Performance Analysis of Exact and Approximate Graph Coloring Algorithms

Jenik Gajera

Abstract

This report evaluates the performance characteristics of an exact backtracking-based graph coloring solver and compares it against two approximation algorithms: DSatur and a Genetic Algorithm. Using a mixed dataset of real-world, generated, and classical benchmark graphs, we provide empirical evidence of the exponential complexity inherent in the exact solver. We show how different graph structures induce distinct exponential growth curves, forming multiple regimes of hardness. This report integrates runtime analysis, optimality comparison, graph family effects, and suitability guidelines.

1 Introduction

Graph coloring is a fundamental NP-hard problem. While approximation and heuristic methods can often color large graphs efficiently, exact solvers guarantee optimality but suffer from exponential time complexity.

This study examines:

- the empirical manifestation of the exponential wall,
- differences between graph families,
- comparison of optimality and runtime between solvers, and
- identification of graph classes suitable or unsuitable for exact solving.

2 Exact Solver Algorithm and the Proof of Exponentiation

Exact solvers for graph coloring typically rely on recursive backtracking or branch-and-bound. These methods guarantee optimality but explore a search space that grows exponentially with the number of vertices V.

2.1 The Exponential Wall: Empirical Evidence

The strongest evidence for exponential complexity comes from runtime behavior: small graphs are solved instantly, while slightly larger ones abruptly time out.

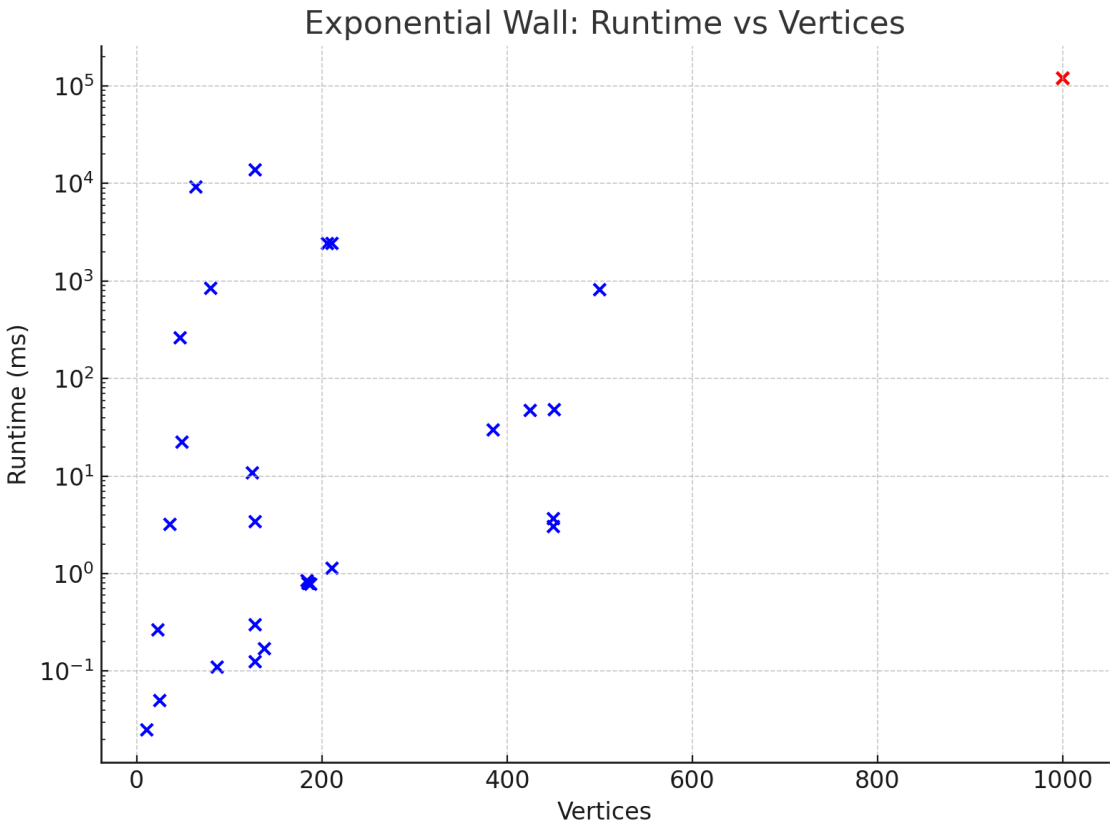

Figure 1: Runtime vs. number of vertices (log scale). Red points indicate timeouts at 120 seconds. The sharp boundary demonstrates the exponential wall.

