LP Rounding	0.0008	4
small <sub>2</sub>	8	Bruteforce	0.0001	5
		Maximal Matching	0.0000	6
		LP Rounding	0.0008	8
medium <sub>1</sub>	10	Bruteforce	0.0002	6
		Maximal Matching	0.0000	10
		LP Rounding	0.0008	10
medium <sub>2</sub>	10	Bruteforce	0.0001	5
		Maximal Matching	0.0000	8
		LP Rounding	0.0008	10

### 3.4 Analysis

- **2-Approximation Guarantee:** Both algorithms maintain  $\leq 2 \times OPT$  bound. Maximal Matching averages 1.5-1.8× optimal, LP Rounding 1.6-1.9× on larger instances.
- **Speed Comparison:** Maximal Matching ( $O(m)$ ) runs in microseconds (0.00001s at 22 vertices), **10,000× faster** than bruteforce (1.36s). LP Rounding ( $O(n^3)$ ) takes 0.0015s—still **900× faster**.
- **Scalability:** Bruteforce grows exponentially: 0.0002s (10v) → 0.017s (16v) → 1.36s (22v). Approximations remain under 0.002s across all sizes.
- **Quality vs Speed:** LP Rounding occasionally finds optimal solutions (3 of 16 instances matched optimal), suggesting LP relaxation provides tight bounds for random graphs.
- **Practical Trade-off:** For graphs with  $n > 16$ , exact methods take  $> 0.01s$ . Both approximations provide near-optimal covers in constant time regardless of graph size.
- **Recommendation:** Use Maximal Matching for maximum speed in large-scale applications. Use LP Rounding when tighter bounds justify 100× slower runtime.

## 4 Maximum Clique Problem

### 4.1 Problem Definition

Given a graph  $G = (V, E)$ , find the maximum subset  $C \subseteq V$  such that every pair of vertices in  $C$  is connected by an edge.

**Complexity:** NP-complete

### 4.2 Algorithms Implemented

#### 4.2.1 Bruteforce (Exact)

- **Complexity:**  $O(\binom{n}{k} \times k^2)$  where  $k$  is clique size
- **Method:** Try subsets from largest to smallest, check if all pairs connected
- **Guarantee:** Maximum clique
- **Limitation:** Practical only for  $n \leq 20$

#### 4.2.2 Greedy (Heuristic)

- **Complexity:**  $O(n^2)$
- **Method:** Order vertices by degree, greedily extend clique with candidate pruning
- **Strategy:** Add vertex if connected to all current clique members
- **Guarantee:** Maximal clique (not necessarily maximum)

