A Formal Proof of Wigderson's Graph Coloring Algorithm

Abstract

We present a complete formal verification of Wigderson's 1983 graph coloring algorithm in the Coq proof assistant. The algorithm produces an $O(\sqrt{n})$-coloring of any 3-colorable graph on n vertices. Our development covers all the necessary infrastructure: an adjacency-list graph representation using positive maps and sets, a greedy coloring algorithm, induced subgraphs, neighborhoods, independent sets, bipartiteness, BFS-based 2-coloring of bipartite graphs, and both phases of Wigderson's algorithm. The main results are: (1) the algorithm produces a proper coloring of any 3-colorable undirected graph, and (2) every color assigned satisfies $c_i \leq 3\lfloor n/(k+2)\rfloor + k + 2$, giving $O(\sqrt{n})$ colors when $k = \lfloor\sqrt{n}\rfloor$. The formalization is approximately 6,600 lines of Coq, with extensive use of the CoqHammer automated tactic for proof automation. No axioms beyond the standard Coq library are used, and no proofs are left admitted.

1 Introduction

The graph coloring problem asks: given an undirected graph G, assign colors to vertices so that no two adjacent vertices share a color. The *chromatic number* $\chi(G)$ is the minimum number of colors needed. Determining $\chi(G)$ is NP-hard in general, and even deciding whether $\chi(G) \leq 3$ is NP-complete.

In 1983, Wigderson [1] observed that while *finding* a 3-coloring is hard, one can *use* the existence of a 3-coloring to obtain an efficient algorithm that colors a 3-colorable graph with $O(\sqrt{n})$ colors, where $n = |V(G)|$. The key insight is that in any 3-colorable graph, the neighborhood of every vertex is 2-colorable (bipartite), so the neighborhood of a high-degree vertex can be efficiently 2-colored, removing many vertices at once.

In this paper we present a complete formalization of this result in the Coq proof assistant. Our development proceeds as follows:

1. We represent graphs as finite maps from vertices to adjacency sets, using Coq's `PositiveMap` and `PositiveSet` modules.
2. We develop the theory of induced subgraphs, neighborhoods, degrees, and independent sets.
3. We prove that a greedy coloring algorithm produces valid colorings.
4. We formalize bipartiteness and prove the equivalence between bipartiteness and 2-colorability.
5. We implement and verify a BFS-based 2-coloring algorithm for bipartite graphs.
6. We implement both phases of Wigderson's algorithm, prove the main correctness theorem, and formally verify the $O(\sqrt{n})$ color bound.

2 Preliminaries: Graph Representation

2.1 Core Definitions

We represent graphs using Coq's `PositiveMap` and `PositiveSet` modules, which provide efficient finite maps and sets over the type of positive integers.

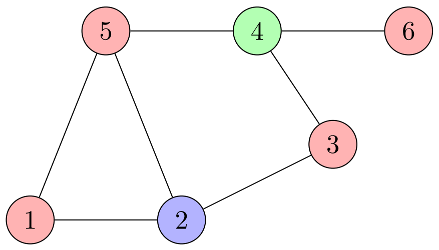


Figure 1: A small 3-colorable graph with a proper 3-coloring indicated by vertex shading.

Definition 2.1 (Graph)**.** A *graph* is a finite map g : `node` $\to$ `nodeset`, where `node` $= \mathbb{Z}^+$ and `nodeset` $= \mathcal{P}_{\text{fin}}(\mathbb{Z}^+)$. For a vertex i, its *adjacency set* is

$$\text{adj}(g,i) = \begin{cases} s & \text{if } g(i) = \texttt{Some } s, \\ \emptyset & \text{if } g(i) = \texttt{None}. \end{cases}$$

The *vertex set* (nodes) of g is $\text{nodes}(g) = \text{dom}(g) = \{k \mid k \in \text{dom}(g)\}$.

Definition 2.2 (Graph Properties)**.** A graph g is:

- *well-formed* if for all i, j: $j \in \text{adj}(g,i) \implies j \in \text{dom}(g)$.
- *undirected* if for all i, j: $j \in \text{adj}(g,i) \implies i \in \text{adj}(g,j)$.
- *irreflexive* (no self-loops) if for all i: $i \notin \text{adj}(g,i)$.

Lemma 2.3. *Every undirected graph is well-formed.*

Proof. If $j \in \text{adj}(g,i)$ and g is undirected, then $i \in \text{adj}(g,j)$, which requires $g(j) = \texttt{Some } s$ for some s, hence $j \in \text{dom}(g)$. □

2.2 Colorings

Definition 2.4 (Coloring)**.** A *coloring* is a finite map f : `node` $\to$ `node`, assigning color values to vertices.

Definition 2.5 (Valid Coloring)**.** A coloring f is *valid* (or *OK*) with respect to a palette P and graph g if for every edge (i, j) in g:

1. if $f(i) = c_i$ then $c_i \in P$ (palette membership), and
2. if $f(i) = c_i$ and $f(j) = c_j$ then $c_i \neq c_j$ (proper coloring).

