CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Regular pattern-free coloring
Authors	Escamocher, Guillaume;O'Sullivan, Barry
Publication date	2022-11-15
Original Citation	Escamocher, G. and O'Sullivan, B. (2022) 'Regular pattern-free coloring', Discrete Applied Mathematics, 321, pp. 109-125. https://doi.org/10.1016/j.dam.2022.06.034
Type of publication	Article (peer-reviewed)
Link to publisher's version	https://www.sciencedirect.com/science/article/pii/S0166218X2200227X - 10.1016/j.dam.2022.06.034
Rights	© 2022 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/). - http://creativecommons.org/licenses/by/4.0/
Download date	2026-11-06 08:33:26
Item downloaded from	https://hdl.handle.net/10468/13338

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Discrete Applied Mathematics 321 (2022) 109–125

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Regular pattern-free coloring

Guillaume Escamocher *, Barry O'Sullivan

Insight Centre for Data Analytics, School of Computer Science & Information Technology, University College Cork, Ireland

ARTICLE INFO

Article history:
Received 18 June 2021
Received in revised form 16 June 2022
Accepted 21 June 2022
Available online xxxx

Keywords:
Graph coloring
Forbidden subgraphs
Regular graphs
Complexity results
Difficult instance generation

ABSTRACT

We study the graph coloring problem under two kinds of simultaneous restrictions. First we forbid some patterns to appear in the graph, where a pattern is a small subgraph. Second we only consider regular graphs, meaning that all nodes have the same degree. Having both types of constraints at once leads us to the discovery of new tractable classes for graph coloring. However, we also show that some classes of pattern-free graphs remain NP-Complete even after enforcing regularity. Based on the latter results, we provide several complementary ways to generate difficult graph coloring instances, relying on balancing the degree of the nodes and avoiding a particular subgraph. Our constructions are parameterizable, so characteristics of the instances like size (number of nodes) and density (number of edges) can be set to any value.

© 2022 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

The k-Coloring problem consists in determining whether the vertices of a given graph can be labeled with k colors, such that vertices which share an edge are labeled with different colors. It was one of the first problems to be proved NP-Complete, figuring among Karp's list of 21 NP-Complete problems [17]. It has many practical applications, including frequency assignment [9] and register allocation [1].

In order to find tractable classes for the problem, restrictions on the graph have been proposed. One such way to define classes of graphs is by forbidding the occurrence of a particular structure, that we call a pattern. Planar graphs fall under this category, because they can be characterized by forbidding subdivisions of the clique of size 5 and of the complete bipartite graph of size 6 (Kuratowski's theorem). Patterns are often small graphs [13], and sometimes are only forbidden as induced subgraphs [18].

A different way to define classes of graphs is through regularity. This property is orthogonal to forbidden patterns because, with the trivial exception of edgeless graphs, regular graphs cannot be characterized by the absence of a forbidden pattern. The k-Coloring problem is still NP-hard when restricted to regular graphs [8], but can become tractable when combined with other structures.

The classes that we study in this paper are hybrid, in that they stand at the intersection of forbidden patterns and regular graphs. We take NP-hard classes of forbidden patterns and examine their complexities when regularity is imposed. Depending on the initial pattern considered, we obtain both polynomial and NP-hardness results. The former provide new tractable classes for the k-Coloring problem, while the latter are used as a foundation to generate difficult coloring instances.

The outline of our paper is as follows. In the next Section, we define the concepts that are central to the k-Coloring problem, including the two core notions (forbidden patterns and regularity) that the classes that we study are based on. In

* Corresponding author.
E-mail addresses: guillaume.escamocher@insight-centre.org (G. Escamocher), b.osullivan@cs.ucc.ie (B. O'Sullivan).

https://doi.org/10.1016/j.dam.2022.06.034
0166-218X/© 2022 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Section 3, we present our complexity results. First, we show that some NP-Complete patterns remain NP-hard when the even stricter restriction of regularity is added. Afterwards we identify some patterns that are NP-Complete by themselves but become tractable when coupled with regularity.

Section 4 is devoted to our generators of difficult coloring instances. We detail the two heuristics that they follow, compare them to other attempts [15,22,26] that have been made in the past to create such instances, and finally present empirical results showing that our generators are successful in building hard coloring instances for various constrainedness values.

