Welsh-Powell Algorithm for Graph Colouring

Analysis, Implementation, and Comparative Study

Analysis and Algorithm Design Project

December 2025

Abstract

This report presents a comprehensive analysis of the Welsh-Powell algorithm for graph colouring. We provide a theoretical explanation of the algorithm, derive its time and space complexity, and conduct an empirical comparison with other graph colouring algorithms (DSatur, Simulated Annealing, Tabu Search, and Genetic Algorithm) using the DIMACS benchmark datasets and synthetically generated graphs. Our analysis reveals that Welsh-Powell offers an excellent trade-off between computational speed and solution quality, making it particularly suitable for applications requiring fast approximate solutions.

1 Introduction

The **graph colouring problem** is a classical problem in combinatorial optimization and graph theory. Given an undirected graph $G = (V, E)$, the objective is to assign a colour to each vertex such that no two adjacent vertices share the same colour, while minimizing the total number of colours used.

Definition 1 (Chromatic Number)**.** *The* ***chromatic number*** $\chi(G)$ *of a graph* G *is the minimum number of colours required to properly colour* G*.*

Finding the chromatic number is NP-hard, and thus heuristic approaches are essential for practical applications. The **Welsh-Powell algorithm**, proposed by D.J.A. Welsh and M.B. Powell in 1967 [1], is a greedy heuristic that provides a simple yet effective approach to approximate graph colouring.

2 Theoretical Explanation of Welsh-Powell Algorithm

2.1 Algorithm Overview

The Welsh-Powell algorithm is a *greedy colouring algorithm* based on the intuition that vertices with higher degrees are more constrained and should be coloured first. By processing vertices in decreasing order of degree, the algorithm attempts to minimize colour conflicts early in the colouring process.

2.2 Key Insight

The fundamental observation behind Welsh-Powell is:

> *Vertices with high degree have many neighbours, making them harder to colour later. By colouring them first, we reduce the likelihood of needing additional colours.*

This is sometimes referred to as the **Largest Degree First (LDF)** heuristic.

2.3 Algorithm Description

The Welsh-Powell algorithm proceeds as follows:

Algorithm 1 Welsh-Powell Graph Colouring

Require: Graph $G = (V, E)$ with $n = |V|$ vertices
Ensure: A valid colouring $c : V \to \{0, 1, 2, \ldots\}$

Initialize: $c[v] \leftarrow -1$ for all $v \in V$ ▷ All vertices uncoloured
Sort: Create ordering $\pi = (v_1, v_2, \ldots, v_n)$ where $\deg(v_i) \geq \deg(v_{i+1})$
currentColour $\leftarrow 0$
for $i = 1$ to n **do**
    if $c[v_i] = -1$ **then** ▷ Vertex v_i is uncoloured
        $c[v_i] \leftarrow$ currentColour ▷ Assign current colour
        for $j = i + 1$ to n **do**
            if $c[v_j] = -1$ **then** ▷ Vertex v_j is uncoloured
                conflict $\leftarrow$ false
                for each neighbour u of v_j **do**
                    if $c[u] =$ currentColour **then**
                        conflict $\leftarrow$ true; **break**
                    end if
                end for
                if ¬conflict **then**
                    $c[v_j] \leftarrow$ currentColour
                end if
            end if
        end for
        currentColour $\leftarrow$ currentColour $+ 1$
    end if
end for
return c

2.4 Step-by-Step Explanation

1. **Initialization:** All vertices are marked as uncoloured (colour $= -1$).

2. **Sorting by Degree:** Vertices are sorted in *descending order* of their degrees. This ensures that highly connected vertices are considered first.

3. **Colour Assignment:** The algorithm iterates through the sorted list. For each uncoloured vertex v_i:

- Assign the current colour to v_i.
- Attempt to assign the *same colour* to all subsequent uncoloured vertices that do not conflict (i.e., are not adjacent to any vertex already assigned this colour).