Formally: $\forall\, i\, j,\ j \in \text{adj}(g,i) \implies (\forall c_i,\ f(i) = c_i \implies c_i \in P) \wedge (\forall c_i\, c_j,\ f(i) = c_i \implies f(j) = c_j \implies c_i \neq c_j)$.

Note that this definition permits *partial* colorings: vertices with $f(i) = \texttt{None}$ impose no constraint.

Definition 2.6 (Complete Coloring, n-Coloring)**.** A coloring f is *complete* for (P, g) if it is valid and every vertex of g is colored: $\forall i,\ i \in \text{dom}(g) \implies i \in \text{dom}(f)$.

An *n-coloring* is a coloring f with a palette P satisfying $|P| = n$ and every color used by f belongs to P. A *three-coloring* is a 3-coloring; a *two-coloring* is a 2-coloring.

Lemma 2.7. *If g admits a complete coloring, then g has no self-loops.*

Lemma 2.8. *If g' is a subgraph of g and f is a valid coloring of g, then f is a valid coloring of g'.*

3 Map Operations: Restriction and Union

3.1 Map Restriction

Definition 3.1 (Restriction)**.** For a map m and a set s, the *restriction* $\text{restrict}(m, s)$ retains only the entries whose keys belong to s:

$$\text{restrict}(m, s)(k) = \begin{cases} m(k) & \text{if } k \in s, \\ \texttt{None} & \text{otherwise.} \end{cases}$$

This is implemented as $\texttt{WP.filter}\ (\lambda\, k\, v.\ k \in s)\ m$.

Lemma 3.2 (Restriction Characterization)**.** $\text{restrict}(m, s)(k) = \texttt{Some}\ v$ *if and only if* $k \in s$ *and* $m(k) = \texttt{Some}\ v$*.*

Lemma 3.3 (Adjacency under Restriction)**.** *For a graph g and set s:* $i \in \text{adj}(\text{restrict}(g, s), j) \iff i \in \text{adj}(g, j) \wedge j \in s$*.*

3.2 Map Union

Definition 3.4 (Left-Biased Map Union)**.** The *union* of two maps c and d, written $\text{Munion}(c, d)$, is defined by folding insertion of all entries of c into d: $\text{Munion}(c, d) = \texttt{fold}\ (\lambda\, k\, v\, m.\ m[k \mapsto v])\ c\ d$. When both maps define a key, the left map's value takes priority.

Definition 3.5 (Map Disjointness)**.** Two maps f, g are *disjoint*, written $\texttt{Mdisjoint}(f, g)$, if $\text{dom}(f) \cap \text{dom}(g) = \emptyset$.

Lemma 3.6 (Union Case Analysis)**.** *If* $\text{Munion}(c, d)(i) = v$*, then* $c(i) = v$ *or* $d(i) = v$*.*

Lemma 3.7 (Left Priority)**.** *If* $c(i) = v$*, then* $\text{Munion}(c, d)(i) = v$*.*

Lemma 3.8 (Union Membership)**.** $i \in \text{dom}(\text{Munion}(c, d)) \iff i \in \text{dom}(c) \vee i \in \text{dom}(d)$*.*

4 Subgraphs, Neighborhoods, and Independent Sets

4.1 Induced Subgraphs

Definition 4.1 (Induced Subgraph)**.** The *induced subgraph* of g on a vertex set s is

$$\texttt{subgraph\_of}(g, s) = \texttt{map}\ (\lambda\, a.\ s \cap a)\ (\text{restrict}(g, s)).$$

This restricts both the vertex set and every adjacency set to s.

Definition 4.2 (Subgraph Predicate)**.** We write $\texttt{is\_subgraph}(g', g)$ when $\text{nodes}(g') \subseteq \text{nodes}(g)$ and for all v, $\text{adj}(g', v) \subseteq \text{adj}(g, v)$.

Lemma 4.3 (Adjacency in Induced Subgraphs)**.** $i \in \text{adj}(\textit{subgraph\_of}(g, s), j) \iff i \in \text{adj}(g, j) \wedge i \in s \wedge j \in s$*.*

Lemma 4.4. *$\textit{subgraph\_of}(g, s)$ is a subgraph of g. Induced subgraphs preserve undirectedness.*

4.2 Node Removal

Definition 4.5 (Removing Nodes)**.** For a set s of vertices to remove:

$$\texttt{remove\_nodes}(g, s) = \texttt{map}\ (\lambda\, a.\ a \setminus s)\ (\text{restrict}(g, \text{nodes}(g) \setminus s)).$$

Lemma 4.6 (Adjacency after Removal)**.** $i \in \text{adj}(\textit{remove\_nodes}(g, s), j) \iff i \in \text{adj}(g, j) \wedge i \notin s \wedge j \notin s$.