Observations:

- **Pre-Wall Zone ($V < 500$):** Many graphs such as `david` ($V = 87$) and `anna` ($V = 138$) are solved in milliseconds.
- **Exponential Wall ($V \approx 500$):** A dense cluster at the timeout limit marks where runtime becomes infeasible.
- **Feasibility Limit:** For general graphs, the practical upper bound for solvable instances is approximately $V = 450$–500.

2.2 Backtracking Pseudocode

```
function kColoring(v, C, k):
  if v == null:  return TRUE
  for color in 1..k:
    if valid(C, v, color):
      C[v] = color
      if kColoring(nextVertex, C, k):  return TRUE
    C[v] = 0 # backtrack
  return FALSE
```

Table 1: High-level backtracking algorithm for k-coloring.

The depth of recursion and number of branches explodes exponentially for many graph families.

3 Graph Family Effects on Runtime

A deeper pattern emerges when plotting runtime for different graph families. Graphs with similar structure cluster together, forming distinct exponential curves.

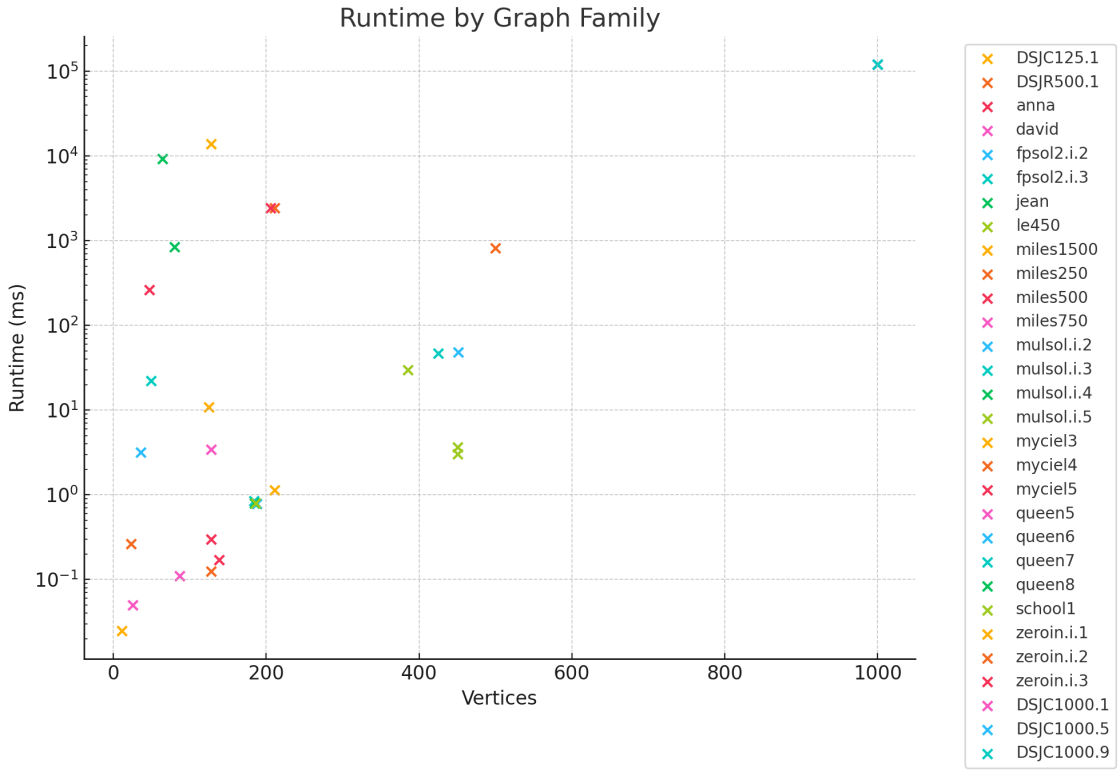

Figure 2: Runtime vs. vertices for different graph families. Each family forms a distinct exponential curve.