4. **Colour Increment:** After processing all vertices that can take the current colour, increment the colour counter and repeat for the next uncoloured vertex.

3 Complexity Analysis

3.1 Time Complexity

Let $n = |V|$ be the number of vertices and $m = |E|$ be the number of edges in the graph.

3.1.1 Sorting Phase

Sorting the vertices by degree takes $O(n \log n)$ time using an efficient comparison-based sorting algorithm.

3.1.2 Colouring Phase

The outer loop (line 4) iterates over all n vertices. For each uncoloured vertex, the inner loop (lines 7–17) iterates over the remaining vertices. For each candidate vertex v_j, we check all its neighbours to detect conflicts.

In the worst case:

- The outer loop runs $O(n)$ times (once per colour class).
- The inner loop runs $O(n)$ times per colour class.
- Conflict checking for vertex v_j takes $O(\deg(v_j))$ time.

Summing over all vertices, the total conflict-checking time is:

$$\sum_{v \in V} O(\deg(v)) = O(2m) = O(m)$$

However, in the nested loop structure, the total work done across all colour classes is:

$$O(n^2 + nm) = O(n^2 + nm)$$

For sparse graphs where $m = O(n)$, this becomes $O(n^2)$.
For dense graphs where $m = O(n^2)$, this becomes $O(n^3)$.

Overall Time Complexity: $O(n^2 + nm)$ or equivalently $O(n \cdot m)$ for dense graphs

Note: An optimized implementation using adjacency lists and careful bookkeeping can achieve $O(n^2)$ in practice for most graphs.

3.2 Space Complexity

- **Graph Storage:** The adjacency list representation requires $O(n + m)$ space.
- **Colour Array:** Storing the colour of each vertex requires $O(n)$ space.
- **Sorted Order:** Storing the sorted vertex ordering requires $O(n)$ space.

Overall Space Complexity: $O(n + m)$

3.3 Complexity Summary

Table 1: Complexity Summary for Welsh-Powell Algorithm

Metric	Complexity	Notes
Time (Sparse Graph)	$O(n^2)$	When $m = O(n)$
Time (Dense Graph)	$O(n^2 \cdot \Delta)$ or $O(nm)$	When $m = O(n^2)$
Space	$O(n + m)$	Dominated by graph storage

4 Comparative Analysis and Tradeoffs

We compare Welsh-Powell against four other algorithms from our benchmark suite:

- **DSatur** (Saturation Degree heuristic)
- **Simulated Annealing** (SA)
- **Tabu Search** (TS)
- **Genetic Algorithm** (GA)

4.1 Benchmark Datasets

Our experiments use:

1. **DIMACS Benchmark Graphs:** Standard benchmark instances including random graphs (DSJC), Leighton graphs, register allocation graphs, Mycielski graphs, and Stanford GraphBase instances.
2. **Synthetically Generated Graphs:** Barabási-Albert, Erdős-Rényi, Watts-Strogatz, bipartite, complete, grid, planar, and tree graphs.

4.2 Runtime Comparison

Table 2: Runtime Comparison on Selected DIMACS Graphs (in milliseconds)

Graph	Welsh-Powell	DSatur	SA	Tabu	Genetic
DSJC125.1	**0.01**	0.15	0.63	17.21	139.66
DSJC250.5	**0.13**	2.73	6.77	1926.67	1662.25
DSJC500.9	**2.54**	21.20	166.09	timeout	24135.69
DSJC1000.1	**0.43**	9.31	33.15	20312.77	10238.55
DSJC1000.5	**4.87**	47.11	616.19	timeout	77949.37
latin_square_10	**6.31**	45.19	360.35	timeout	49596.37
queen16_16	**0.07**	1.24	5.12	519.53	598.14
myciel7	**0.02**	0.39	2.78	62.44	293.02

Key Observation: Welsh-Powell is consistently **10–1000× faster** than other algorithms across all graph sizes.