Lemma 4.7. *Removing a non-empty set of vertices that intersects* $\text{dom}(g)$ *strictly decreases the graph's cardinality. Removal preserves undirectedness and irreflexivity.*

4.3 Neighborhoods

Definition 4.8 (Neighborhood)**.** The *(open) neighborhood* of vertex v in g is the induced subgraph on $\text{adj}(g, v)$, with v itself removed:

$$\texttt{neighborhood}(g, v) = \texttt{remove\_node}(v, \texttt{subgraph\_of}(g, \text{adj}(g, v))).$$

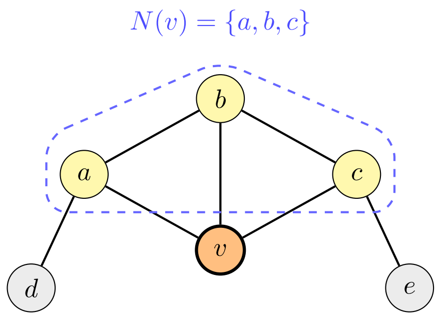


Figure 2: Vertex v (orange) with its open neighborhood $N(v) = \{a, b, c\}$ (yellow, enclosed by dashed boundary). Edges within the neighborhood (a–b, b–c) form the induced subgraph. Vertices d, e are outside.

Lemma 4.9 (Neighborhood Nodes)**.** *For an irreflexive undirected graph* g*,*

$$\text{nodes}(\textit{neighborhood}(g, v)) = \text{adj}(g, v).$$

Lemma 4.10. *The neighborhood of* v *is a subgraph of* g *and inherits undirectedness.*

4.4 Degrees and Maximum Degree

Definition 4.11 (Degree, Maximum Degree)**.** The *degree* of v in g is $\deg(v, g) = |\,\text{adj}(g, v)|$ (when $v \in g$). The *maximum degree* is $\text{maxdeg}(g) = \max_{v \in g} \deg(v, g)$.

Lemma 4.12. *For any vertex* v *with adjacency set* e*:* $|e| \leq \text{maxdeg}(g)$*. Maximum degree is non-increasing under subgraph formation.*

4.5 Independent Sets

Definition 4.13 (Independent Set)**.** s is an *independent set* in g if no two vertices in s are adjacent: $\forall\, i\, j \in s,\ i \notin \mathrm{adj}(g, j)$.

Theorem 4.14 (Maximum Degree Extraction)**.** *In an undirected irreflexive graph g with $d = \mathrm{maxdeg}(g)$, iteratively extracting vertices of degree d (removing each from the graph before extracting the next) yields an independent set.*

Proof. By well-founded induction on the cardinality of g. When a vertex v of degree d is removed, any remaining vertex w still having degree d in the reduced graph cannot have been adjacent to v (since removal would have decreased w's degree). Hence the extracted set has no internal edges. □

5 Greedy Coloring

Definition 5.1 (Select)**.** $\mathtt{select}(K, g)$ repeatedly finds a vertex of degree less than K in g, removes it, and records it. The output is a list of vertices in removal order.

Definition 5.2 (Greedy Coloring)**.** $\mathtt{color}(P, g) = \mathtt{fold\_right}\ (\mathtt{color1}\ P\ g)\ \emptyset\ (\mathtt{select}\ |P|\ g)$, where `color1` assigns the first available color from P to each vertex.

Theorem 5.3 (Greedy Coloring Correctness)**.** *For any irreflexive undirected graph g and palette P: $\mathtt{color}(P, g)$ is a valid coloring of g with respect to P.*

Proof. By induction on the select list, using the fact that `color1` preserves the validity of an existing coloring: the chosen color differs from all colors already assigned to neighbors. □

6 Connectivity and Bipartiteness

6.1 Walks and Reachability

Definition 6.1 (Walk, Reachability)**.** A *step* from x to y in g means $y \in \mathrm{adj}(g, x)$. A *walk* from x to z through l is an inductive relation: either $l = []$ and $x = z$ (with $x \in g$), or $l = y :: l'$ with a step from x to y and a walk from y to z through l'. Vertex v is *reachable* from u if there exists a walk from u to v.

6.2 Bipartiteness

Definition 6.2 (Bipartition, Bipartite)**.** A *bipartition* of g is a pair (L, R) such that $L \cap R = \emptyset$, $L \cup R = \mathrm{nodes}(g)$, and both L and R are independent sets in g. A graph is *bipartite* if it admits a bipartition.

Lemma 6.3 (Walk Parity)**.** *In a bipartite graph (L, R), if a walk of length ℓ starts at a vertex in L, then it ends in L if ℓ is even, and in R if ℓ is odd (and symmetrically for walks starting in R).*

Proof. By induction on the walk. Each step crosses from L to R or vice versa (since both sides are independent sets), toggling the parity. □

Definition 6.4 (Bicolor)**.** Given a bipartition (L, R) and two colors $c_1 \neq c_2$: $\mathtt{bicolor}(L, R, c_1, c_2) = \mathrm{Munion}(\mathtt{constant\_color}(L, c_1), \mathtt{constant\_color}(R, c_2))$.