2. Definitions

We consider undirected finite graphs with no redundant edges. A *graph* is a couple $\langle V, E\rangle$ where V is a finite set of *vertices* (also called *nodes*) and E is a finite set of *edges*. Each edge $e \in E$ is a pair of vertices $\{v_1, v_2\} \subset V$, and we say that v_2 and e are *incident* to v_1.

We now define the problem studied in this paper.

Definition 1. The *k-Coloring problem* takes as input a graph $G = \langle V, E\rangle$ and asks the following question: can each node in V be labeled with one of k colors, such that for each edge $\{v_1, v_2\} \in E$, v_1 and v_2 are not labeled with the same color?

If for some integer k the answer to the k-Coloring problem is *yes* for some graph G, then we say that G is *k-colorable*. A graph is 1-colorable if and only if it does not contain any edge, and 2-colorable if and only if it is bipartite. For higher values, determining k-colorability is NP-Complete [17]. The graphs returned by our Section 4 generators are not restricted to a specific number of colors and can be given as input to any k-coloring problem, regardless of the value of k. Our experiments in particular cover three different values for k: 3, 4 and 5.

Definition 2. Let G be a graph. The *chromatic number* of G is the smallest k for which G is k-colorable.

Definition 3. The *Coloring problem* takes as input a graph G and asks what the chromatic number of G is.

The k-Coloring problem is a decision problem, because the answer is either *yes* or *no*. In contrast, the Coloring problem is an optimization problem. Their complexity is the same however, because a k-Coloring instance can be solved instantly when knowing its chromatic number, and the Coloring problem can be reduced to $O(\log n)$ (with binary search) iterations of the k-Coloring problem.

The classes that we study in the paper contain graphs that are regular and pattern-free. We now define these two notions, starting with the former.

Definition 4. With the *degree* of a node v being the number of nodes incident to v, we say that a graph G is *regular* if all nodes in G have the same degree. We additionally say that G is d-regular if this degree is equal to d.

The *Regular (k-)Coloring problem* is the (k-)Coloring problem restricted to regular graphs.

If G and P are graphs, then we say that G is P-free if P is not a subgraph of G. The class of graphs associated with a given subgraph, or *pattern*, P is the set of graphs that are P-free.

Definition 5. Let P be a pattern and k be an integer. We say that P is *tractable for the (Regular) k-Coloring problem* if there is a polynomial-time algorithm that can determine for each (regular) P-free graph G whether G is k-colorable. On the other hand, we say that P is *NP-Complete for the (Regular) k-Coloring problem* if determining the k-colorability of (regular) P-free graphs is NP-Complete.

Definition 6. Let P be a pattern. We say that P is *tractable* (respectively *NP-hard*) *for the (Regular) Coloring problem* if determining the chromatic number of (regular) P-free graphs is a polynomial-time problem (respectively is NP-hard).

Note that if P is a pattern and P' is another pattern containing P, then any P-free graph will also be P'-free. Therefore, a tractability result on P' implies the same tractability result on P, and a hardness result on P implies the same hardness result on P'.

3. Theoretical complexity

In this section, we look at how the complexity of different patterns is affected by regularity. Forbidden patterns have been extensively studied in constraint satisfaction problems [2,4,6], including in graph coloring [12]. Moreover, it is known that coloring regular graphs is NP-hard [8], even in the case of line graphs [14,19]. However, very few results exist on the computational complexity of the classes found at the intersection of both types of problems.

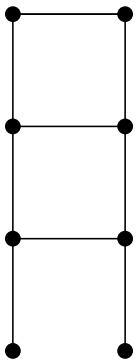


Fig. 1. A triangle-free graph G with maximum degree $d = 3$.

3.1. NP-hardness results

Since tractable patterns for the coloring problem will trivially remain tractable by additionally imposing regularity, we only look at patterns that are already NP-hard for the coloring problem. For all $k \geq 3$, any pattern containing a triangle is NP-Complete for the k-coloring problem [18]. We extend this result to regular graphs.