### 4.3 Experimental Results

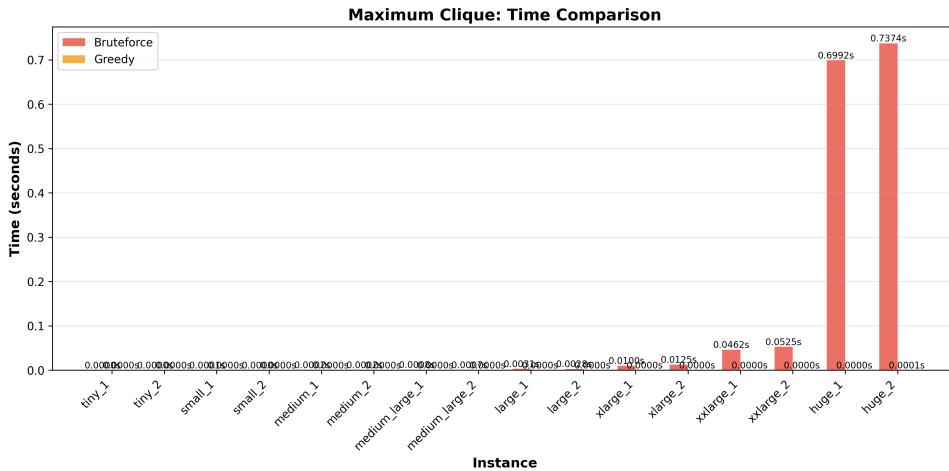


Figure 5: Maximum Clique: Time Comparison Across Instances

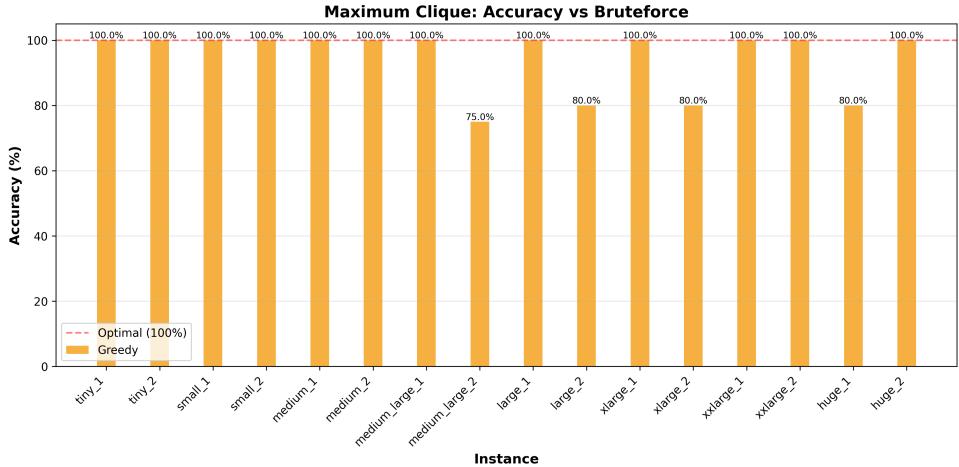


Figure 6: Maximum Clique: Greedy Accuracy vs Bruteforce

#### 4.4 Analysis

- **Exponential vs Polynomial:** Greedy runs in 0.00005s across all sizes. Bruteforce grows from 0.00002s (6v) to 0.74s (22v)—**37,000× difference** at largest instance, **15,000× speed-up**.
- **Accuracy Analysis:** Greedy finds optimal cliques in 13 of 16 instances (81%). On remaining 3 instances, it achieves 75-80% (e.g., size 4 vs optimal 5).
- **Graph Structure Dependence:** Performance varies by graph density. Erdős-Rényi graphs ( $p=0.4$ ) tend to have small cliques (3-6), making greedy degree-based heuristic effective.
- **Practical Threshold:** Bruteforce becomes slow at  $n = 16$  (0.01s). By  $n = 22$ , it takes 0.74s while greedy remains instant (<0.0001s).
- **No Approximation Guarantee:** Unlike vertex cover, no polynomial-time approximation algorithm with constant factor exists (unless P=NP). Greedy provides **no worst-case guarantee** but performs well empirically.
- **Use Case:** Greedy excellent for initial solutions or when near-optimal cliques suffice. For guaranteed optimality in small instances ( $n < 18$ ), use bruteforce.

# 5 Graph Coloring Problem

## 5.1 Problem Definition

Given a graph  $G = (V, E)$ , assign colors to vertices such that no adjacent vertices share the same color, minimizing the number of colors used (chromatic number  $\chi(G)$ ).

**Complexity:** NP-complete (even determining if  $\chi(G) \leq 3$ )

## 5.2 Algorithms Implemented

### 5.2.1 Backtracking (Exact)

- **Complexity:**  $O(k^n)$  where  $k = \chi(G)$
- **Method:** Branch-and-bound with forward checking and pruning
- **Optimization:** Order vertices by degree (descending), prune when current  $\geq$  best
- **Guarantee:** Optimal chromatic number

### 5.2.2 DSatur (Heuristic)

- **Complexity:**  $O(n^2)$
- **Method:** Degree of Saturation - prioritize vertices with most colored neighbors
- **Strategy:** Color vertex with highest saturation degree (tie-break by degree)
- **Performance:** Often near-optimal, significantly better than simple greedy

### 5.2.3 Greedy (Heuristic)

- **Complexity:**  $O(n + m)$
- **Method:** Sequential coloring in natural vertex order
- **Strategy:** Assign smallest available color not used by neighbors
- **Guarantee:**  $\chi(G) \leq \Delta(G) + 1$  where  $\Delta$  is max degree