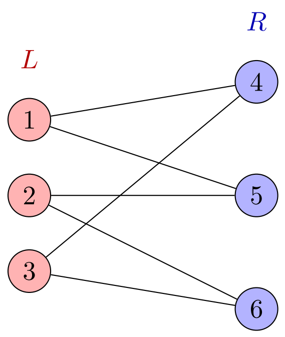


Figure 3: A bipartite graph with sides $L = \{1, 2, 3\}$ (red) and $R = \{4, 5, 6\}$ (blue). All edges cross between sides.

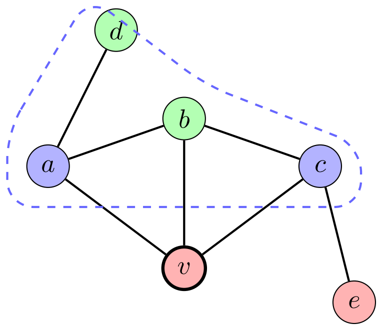
Theorem 6.5 (Bipartite $\iff$ 2-Colorable)**.** *For an undirected graph g: g is bipartite if and only if there exists a complete 2-coloring of g.*

Proof. ($\Rightarrow$) Given a bipartition (L, R), the bicolor construction yields a complete valid 2-coloring: vertices in L get color c_1, vertices in R get c_2; independence of each side ensures properness.

($\Leftarrow$) Given a complete 2-coloring f with palette $\{c_1, c_2\}$, define $L = \{v \mid f(v) = c_1\}$ and $R = \text{nodes}(g) \setminus L$. Since f is proper, both sides are independent. $\square$

Theorem 6.6 (Neighborhoods of 3-Colorable Graphs are Bipartite)**.** *Let g be an undirected graph with a complete 3-coloring (f, P). Then for every vertex v, the neighborhood $\texttt{neighborhood}(g, v)$ is bipartite.*

Proof. If $v \notin g$, the neighborhood is empty and trivially bipartite. Otherwise, $f(v) = c$ for some color $c \in P$. Since $|P| = 3$, removing c leaves a 2-element palette $P' = P \setminus \{c\}$. The restriction of f to the neighborhood avoids color c (by properness of f across the edges from v to its neighbors), yielding a complete 2-coloring of the neighborhood. By Theorem 6.5, the neighborhood is bipartite. $\square$



$N(v) = \{a, b, c\}$: only blue and green
(bipartite under 2 remaining colors)

Figure 4: In a 3-colored graph, vertex v (red) has neighbors using only the other two colors (blue, green), making the neighborhood bipartite.

7 BFS-Based 2-Coloring of Bipartite Graphs

We now describe an algorithm that, given a bipartite graph, constructs a valid 2-coloring by BFS-based layer expansion.

7.1 Neighborhood of a Set

Definition 7.1. For a graph g and set s, the *neighborhood of the set* is: $\mathrm{nbs}(g, s) = \bigcup_{v \in s} \mathrm{adj}(g, v)$.

Lemma 7.2. *$i \in \mathrm{nbs}(g, s) \iff \exists v \in s,\ i \in \mathrm{adj}(g, v)$.*

7.2 Force Layers

Definition 7.3 (Force Layers)**.** $\texttt{force\_layers}(g, L, R, F_L, F_R, k)$ iteratively expands two frontier sets, alternating BFS layers:

- Compute $R_{\mathrm{add}} = \mathrm{nbs}(g, F_L) \setminus (L \cup R)$ (new right-side vertices).
- Compute $L_{\mathrm{add}} = \mathrm{nbs}(g, F_R) \setminus (L \cup R)$ (new left-side vertices).
- Recurse with updated sets $L \cup L_{\mathrm{add}}$, $R \cup R_{\mathrm{add}}$, and the new frontiers.
- Stop after k iterations (where $k = |\,\mathrm{nodes}(g)|$ suffices for convergence).

Definition 7.4 (Force Component, Force All)**.** $\texttt{force\_component}(g, \mathrm{seed}, c_1, c_2)$ runs `force_layers` from a seed vertex, then 2-colors the resulting (L, R) using `bicolor`.

$\texttt{force\_all}(g, c_1, c_2)$ iterates: pick any vertex as seed, 2-color its connected component, remove those vertices, and recurse.

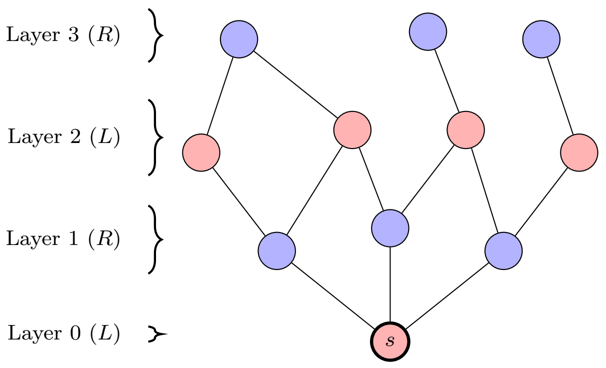


Figure 5: BFS layers expanding from seed s. Even layers (0, 2) are assigned to L (red), odd layers (1, 3) to R (blue). In a bipartite graph, this respects the true bipartition.