Theorem 1. *Let P be a pattern containing a triangle and let $k \geq 3$. Then P is NP-Complete for the Regular k-Coloring problem.*

Proof. Let G be a triangle-free graph. Let d be the maximum degree of a node in G. We are going to create a d-regular triangle-free graph G' with the same chromatic number as G.

For each node v in G, we add $d - d_v$ nodes, with d_v the degree of v, and we add an edge between each of these nodes and v. Now each node that was originally in G has degree d, and we also have new nodes, each of degree 1. Let p be the number of these additional nodes, labeled l_1 to l_p, and let $H_{left,1}$ be the graph at this point of the reduction. We know that $H_{left,1}$ is triangle-free, because G is triangle-free and the only edges that have been added have a degree 1 node at one of their extremities, so no cycle has been introduced.

We then duplicate $H_{left,1}$ until we have $2(d-1)$ copies $H_{left,1}, H_{left,2}, \ldots, H_{left,d-1}, H_{right,1}, \ldots, H_{right,d-1}$. For each $1 \leq i \leq p$ and for each $1 \leq j < d$, the copy of the node l_i in $H_{left,j}$ is also labeled l_i, and the copy of l_i in $H_{right,j}$ is labeled r_i. To complete the construction of G', for each $1 \leq i \leq p$ we add $(d-1)^2$ edges, connecting the $d-1$ nodes labeled l_i to the $d-1$ nodes labeled r_i. To illustrate the reduction, we present in Fig. 1 a graph G which is triangle-free but not regular, and in Fig. 2 the graph G' obtained from G after the reduction.

Since triangles are NP-Complete for the k-Coloring problem [18], we only need to show that the reduction takes polynomial time in the size of G, that G' is both d-regular and triangle-free, and that for any $k \geq 3$, G' is k-colorable if and only if G is k-colorable.

Let n be the number of nodes in G. For each node in G, we added at most d nodes in $H_{left,1}$. Since d is the maximum degree of a node in G, the total number of nodes added to G in $H_{left,1}$ is at most $n(n-1)$. Combined with the original n nodes in G, we know that $H_{left,1}$ contains at most n^2 nodes. G' is composed of $2(d-1)$ copies of $H_{left,1}$, so the number of nodes in G' is less than n^3. Therefore the reduction takes polynomial time in the size of G.

In each subgraph $H_{left,i}$ (and $H_{right,i}$), just enough nodes were added to G so that each node of G has degree d in G'. Furthermore, each node labeled l_i (respectively r_i) is connected in G' to exactly 1 node in a copy of G, and to the $d-1$ nodes labeled r_i (respectively l_i). So each node labeled l_i or r_i is also of degree d. So G' is d-regular.

We have shown earlier that the subgraphs $H_{left,i}$ and $H_{right,i}$ are triangle-free. The only other edges in G' connect a node labeled l_j to a node labeled r_j. The nodes labeled l_j are only connected to nodes from the subgraphs $H_{left,i}$ and to nodes labeled r_j. Similarly, the nodes labeled r_j are only connected to nodes from the subgraphs $H_{right,i}$ and to nodes labeled l_j. Therefore, no node is connected to both a node labeled l_j and a node labeled r_j. Therefore the edges between the l_j and the r_j cannot be part of a triangle. Therefore G' is triangle-free.

Let $k \geq 3$. G is a subgraph of G', so if G' is k-colorable, then G is k-colorable. Suppose now that G is k-colorable. Let $c_1, c_2, \ldots, c_k$ be the k colors used to color G. We are going to color G' with these k colors. First, color the subgraph corresponding to G in $H_{left,1}$ in the same way that G was k-colored. Then pick for each l_i in $H_{left,1}$ a color that was not used to color a node connected to that l_i. This is always possible, because each l_i is only connected to one node in G, and there are more than one color. At this point, $H_{left,1}$ has been completely colored. Now for each $1 < i < d$, color the subgraph $H_{left,i}$ in the same way that $H_{left,1}$ was colored. Also color the subgraphs $H_{right,i}$ in the same manner, but substitute c_j for c_{j+1} for each $1 \leq j < k$, and c_k for c_1. This completes the coloring of G'. There cannot be an edge connecting two similarly colored nodes in one of the subgraphs $H_{left,i}$ (respectively $H_{right,i}$), because the graph G was correctly colored, and the nodes labeled l_i (respectively r_i) were colored such that no constraint was falsified. The only other edges in G' connect a