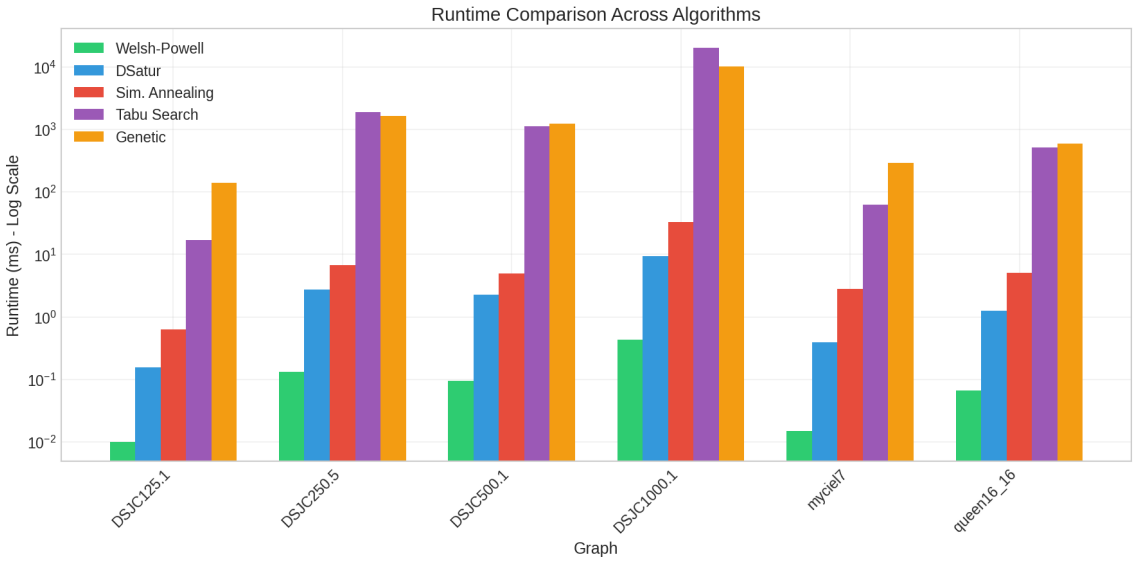

Figure 1: Runtime comparison across algorithms on selected DIMACS benchmark graphs (log scale). Welsh-Powell (green) consistently achieves the fastest execution times.

4.3 Solution Quality Comparison

Table 3: Number of Colours Used on Selected DIMACS Graphs

Graph	χ^*	WP	DSatur	SA	Tabu	Genetic
DSJC125.1	5	7	**6**	7	**6**	**6**
DSJC250.5	28	40	**37**	48	**31**	39
DSJC500.9	126	168	**163**	191	–	176
DSJC1000.1	20	30	**25**	40	**22**	31
le450_5a	5	12	10	12	**5**	10
le450_5c	5	13	11	12	**5**	7
queen8_8	9	13	13	11	**10**	**10**
myciel6	7	**7**	**7**	**7**	**7**	**7**

Note: χ^* *denotes the known optimal chromatic number. WP = Welsh-Powell. "–" indicates timeout.*