7.3 Key Invariants

Lemma 7.5 (Layers Respect Bipartition)**.** *If g has a bipartition (B_L, B_R) and the seed lies in B_L, then after forcing, $L \subseteq B_L$ and $R \subseteq B_R$.*

Proof. By induction on the number of BFS iterations. The initial state satisfies $\{\mathrm{seed}\} \subseteq B_L$ and $\emptyset \subseteq B_R$. At each step, neighbors of left-frontier vertices land in B_R (since B_L is independent), and vice versa. $\square$

Lemma 7.6 (Reached Set is Closed)**.** *For an undirected bipartite graph g with seed $\in \mathrm{nodes}(g)$:* $\mathrm{nbs}(g, \texttt{reached}(g, \mathit{seed})) \subseteq \texttt{reached}(g, \mathit{seed})$.

Proof. After sufficiently many BFS iterations (bounded by $|\operatorname{nodes}(g)|$), no new vertices are discovered. The reached set is then closed under adjacency, since any undiscovered neighbor would have been added during expansion. □

Lemma 7.7 (Force Component OK). *$\texttt{force\_component}(g, seed, c_1, c_2)$ produces a complete valid 2-coloring of the induced subgraph on the reached vertices.*

7.4 Combining Components

Lemma 7.8 (Coloring Union with No Cross Edges). *Let $S \subseteq V(g)$ be closed under adjacency ($\operatorname{nbs}(g, S) \subseteq S$). If f_1 is a valid coloring of $\texttt{subgraph\_of}(g, S)$ with $\operatorname{dom}(f_1) = S$, and f_2 is a valid coloring of $\texttt{remove\_nodes}(g, S)$, then $\operatorname{Munion}(f_1, f_2)$ is a valid coloring of g.*

Proof. For any edge (i, j) in g, either both endpoints are in S (handled by f_1), or both are outside S (handled by f_2). The closure property prevents cross-component edges. □

Theorem 7.9 (Force All Correctness). *For an undirected bipartite graph g with $c_1 \neq c_2$, the coloring $\texttt{force\_all}(g, c_1, c_2)$ is valid for g with palette $\{c_1, c_2\}$.*

Proof. By well-founded induction on $|V(g)|$. At each step, `force_component` correctly 2-colors the reached set (Lemma 7.7), which is closed under adjacency (Lemma 7.6). By Lemma 7.8, the union with the recursive coloring of the reduced graph is valid. Bipartiteness is preserved under vertex removal. Termination follows because the reached set is non-empty (it contains the seed), so the graph strictly shrinks. □

8 Wigderson's Algorithm

8.1 Phase 1: High-Degree Vertex Processing

Definition 8.1 (Two-Color Neighborhood).

$$\texttt{two\_color\_nbd}(g, v, c_1, c_2) = \texttt{force\_all}(\texttt{neighborhood}(g, v), c_1, c_2).$$

This uses the BFS-based 2-coloring from Section 7 on the (bipartite) neighborhood.

Definition 8.2 (Phase 1). Given a degree threshold k and starting color c:

```
1:  function PHASE1($k, c, g$)
2:      if $\exists v \in g$ with $\deg(v, g) > k$ then
3:          $m' \leftarrow \texttt{two\_color\_nbd}(g, v, c{+}1, c{+}2)$
4:          $g' \leftarrow \texttt{remove\_nodes}(g, \{v\} \cup \operatorname{nodes}(\texttt{neighborhood}(g, v)))$
5:          $(f_2, g_2) \leftarrow \texttt{phase1}(k, c{+}3, g')$
6:          return $(\operatorname{Munion}(\{v \mapsto c\} \cup m',\ f_2),\ g_2)$
7:      else
8:          return $(\emptyset,\ g)$
9:      end if
10: end function
```

Termination is by strict decrease of $|V(g)|$: each step removes v and its neighborhood.

The key feature is that colors increment by 3 at each recursive step: vertex v gets color c, its neighbors get colors $c{+}1$ and $c{+}2$, and the next step starts from $c{+}3$. This ensures complete separation between coloring steps.

Lemma 8.3 (Color Lower Bound)**.** *If* `phase1`(k, c, g) *assigns color* c_i *to vertex* i*, then* $c_i \geq c$*.*

Proof. By induction on $|V(g)|$. In the current step, colors c, c+1, c+2 are used (all $\geq c$). The recursive call starts at c+3, and by the inductive hypothesis, all colors it produces are $\geq c+3 > c$. □

Theorem 8.4 (Phase 1 Coloring OK)**.** *If g is undirected, irreflexive, and admits a complete 3-coloring, then for any edge (i, j) in g, if both i and j are colored by* `phase1`(k, c, g)*, they receive distinct colors.*

Proof. By induction on $|V(g)|$. For an edge (i, j), the coloring of each endpoint comes from either the current step or the recursive call. There are four cases:

1. *Both current step:* If both i and j are colored in the same step, they are either (a) the center v and a neighbor (colors c vs. c+1 or c+2, hence distinct), or (b) both neighbors, handled by the correctness of `force_all` on the bipartite neighborhood (Theorem 7.9 and Theorem 6.6).
2. *Current step and recursive call:* The current step uses colors $\leq c+2$, while the recursive call uses colors $\geq c+3$ (Lemma 8.3). Hence the colors differ.
3. *Recursive call and current step:* Symmetric to case 2.
4. *Both recursive:* The edge (i, j) persists in the reduced graph g' (since neither endpoint was removed), and the 3-coloring restricts to g'. The inductive hypothesis applies.