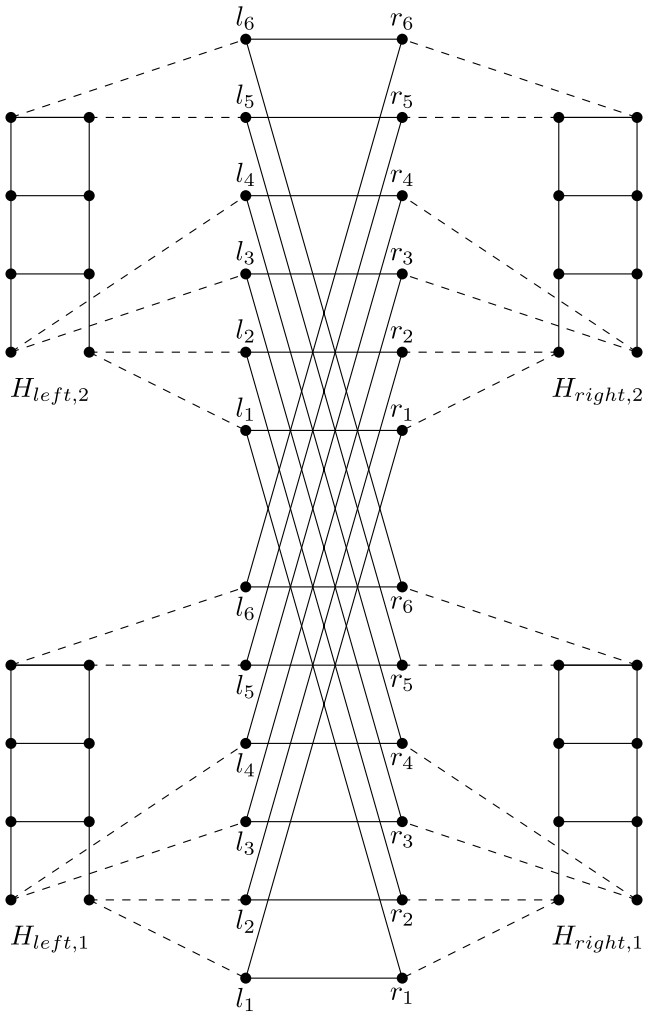


Fig. 2. A 3-regular triangle-free graph G'.

node labeled l_j to a node labeled r_j. Let j be such that $1 \leq j \leq p$, and let c_x be the color picked for the nodes labeled l_i. The same color was chosen for all these nodes, because all the $H_{left,i}$ were colored similarly. Apart from the $H_{left,i}$, which as we have already established do not contain any incompatible coloring, the nodes labeled l_j are only connected to the nodes labeled r_j. Because of the substitution, the nodes labeled r_j have been colored with c_{x+1} (or c_1 if $x = k$). Since $k \geq 3$, we have that any edge between a node labeled l_j and a node labeled r_j connects two differently colored nodes. Therefore the k-coloring of G' is valid. Therefore, if G is k-colorable, then G' is k-colorable. □

Another example of an NP-Complete pattern for the 3-Coloring problem is *Crab*, shown in Fig. 3 [13]. This pattern is a tree, but as with triangles the hardness result can be extended to regular graphs.

Theorem 2. *Let P be a pattern containing Crab. Then P is NP-Complete for the Regular 3-Coloring problem.*

Proof. We reduce from the Regular 3-Coloring problem, which is NP-hard even when restricted to 4-regular graphs [8]. Let G be a 4-regular graph. We replace each occurrence of *Crab* in G by the gadget shown in Fig. 4. The gadget does not contain *Crab*, so we only have to show that the graph G' obtained after the operation is 4-regular, and that it is 3-colorable if and only if G is 3-colorable.

Each node in G' belongs to one of three sets: nodes already present in G, new nodes added by the gadget, and nodes modified by it. Since G is 4-regular, all nodes in the first set still have degree 4 in G'. For each occurrence of the gadget, 5 nodes are added to the second set: v_2, a', b', c' and d'. All of these nodes have degree 4 by construction of the gadget. Lastly, the only node from G that is altered by the gadget is v, which is replaced by the two nodes v_1 and v_3, both of degree 4. Therefore all the nodes in G' have degree 4, making G' 4-regular.