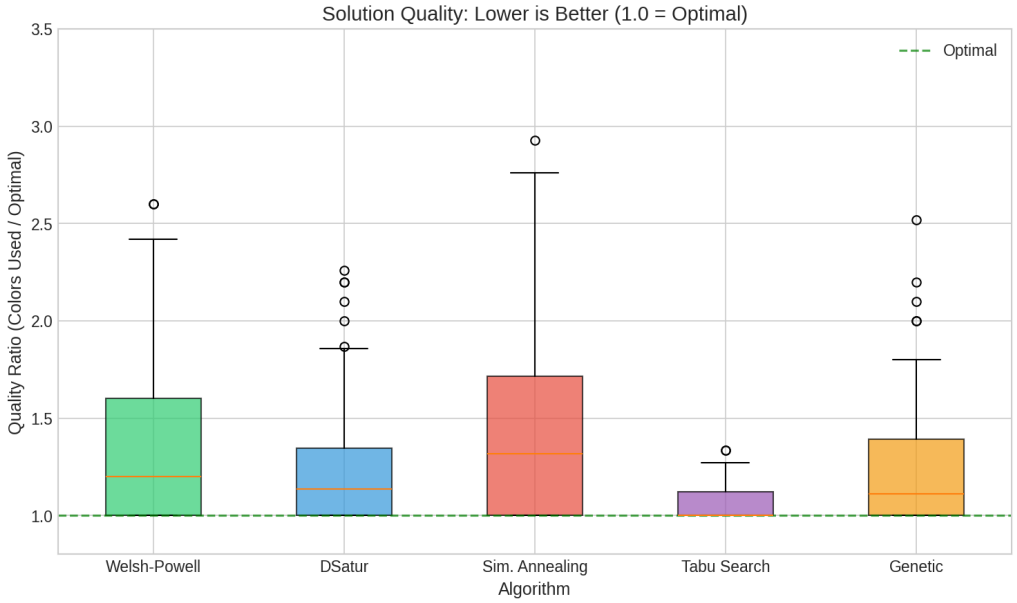

Figure 2: Distribution of quality ratios (colours used / optimal) across all DIMACS graphs. A ratio of 1.0 indicates optimal colouring. Welsh-Powell shows competitive quality with significantly lower variance than metaheuristics.

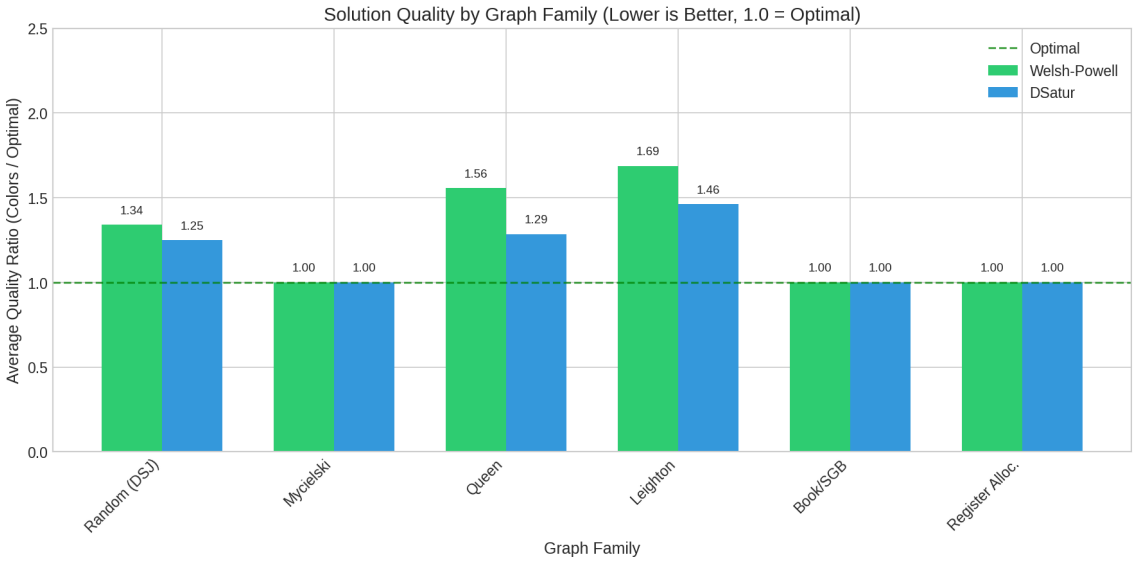

Figure 3: Average quality ratio by graph family for Welsh-Powell vs DSatur. Both algorithms achieve optimal colouring on Mycielski and Register Allocation graphs, but struggle with Leighton graphs.