□

Lemma 8.5 (Phase 1 Structural Properties)**.** *For any undirected graph g:*

1. *The residual graph g_2 from* `phase1`(k, c, g) *is a subgraph of g.*
2. *g_2 is undirected and irreflexive.*
3. *g_2 has no vertex with degree exceeding k.*
4. *Edges between surviving vertices are preserved in g_2.*

8.2 Phase 2: Degree-Bounded Greedy Coloring

Phase 2 operates on the residual graph, which has bounded maximum degree.

Definition 8.6 (Phase 2)**.**

```
1: function PHASE2($g$)
2:     if maxdeg($g$) = 0 then
3:         return (constant_color(nodes($g$), 1), $\emptyset$)
4:     else
5:         $d \leftarrow$ maxdeg($g$)
6:         $(s, g') \leftarrow$ extract_vertices_degs($g, d$)        ▷ independent set of max-degree vertices
7:         $(f', g'') \leftarrow$ phase2($g'$)
8:         return (Munion(constant_color($s, d$+1), $f'$), $g''$)
9:     end if
10: end function
```

At each step, the vertices of maximum degree form an independent set (Theorem 4.14), so they can all receive the same color. The maximum degree strictly decreases after extraction, so the recursion terminates and uses at most maxdeg$(g) + 1$ distinct colors.

Theorem 8.7 (Phase 2 Correctness)**.** *For any undirected irreflexive graph g,* `phase2`(g) *produces a valid coloring.*

Proof. By functional induction on `phase2`. The base case (maxdeg $= 0$) is vacuous—no edges exist. In the step case, the independent set s is colored with a fresh color $d+1$ not used by the recursive coloring (which uses colors corresponding to the smaller maximum degree of g'). The union of an independent-set coloring with a valid coloring, using a fresh color, is valid (proved as lemma `indep_set_union`). □

8.3 The Full Algorithm

Definition 8.8 (Wigderson's Algorithm)**.** Given a degree threshold k and graph g:

```
1: function WIGDERSON($k, g$)
2:     $(f_1, g') \leftarrow$ phase1($k, 1, g$)
3:     offset $\leftarrow$ max_color($f_1$)
4:     $f_2 \leftarrow$ phase2($g'$)
5:     return Munion($f_1$, map $(\lambda c.$ offset $+ c)$ $f_2$)
6: end function
```

The offset ensures phase 2 colors are strictly larger than all phase 1 colors.

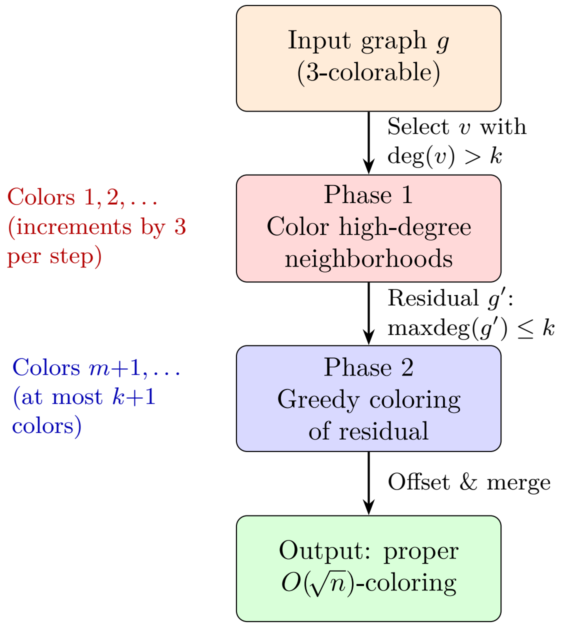


Figure 6: Overview of Wigderson's algorithm. Phase 1 processes high-degree vertices, removing each along with its neighborhood. Phase 2 colors the bounded-degree residual. Color offsets prevent conflicts.

Definition 8.9 (Max Color)**.** `max_color`$(f) =$ `fold` $(\lambda\, k\, v\, a.\ \max(v, a))\ f\ 1$, computing the largest color value in f.