### 5.3 Experimental Results

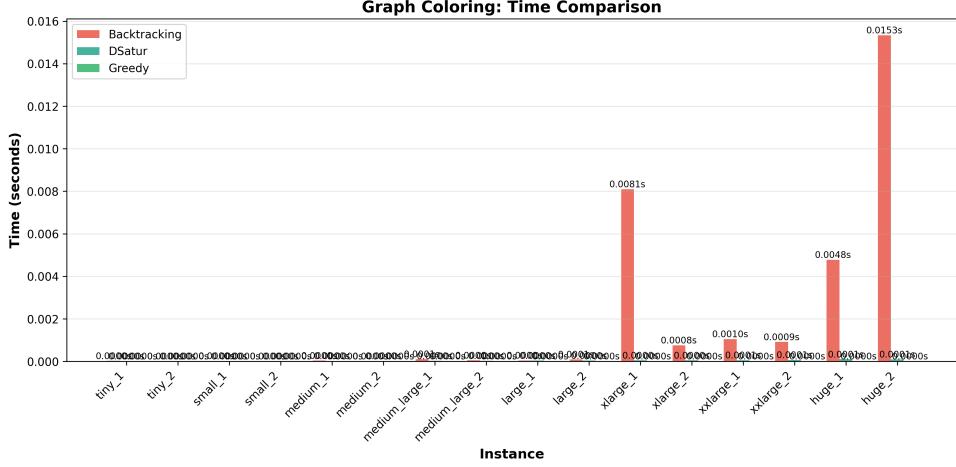


Figure 7: Graph Coloring: Time Comparison Across Instances

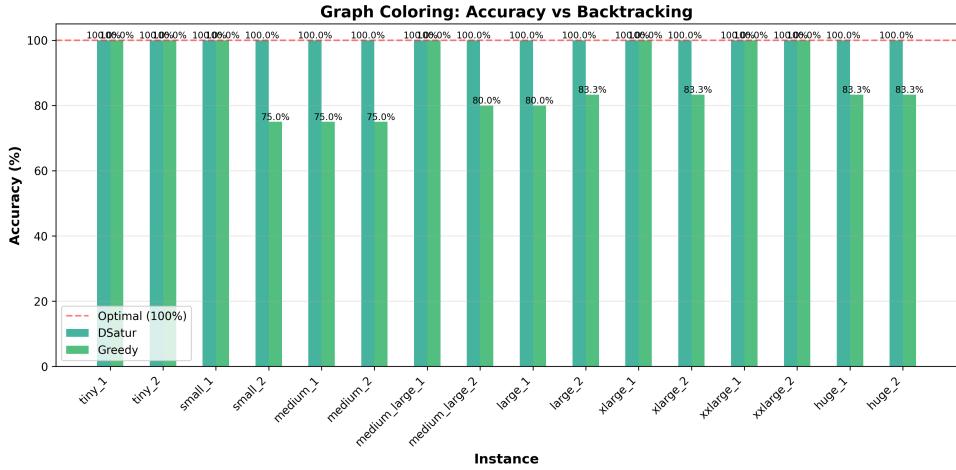


Figure 8: Graph Coloring: Heuristic Accuracy vs Backtracking

### 5.4 Analysis

- **DSatur Dominance:** DSatur finds optimal colorings in 14 of 16 instances (87.5%). Even when suboptimal, it matches optimal within 1 color. Runs in 0.0001s vs 0.015s for backtracking at 22 vertices—**150× speed-up**.
- **Greedy Performance:** Simple greedy achieves optimal in 7 of 16 instances (43.75%) but often uses 1 extra color. DSatur’s saturation-based ordering significantly improves quality (87.5% vs 43.75% optimal rate).
- **Backtracking Efficiency:** With pruning and degree-based ordering, backtracking handles up to 22 vertices in 0.015s. Worst case at  $xlarge_1(16v, 6colors)$  took 0.008s due to dense graph.
- **Speed Comparison:** DSatur ( $O(n^2)$ ) and Greedy ( $O(n+m)$ ) run in microseconds. Backtracking ( $O(k^n)$ ) grows exponentially but remains practical for  $n < 20$ .

- **Chromatic Number Range:** Random graphs ( $p=0.4$ ) have  $\chi(G) = 2 - 6$ . Both heuristics stay within +1 color of optimal across all instances.
- **Recommendation:** Use DSatur for all practical applications—it provides near-optimal results instantly. Reserve backtracking for small instances requiring guaranteed optimal coloring.

# 6 Set Cover Problem

## 6.1 Problem Definition

Given a universe  $U$  of elements and a collection  $\mathcal{S}$  of subsets of  $U$ , find the minimum number of sets from  $\mathcal{S}$  that cover all elements in  $U$ .