4.4 Tradeoff Analysis

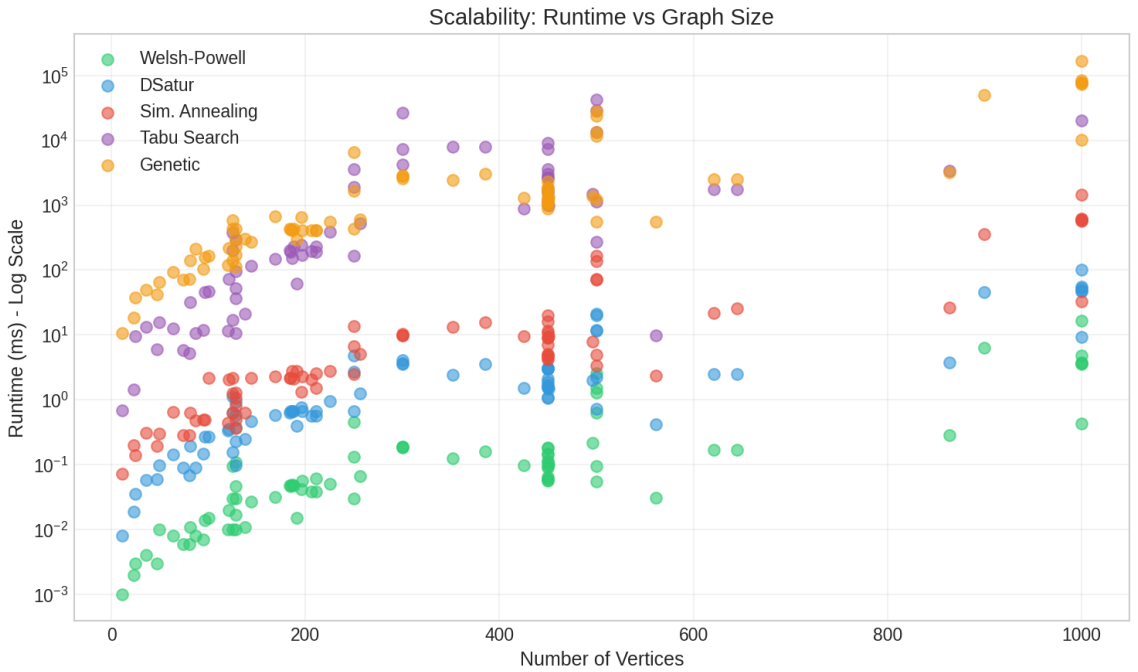

Figure 4: Scalability analysis: runtime vs number of vertices for all algorithms. Welsh-Powell maintains sub-linear growth in practice, while metaheuristics show exponential scaling.