Lemma 8.10. *For any i with $f(i) = c_i$: $c_i \leq$ `max_color`(f).*

Theorem 8.11 (Wigderson Correctness)**.** *Let g be an undirected, irreflexive graph with a complete 3-coloring. Then for any edge (i, j) in g, if* `wigderson`(k, g) *assigns colors c_i and c_j to i and j respectively, then $c_i \neq c_j$.*

Proof. Let f_1 be the phase 1 coloring, f_2' the offset phase 2 coloring, and $w = \text{Munion}(f_1, f_2')$ the combined result. For an edge (i, j), we analyze four cases based on which map provides each color:

1. *Both from f_1:* Apply Theorem 8.4.
2. *i from f_1, j from f_2':* We have $c_i \le \texttt{max\_color}(f_1) = \text{offset}$ by Lemma 8.10, while $c_j = \text{offset} + c_j^0$ for some $c_j^0 \ge 1$, so $c_j > c_i$.
3. *i from f_2', j from f_1:* Symmetric to case 2.
4. *Both from f_2':* Then $c_i = \text{offset} + c_i^0$ and $c_j = \text{offset} + c_j^0$. Since i and j are both in the residual graph g' and the edge persists in g' (Lemma 8.5), Phase 2 assigns $c_i^0 \ne c_j^0$ (Theorem 8.7), hence $c_i \ne c_j$ by injectivity of the offset.

□

8.4 Color Complexity

We now prove that, with an appropriate choice of threshold, the algorithm uses $O(\sqrt{n})$ colors. We begin by bounding the number of colors used by each phase separately.

Lemma 8.12 (Phase 1 Iteration Count)**.** *Let g be an undirected irreflexive graph on n vertices, and let k be the degree threshold. Phase 1 performs at most $\lfloor n/(k+2) \rfloor$ iterations.*

Proof. At each iteration, a vertex v with $\deg(v, g) > k$ is selected. The algorithm removes the set $\{v\} \cup \text{adj}(g, v)$ from the current graph. Since g is irreflexive, $v \notin \text{adj}(g, v)$, so $|\{v\} \cup \text{adj}(g, v)| = 1 + |\text{adj}(g, v)| \ge 1 + (k+1) = k+2$.

Successive iterations remove disjoint subsets of the original vertex set (each iteration removes vertices from the current graph, which excludes all previously removed vertices). If phase 1 performs t iterations, then $t \cdot (k+2) \le n$, giving $t \le \lfloor n/(k+2) \rfloor$. □

Lemma 8.13 (Phase 1 Color Count)**.** *Phase 1 uses at most $3\lfloor n/(k+2) \rfloor$ distinct colors. More precisely, $\texttt{max\_color}(f_1) \le 3\lfloor n/(k+2) \rfloor$.*

Proof. At the i-th iteration ($i = 1, 2, \ldots, t$), the starting color parameter is $c = 1 + 3(i-1)$. The three colors used are c, $c+1$, $c+2$, *i.e.*, $3i-2$, $3i-1$, $3i$. After t iterations the largest color is $3t$. By Lemma 8.12, $t \le \lfloor n/(k+2) \rfloor$, so the maximum color is at most $3\lfloor n/(k+2) \rfloor$. □

Lemma 8.14 (Phase 2 Color Count)**.** *Let g' be the residual graph after phase 1. Phase 2 uses at most* $\text{maxdeg}(g') + 1$ *distinct colors.*

Proof. Phase 2 proceeds by repeatedly extracting the independent set of maximum-degree vertices and assigning them a single fresh color. At each step the maximum degree strictly decreases (by Theorem 4.14 and the exhaustion lemma). Starting from $\text{maxdeg}(g') = d$, the algorithm uses one color for degree d, one for degree $d-1$, and so on down to degree 0 (whose vertices all receive color 1). This gives exactly $d + 1$ colors, numbered $1, 2, \ldots, d+1$.

More precisely, the Coq proof establishes (via `phase2_color_bound`) that every color c assigned by phase 2 satisfies $c \le \text{maxdeg}(g') + 1$ when expressed as a natural number. □

Theorem 8.15 (Color Bound — `wigderson_color_bound`)**.** *Let g be an undirected irreflexive graph on n vertices. With degree threshold $k \ge 0$, every color c_i assigned by $\texttt{wigderson}(k, g)$ satisfies*

$$c_i \;\le\; 3\left\lfloor \frac{n}{k+2} \right\rfloor + k + 2.$$

In particular, setting $k = \lfloor \sqrt{n} \rfloor$ gives $O(\sqrt{n})$ colors.

Proof. Let $(f_1, g') = \texttt{phase1}(k, 1, g)$ and $f_2 = \texttt{phase2}(g')$. Let $m = \texttt{max\_color}(f_1)$. The combined coloring is

$$w = \text{Munion}\big(f_1,\ \text{map}\ (\lambda c.\ m + c)\ f_2\big).$$

For any vertex i with $w(i) = c_i$, we case-split on whether c_i comes from f_1 or the shifted f_2.