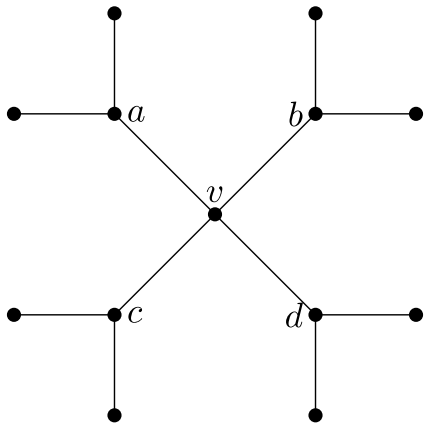


Fig. 3. The pattern *Crab*.

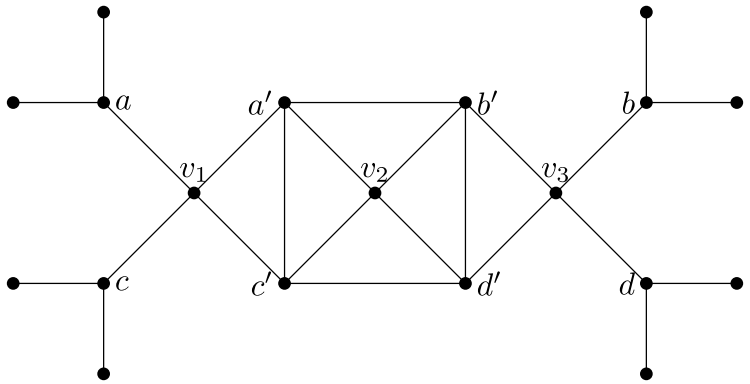


Fig. 4. A *Crab*-free gadget.

Suppose that there is a 3-coloring of G. Assume without loss of generality that in this coloring v is labeled red. Then by labeling v_1, v_2 and v_3 red, a' and d' green, b' and c' blue, and all other vertices of G' by the color assigned to them in G, we have a 3-coloring of G'.

Suppose now that there is a 3-coloring of G'. Notice that if v_1 and v_2 are labeled with different colors, then it is not possible to label the four vertices v_1, v_2, a' and c' with only 3 colors. Therefore v_1 and v_2 have the same label in the 3-coloring of G'. Similarly, v_2 and v_3 also are labeled the same. Assume without loss of generality that the label shared by v_1, v_2 and v_3 is red. Then by labeling v red and all other vertices of G by the color assigned to them in G', we have a 3-coloring of G. □

3.2. Tractable classes

Several tractable classes based on a forbidden pattern have already been identified for the Coloring problem. For example, any forest with at most 7 nodes and no node of degree 5 or more is tractable for the Coloring problem [13]. This proves in particular that the pattern X_2 from Fig. 5 is tractable for the 3-Coloring problem. The authors of the above paper could not determine the tractability of X_3, but they managed to prove the NP-Completeness of X_4 for the 3-Coloring problem.

In this section, we will prove that X_4 is tractable for the Regular 3-Coloring problem. In order to do so, we will need to prove the tractability of X_3 for the Regular 3-Coloring problem.

Lemma 1. *X_3 is tractable for the Regular 3-Coloring problem.*

Proof. Let $G = \langle V, E\rangle$ be a regular graph. Suppose that the pattern X_0 appears in G. Let $V_{out} = V \setminus \{v, a, b, c, d\}$. Let E_{out} be the set of edges connecting a node in $\{a, b, c, d\}$ to a node in V_{out}, and let E_{in} be the set of edges connecting two nodes from $\{a, b, c, d\}$. Since G is regular, all nodes in V have the same degree Δ.