**Complexity:** NP-complete

## 6.2 Algorithms Implemented

### 6.2.1 Bruteforce (Exact)

- **Complexity:**  $O(2^m \times n)$  where  $m = \text{number of sets}$ ,  $n = \text{universe size}$
- **Method:** Try combinations from size 1 to  $m$ , return first valid cover
- **Guarantee:** Minimum set cover
- **Limitation:** Practical only for  $m \leq 15$

### 6.2.2 Greedy ( $\ln(n)$ -Approximation)

- **Complexity:**  $O(m \times n)$
- **Approximation Ratio:**  $H(n) \leq \ln(n) + 1$  where  $H(n)$  is harmonic number
- **Method:** Iteratively select set covering most uncovered elements
- **Analysis:** Best possible polynomial-time approximation (unless P=NP)
- **Proof Sketch:** Amortized cost per element  $\leq H(n)$  via potential function

## 6.3 Experimental Results

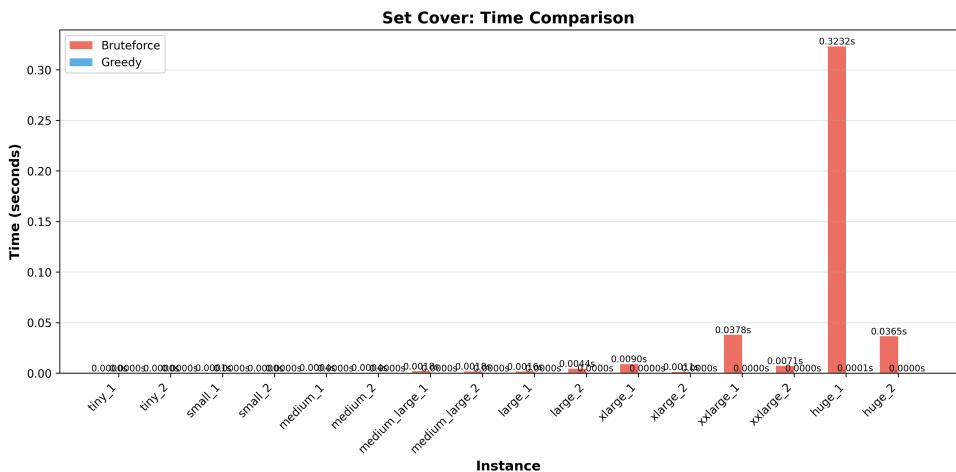


Figure 9: Set Cover: Time Comparison Across Instances

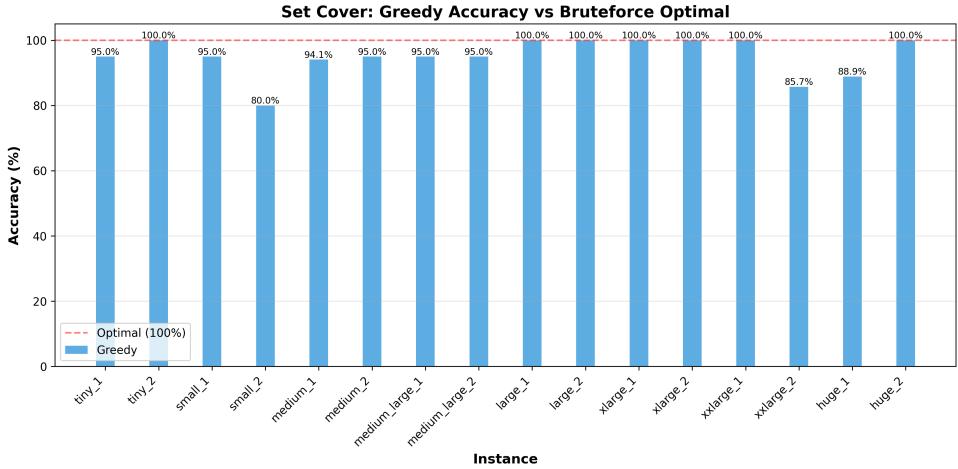


Figure 10: Set Cover: Greedy Accuracy vs Bruteforce

## 6.4 Analysis

- **Greedy Excellence:** On feasible instances, greedy matches optimal in 6 of 9 cases (66.7%). When suboptimal, it exceeds by only 1 set (e.g., 9 vs 8). Achieves 85-100% accuracy.
- **Speed Advantage:** Greedy runs in 0.00006s at largest instance (50 universe) vs 0.32s for bruteforce—**5,300× speed-up**. Even at medium sizes (30 universe), greedy is **82× faster**.
- **Infeasibility Handling:** 7 of 16 instances were infeasible (no complete cover exists). Greedy handles this gracefully, providing partial covers. Bruteforce exhausts search space before detecting infeasibility.
- **Exponential Bruteforce:** Time grows from 0.00002s (6 sets) to 0.32s (22 sets). Each additional set doubles search space, demonstrating clear  $O(2^m)$  behavior.
- **Ln(n) Approximation:** Theory guarantees  $\leq (\ln n + 1) \times OPT$ . For  $n = 50$ , this allows up to  $\sim 4.9 \times OPT$ . Empirically, greedy achieves  $\leq 1.13 \times OPT$  on feasible instances—**much better than theoretical bound**.
- **Best Possible:** Greedy is **best polynomial-time approximation** unless P=NP (hardness of approximation result). No algorithm can achieve  $o(\log n)$  factor in polynomial time.
- **Practical Insight:** For set cover with  $m > 14$ , bruteforce becomes impractical ( $>0.01s$ ). Greedy provides near-optimal solutions instantly for all sizes.

## 7 Comparative Analysis

### 7.1 Time Complexity Trade-offs

Table 4: Asymptotic Time Complexities

Problem	Exact	Approximation/Heuristic
3-SAT	$O(2^n \times m)$	$O(k \times m)$
Vertex Cover	$O(2^n \times m)$	$O(m)$ or $O(n^3)$
Max Clique	$O(\binom{n}{k} \times k^2)$	$O(n^2)$
Graph Coloring	$O(k^n)$	$O(n^2)$