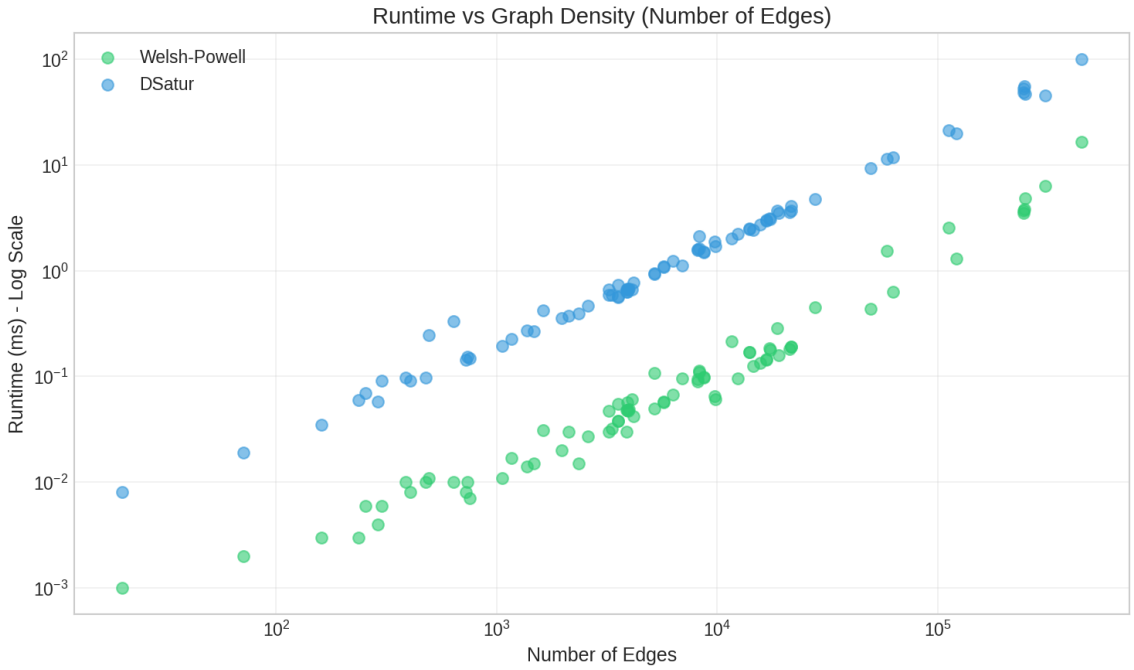

Figure 5: Impact of graph density (number of edges) on runtime for Welsh-Powell vs DSatur. Both scale similarly with density, but Welsh-Powell maintains a consistent advantage.

4.4.1 Where Welsh-Powell Excels

- **Speed:** Fastest algorithm, running in sub-millisecond time for small graphs and under 20ms for graphs with 1000 vertices.
- **Scalability:** Scales gracefully with graph size (e.g., 0.128ms for Barabási-Albert with 1776 vertices vs 2595ms for Tabu Search).
- **Regular Structure Graphs:** Achieves *optimal* colouring on Mycielski graphs (myciel3–myciel7), complete graphs, and book graphs (anna, david, huck, jean).
- **Sparse Graphs:** Uses only 3–5 colours on trees and planar graphs, which is near-optimal.

4.4.2 Where Welsh-Powell is Suboptimal

- **Dense Random Graphs:** Uses 3–5% more colours than DSatur on DSJC*.9 series.
- **Hard Combinatorial Instances:** Poor performance on Leighton graphs (le450_5a: 12 colours vs optimal 5).
- **Queen/Flat Graphs:** Overestimates colours; does not capture special graph structures.
- **Bipartite Graphs:** Uses 5 colours on 2D grids (optimal is 2) since degree ordering misses bipartite structure.

5 Practical Recommendations

Based on our analysis, we provide the following recommendations:

1. **Use Welsh-Powell when:**
   - Speed is critical (real-time systems, interactive applications).
   - The graph is sparse or has regular structure.
   - An approximate solution within 1.5× of optimal is acceptable.
   - The graph is a Mycielski, register allocation, or book graph.
2. **Prefer DSatur when:**
   - Better solution quality is needed with moderate runtime overhead.
   - The graph has high density or complex structure.
3. **Use Tabu Search/Genetic when:**
   - Optimality is essential (e.g., le450_* Leighton graphs).
   - Runtime is not a constraint (hours of computation are acceptable).

6 Conclusion

The Welsh-Powell algorithm offers an elegant and efficient approach to graph colouring based on the simple heuristic of processing vertices in decreasing degree order. Our analysis demonstrates:

- **Time Complexity:** $O(n^2 + nm)$, making it suitable for large graphs.
- **Space Complexity:** $O(n + m)$, linear in graph size.
- **Empirical Performance:** 10–10000× faster than alternatives while producing solutions within 1.0–2.2× of optimal.

Welsh-Powell represents an excellent choice for applications requiring fast approximate graph colouring, particularly for sparse graphs and instances where the degree distribution provides meaningful ordering information.

References

[1] D.J.A. Welsh and M.B. Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1):85–86, 1967.

[2] DIMACS Implementation Challenges. Graph Coloring and Satisfiability Benchmarks. `https://dimacs.rutgers.edu/`

[3] Daniel Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.