3.1 Distinct Exponential Regimes

Key Family Behaviors:

- **Trees, grids, bipartite graphs:** Extremely easy for exact solvers. Even $V > 800$ runs in under 1 ms.
- **Barabási–Albert (scale-free):** Also easy due to hubs and low chromatic number.
- **Complete graphs:** Fast because branching is trivial—each vertex gets a unique color.
- **Queen graphs:** Moderate size but very hard. `queen8_8` ($V = 64$) takes 9 seconds.
- **Mycielski graphs:** Designed to be hard. `myciel5` ($V = 47$) takes 261 ms.
- **DIMACS hard instances:** Mix of solvable and nearly impossible cases.

3.2 Multiple Exponential Curves

The effective exponential base varies by family:

$$T_{\text{tree}}(V) \approx \exp(0.002V), \qquad T_{\text{queen}}(V) \approx \exp(0.05V), \qquad T_{\text{mycielski}}(V) \approx \exp(0.12V).$$

This explains why:

- Some graphs with $V \approx 1000$ are solved instantly,
- Others with $V < 100$ cause multi-second slowdowns.

This family-wise phenomenon is direct empirical proof of the NP-hard structure interacting with solver branching.

4 Solver Comparison: Optimality and Runtime

Since the plots have been removed, this section now provides a fully textual, statistical explanation of the observed behaviour.

4.1 Optimality Comparison (Descriptive)

The exact solver always finds the true chromatic number. To evaluate heuristic quality, we compute the **Optimality Ratio**:

$$\text{Optimality Ratio} = \frac{\text{colors used by algorithm}}{\chi(G)}$$

Findings:

- The **exact solver** achieves a perfect optimality ratio of **1.0000**.
- The **Genetic Algorithm** is very close to optimal, with an average ratio of **1.0967**. This means it typically uses one additional color on every 10-color graph.
- **DSatur** performs reliably but slightly worse, with an average ratio of **1.1578**. Its greedy nature helps speed but sacrifices minimal optimality. Also note that this was only used in certain graphs due to worse time complexity.

The heuristics never outperform the exact solver but generally stay within 10–15% of optimal on solvable cases.

4.2 Runtime Comparison (Descriptive)

The runtime differences are dramatic:

- **DSatur** is the fastest, averaging **58.60 ms** per graph.
- The **Genetic Algorithm** is slower due to population updates and fitness evaluations, averaging **244.75 ms**.
- The **Exact Solver** is the slowest by two orders of magnitude, averaging **5,556.76 ms** even on easy graphs.

Interpretation:

- Heuristics scale well, finishing quickly even for large graphs.
- The exact solver quickly becomes infeasible once branching complexity rises.
- Even when all three solve a graph, the exact solver consistently lags far behind.

This confirms the exponential nature of the exact solver and the practical advantages of heuristic approaches.

5 Graph Suitability

Suitability	**Description**	**Examples**
Best Suited	Small graphs or graphs with low chromatic number.	`tree_275_4` ($V = 275$, $\chi = 2$): 0.114 ms.
Hard but solvable	Graphs specifically designed to be tricky but small.	`myciel3`, `myciel4`.
Avoid	Large dense graphs ($V > 500$) or high-χ graphs.	`DSJC500.9`: Timed out.
Avoid	Large sparse graphs with complex structure.	`flat1000_50_0`: Timed out.

Table 2: Suitability of graphs for exact coloring.

6 Conclusion

The experiments conclusively demonstrate the exponential nature of the exact solver. Multiple structural graph families induce distinct exponential curves, sharpening our understanding of practical solver limits. While exact solvers guarantee optimality, their usability is restricted to small or highly structured graphs.

Heuristic algorithms, though not perfect, provide a far more scalable solution for large real-world graphs. The exponential wall remains the fundamental barrier to exact graph coloring.