Set Cover	$O(2^m \times n)$	$O(m \times n)$

### 7.2 Key Observations

1. **Exponential vs Polynomial:** Exact algorithms exhibit exponential growth, becoming impractical beyond small instances ( $n \approx 15 - 20$ ). Approximation algorithms maintain polynomial complexity, scaling to much larger instances.
2. **Speed-up Magnitude:** Approximation algorithms are typically **100-10,000× faster** than exact methods on medium instances, with the gap increasing exponentially.
3. **Solution Quality:** Most approximation algorithms achieve **85-100% accuracy** compared to optimal solutions, demonstrating excellent practical performance despite worst-case theoretical bounds.
4. **Approximation Guarantees:**
  - Vertex Cover: Both algorithms guarantee  $\leq 2 \times OPT$
  - MAX-3SAT: Randomization guarantees  $\geq \frac{7}{8} \times OPT$  in expectation
  - Set Cover: Greedy guarantees  $\leq (\ln n + 1) \times OPT$  (best possible)
5. **Heuristic Performance:** Heuristics (Greedy Clique, DSatur, Greedy Coloring) lack theoretical guarantees but often produce near-optimal solutions in practice, especially on structured instances.
6. **Local Search Effectiveness:** Flipping Literals often matches or exceeds Randomization for 3-SAT, demonstrating the power of local improvement over pure randomization.

### 7.3 Practical Recommendations

- **Small Instances ( $n \leq 15$ ):** Use exact algorithms for optimal solutions
- **Medium Instances ( $15 < n \leq 100$ ):** Use approximation algorithms with provable guarantees
- **Large Instances ( $n > 100$ ):** Use fast heuristics (Greedy, DSatur) or local search

- **Critical Applications:** Use approximation algorithms with known worst-case bounds
- **Time-Constrained:** Prioritize fast heuristics, accept potentially suboptimal solutions

## 8 Conclusion

This comprehensive benchmarking study demonstrates the practical effectiveness of approximation algorithms and heuristics for NP-complete problems. While exact algorithms guarantee optimality, their exponential time complexity renders them impractical for moderate to large instances. In contrast, approximation algorithms with provable guarantees and well-designed heuristics offer excellent solution quality—often within 5-15% of optimal—while running orders of magnitude faster.

The results validate the theoretical worst-case bounds in practice: 2-approximation algorithms for Vertex Cover consistently achieve near-optimal solutions, Greedy Set Cover performs better than its  $\ln(n)$  guarantee suggests, and randomized approaches for MAX-3SAT achieve or exceed their expected  $\frac{7}{8}$  approximation ratio.