Phase 1 colors. We prove inductively (`phase1_color_upper_bound`) that every color c_i assigned by $\texttt{phase1}(k, c, g)$ satisfies $c_i + 1 \le c + 3\lfloor |V(g)|/(k+2) \rfloor$. The induction is on $|V(g)|$ using well-founded induction. In the step case, a high-degree vertex v is selected; the current step uses colors $c, c+1, c+2$, which are all $\le c + 2$. Since v has degree $> k$, removing $\{v\} \cup \text{adj}(g, v)$ decreases the vertex count by at least $k+2$ (`phase1_removes_many`). This yields $\lfloor |V(g')|/(k+2) \rfloor + 1 \le \lfloor |V(g)|/(k+2) \rfloor$ by integer division (`nat_div_step`), allowing the inductive bound on colors from the recursive call to be absorbed.

With $c = 1$: $c_i + 1 \le 1 + 3\lfloor n/(k+2) \rfloor$, so $c_i \le 3\lfloor n/(k+2) \rfloor \le 3\lfloor n/(k+2) \rfloor + k + 2$.

Phase 2 colors. If $c_i = m + c_i^0$ for some phase 2 color c_i^0, we bound each term:

- *Bounding m:* By `max_color_bound_nat` and the phase 1 bound above, $m \le 3\lfloor n/(k+2) \rfloor + 1$. (The $+1$ accounts for the base case where f_1 is empty and $m = 1$.)
- *Bounding c_i^0:* By `phase2_color_bound`, $c_i^0 \le \text{maxdeg}(g') + 1$. By `phase1_residual_max_deg`, the residual graph has $\text{maxdeg}(g') \le k$ (no high-degree vertices remain after phase 1). Hence $c_i^0 \le k + 1$.

Combining: $c_i = m + c_i^0 \le (3\lfloor n/(k+2) \rfloor + 1) + (k + 1) = 3\lfloor n/(k+2) \rfloor + k + 2$.

Asymptotic bound. Setting $k = \lfloor \sqrt{n} \rfloor$:

$$3\left\lfloor \frac{n}{\lfloor \sqrt{n} \rfloor + 2} \right\rfloor + \lfloor \sqrt{n} \rfloor + 2 \ \le\ \frac{3n}{\sqrt{n}} + \sqrt{n} + 2 \ =\ 4\sqrt{n} + 2,$$

which is $O(\sqrt{n})$. □

Remark 8.16. The constant can be improved by optimizing the threshold. Setting $k = \lfloor \sqrt{3n} \rfloor - 2$ minimizes the expression $3n/(k+2) + k + 2$, yielding an upper bound of approximately $2\sqrt{3n} + 2 \approx 3.47\sqrt{n}$ colors. However, the standard statement of Wigderson's result simply asserts $O(\sqrt{n})$, which our proof establishes. The Coq theorem (`wigderson_color_bound`) states this bound directly on each color value, requiring only undirectedness and irreflexivity as preconditions—notably, 3-colorability is not needed for the bound itself, only for correctness.

9 Conclusion

We have presented a complete formal verification of Wigderson's graph coloring algorithm in Coq. The main results are: (1) the algorithm produces a proper coloring of any 3-colorable undirected graph (`wigderson_ok`, Theorem 8.11), and (2) every color assigned is bounded by $3\lfloor n/(k+2) \rfloor + k + 2$, giving $O(\sqrt{n})$ colors (`wigderson_color_bound`, Theorem 8.15).

The formalization consists of approximately 6,600 lines of Coq across 9 source files:

- `graph.v`: Core graph definitions, domain lemmas, and greedy coloring.
- `restrict.v`: Map restriction by key sets.
- `munion.v`: Left-biased map union.
- `subgraph.v`: Induced subgraphs, neighborhoods, degrees, independent sets.
- `coloring.v`: Coloring completeness, n-coloring, union of disjoint colorings.
- `connectivity.v`: Walks, reachability, bipartition parity.
- `bipartite.v`: Bipartition, 2-coloring equivalence.

- `forcing.v`: BFS-based 2-coloring of bipartite graphs.
- `wigderson.v`: Both phases, the main correctness theorem, and the color bound.

The proof makes extensive use of CoqHammer [2] for automation, particularly the `hauto`, `sauto`, and `sfirstorder` tactics, which handle many routine first-order reasoning steps. All proofs are complete—no `Admitted` declarations remain.

A notable design choice is the use of Coq's `Function` command with `{measure M.cardinal g}` for defining recursive functions over graphs, enabling termination proofs based on the strict decrease of the vertex count. The BFS-based 2-coloring (`force_all`) uses well-founded induction on graph cardinality, while `phase1` and `phase2` both use the `Function` mechanism.

Acknowledgments. The graph representation and greedy coloring algorithm are based on the VFA (Verified Functional Algorithms) development by Andrew Appel.

References

[1] A. Wigderson. Improving the performance guarantee for approximate graph coloring. *Journal of the ACM*, 30(4):729–735, 1983.

[2] Ł. Czajka and C. Kaliszyk. Hammer for Coq: Automation for dependent type theory. *Journal of Automated Reasoning*, 61(1–4):423–453, 2018.