If $|E_{out}| \geq 9$, then at least one of a, b, c and d has three incident edges in E_{out}, at least two of them have at least two incident edges in E_{out}, and at least three of them have at least one incident edge in E_{out}. Assume without loss of generality

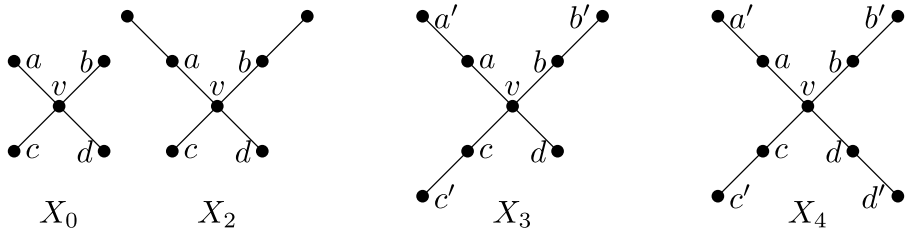


Fig. 5. Some trees.

that a has three incident edges in E_{out}, b has two incident edges in E_{out} and c has an incident edge e_c in E_{out}. At least one of the two edges in E_{out} that are incident to b does not share a node with e_c. Let us call this edge e_b. At least one of the three edges in E_{out} that are incident to a does not share a node with either e_c or e_b. Let us call this edge e_a. Together with X_0, e_a, e_b and e_c form X_3. So we can assume from now on that $|E_{out}| \leq 8$.

Suppose that there is a triangle in E_{in}. Since all nodes in $\{a, b, c, d\}$ are connected to v, we have a clique of size 4 and G is not 3-colorable. So we can assume from now on that there is no triangle in E_{in}, meaning in particular that $|E_{in}| \leq 4$.

Since all nodes in $\{a, b, c, d\}$ have degree Δ and are connected to v, we have $4\Delta = 4 + |E_{out}| + 2|E_{in}|$. Since $|E_{out}| \leq 8$ and $|E_{in}| \leq 4$, we have $\Delta \leq 5$. Since the degree of v is at least 4, either $\Delta = 4$ or $\Delta = 5$.

If $\Delta = 5$, then we have $|E_{out}| = 8$ and $|E_{in}| = 4$. The only way to have four edges but no triangle in E_{in} is if these edges form a cycle of size 4. Since $\Delta = 5$ and each node in $\{a, b, c, d\}$ is incident to v and to exactly two edges in E_{in}, then each node in $\{a, b, c, d\}$ is incident to exactly two edges in E_{out}.

Suppose that all nodes in $\{a, b, c, d\}$ are connected to the same two nodes v_1 and v_2 in V_{out}. Since $\Delta = 5$, v is connected to a node v' not in $\{a, b, c, d\}$. If $v' = v_1$ (respectively $v' = v_2$), then v_1 (respectively v_2), v, a and any node in $\{b, c, d\}$ connected to a form a clique of size 4, so G is not 3-colorable. If $v' \neq v_1$ and $v' \neq v_2$, then the edges (v, a), (a, v_1), (v, b), (b, v_2), (v, c), (c, d) and (v, v') form X_3 (switch the roles of b and d if c is not connected to d).

Suppose on the other hand that at least three of the nodes in V_{out} that are incident to a node in $\{a, b, c, d\}$ are distinct. Let us call f_1, f_2 and f_3 these three nodes. Without loss of generality, assume that f_1 is connected to a. a is incident to only two edges in E_{out}, so either f_2 or f_3 is connected to b, c or d. Assume without loss of generality that f_2 is connected to b. If either c or d is connected to a node in v_{out} that is neither f_1 nor f_2, then we have the pattern X_3. Otherwise, both c and d are connected to both f_1 and f_2, and adding the edges (a, f_3) (or (b, f_3) if f_3 is not connected to a), (c, f_1) and (d, f_2) to X_0 forms X_3.

We have shown that if $\Delta = 5$, then either the pattern X_3 appears in G, or G is not 3-colorable. We can therefore assume from now on that $\Delta = 4$. Since $4\Delta = 4 + |E_{out}| + 2|E_{in}|$, we have $|E_{out}| + 2|E_{in}| = 12$. Since $|E_{out}| \leq 8$ and $|E_{in}| \leq 4$, then the only three possibilities for the values of $|E_{in}|$ and $|E_{out}|$ are $|E_{in}| = 2$ and $|E_{out}| = 8$, $|E_{in}| = 3$ and $|E_{out}| = 6$, and $|E_{in}| = 4$ and $|E_{out}| = 4$.

Let us first look at the case where $|E_{in}| = 4$ and $|E_{out}| = 4$. As mentioned before, the only way to have four edges but no triangle in E_{in} is if these edges form a cycle of size 4. Since $\Delta = 4$, each node in $\{a, b, c, d\}$ is connected to exactly one node in V_{out}. If at least three of these external nodes are distinct, then we have the pattern X_3. If all four of these external nodes are the same node v', then the six nodes v, a, b, c, d and v' form a 4-regular subgraph disconnected from the rest of G. This subgraph is 3-colorable, for example by labeling v and v' red, a and d green, and b and c blue, so we can remove it from G without changing the 3-colorability of G.

If exactly two (let us call them v_1 and v_2) of the four external nodes are distinct, then either they are both connected to exactly two nodes in $\{a, b, c, d\}$, or one of them is connected to three nodes in $\{a, b, c, d\}$ and the other one is connected to the remaining node in $\{a, b, c, d\}$. If the former, assume without loss of generality that v_1 is connected to a and d and that v_2 is connected to b and c, if the latter, assume without loss of generality that v_1 is connected to a and that v_2 is connected to b, c and d. Either way, v_1 is connected to a node v' that is not b nor c (because all edges incident to b and c are already accounted for) and that is not v_2, a nor d either (because $\Delta = 4$). Therefore the edges (a, v_1), (v_1, v'), (a, c), (c, v_2), (a, b), (b, d) and (a, v) form the pattern X_3, which completes the study of the case where $|E_{in}| = 4$ and $|E_{out}| = 4$.

Let us now look at the case where $|E_{in}| = 2$ and $|E_{out}| = 8$. If the two edges in E_{in} share a node, assume without loss of generality that they connect a to b and c. Since $\Delta = 4$, a is connected with a node $a' \in V_{out}$, b is connected to two nodes from V_{out}, at least one of them (let us call it b') not being a', and d is connected to three nodes from V_{out}, at least one of them (let us call it d') being neither a' nor b'. Adding the edges (a, a'), (b, b') and (d, d') to X_0 forms the pattern X_3.

If the two edges in E_{in} do not share a node, then a, b, c and d are all connected to exactly two nodes in V_{out}. If all four are connected to the same two nodes v_1 and v_2, then the seven nodes v, a, b, c, d, v_1 and v_2 form a 4-regular subgraph disconnected from the rest of G. This subgraph is 3-colorable, for example by labeling v, v_1 and v_2 red, a and any node in $\{b, c, d\}$ not connected to a green, and the last two nodes blue, so we can remove it from G without changing the 3-colorability of G. If at least three of the nodes in V_{out} that are incident to a node in $\{a, b, c, d\}$ are distinct, then by the same argument as for $\Delta = 5$, we have the pattern X_3. This concludes the study of the case where $|E_{in}| = 2$ and $|E_{out}| = 8$.

The last case to consider is when $|E_{in}| = 3$ and $|E_{out}| = 6$. The only two ways to have three edges and no triangle in E_{in} are a path of length 3 and a star. If the latter, assume without loss of generality that a is connected to b, c and d. Since $\Delta = 4$, b, c and d are all connected to exactly two nodes in V_{out}. If at least three of the nodes (let us call them v_1, v_2 and v_3) in V_{out} connected to b, c or d are distinct, then we can assume without loss of generality that v_1 is connected to b and that v_2 is connected to c. If one of the two V_{out} nodes connected to d is neither v_1 nor v_2, then we have the pattern X_3. Otherwise, adding the edges (d, v_1), (c, v_3) and (b, v_2) (or (d, v_2), (c, v_1) and (b, v_3) if v_3 is not connected to c) to X_0 forms the pattern X_3.

Still in the subcase where the three edges in E_{in} form a star centered in a, suppose now that b, c and d are connected to the same two nodes in V_{out}: v_1 and v_2. If v_1 and v_2 are connected, then the seven nodes v, a, b, c, d, v_1 and v_2 form a 4-regular subgraph disconnected from the rest of G. This subgraph is 3-colorable, for example by labeling v and v_1 red, a and v_2 green, and b, c and d blue. If v_1 and v_2 are not connected, then there is a node v' in $V_{out} \setminus \{v_2\}$ that is connected to v_1, and the edges (b, v_1), (v_1, v'), (b, v_2), (v_2, c), (b, a), (a, d) and (b, v) form the pattern X_3.

The last subcase to look at is when the edges in E_{in} form a path of length 3. Assume without loss of generality that c is connected to a, which is connected to b, which is connected to d. Since $\Delta = 4$, a and b are each connected to one node in V_{out}, while c and d are both connected to two nodes in V_{out}. If at least three distinct nodes of V_{out} are connected to either c or d, then one of them (let us call it v_1) is connected to c and another (let us call it v_2) is connected to d. If a', the node in V_{out} connected to a, is neither v_1 nor v_2, then adding the edges (c, v_1), (d, v_2) and (a, a') to X_0 forms the pattern X_3. Otherwise, then without loss of generality we can assume that a' is v_1, in which case adding the edges (c, v_3), (d, v_2) and (a, v_1) (or (c, v_2), (d, v_3) and (a, v_1) if v_3 is not connected to c) to X_0 forms the pattern X_3.

Suppose now that c and d are connected to the same two nodes v_1 and v_2 from V_{out}. If either a or b is connected to neither v_1 nor v_2, then we have X_3. If both a and b are connected to v_1, then all of the edges incident to v, a, b, c, d and v_1 are accounted for, so v_2 is connected to some node $v' \in V_{out} \setminus \{v_1\}$, and the edges (d, v_2), (v_2, v'), (d, v_1), (v_1, c), (d, b), (b, a) and (d, v) form the pattern X_3. The same argument, with the roles of v_1 and v_2 switched, applies if both a and b are connected to v_2.

The only remaining possibility is when one of a and b is connected to v_1 and the other is connected to v_2. Assume without loss of generality that a is connected to v_1 and that b is connected to v_2. The edges between v, a, c and v_1 impose that v and v_1 must be labeled with the same color. Similarly, the edges between v, b, d and v_2 impose that v and v_2 must be labeled with the same color. So if v_1 and v_2 are connected to each other, then G is not 3-colorable. Otherwise, v_1 is connected to some node $v' \in V_{out} \setminus \{v_2\}$, and the edges (v_1, a), (a, b), (v_1, c), (c, v), (v_1, d), (d, v_2) and (v_1, v') form the pattern X_3.

We have studied all possibilities for the distribution of the edges from E_{out} and E_{in}. In each case, we have shown that the presence of the pattern X_0 leads to one of three possible outcomes: X_3 appears in G, a disconnected subgraph can be deleted without changing the 3-colorability of G, or G is not 3-colorable. Since X_0 is tractable for the 3-Coloring problem [13], this proves the result. □

Now that we know that X_3 is tractable for the Regular 3-Coloring problem, we can assume that it always appears in the graph, making the proof for X_4 easier.

Theorem 3. *X_4 is tractable for the Regular 3-Coloring problem.*

Proof. Let $G = \langle V, E \rangle$ be a regular graph. Suppose that the pattern X_3 appears in G. Since G is regular, all nodes in V have the same degree Δ. Since v is connected to a, b, c and d, we know that $\Delta \geq 4$.

If $\Delta > 7$, then d is connected to at least one node not in X_3 and we have X_4. If $\Delta = 7$, then either d is connected to at least one node not in X_3 and we have X_4, or d is connected to all other nodes in X_3. If the latter and a is connected to a node a'' not in X_3, then by renaming a' to d' and a'' to a' we have X_4. So a is also connected to all nodes in X_3, and we have a clique of size 4 on the nodes v, a, b and d. Therefore, if $\Delta = 7$ then either X_4 appears in G or G is not 3-colorable. So we can assume from now on that $4 \leq \Delta \leq 6$.

Suppose that $\Delta = 4$ and that the gadget *Link* represented in Fig. 6 appears in G. Notice that all nodes in *Link* apart from w and w' already have four incident edges, so any new node we will introduce cannot be w_1, w_2, w_3, w_4 nor w_5. We know that w' has degree 4, so it is connected to three new nodes w'_1, w'_2 and w'_3.

If w'_1, w'_2 and w'_3 are all connected to each other, then together with w' they form a clique of size 4 and G is not 3-colorable. If there is at most one edge between w'_1, w'_2 and w'_3, then assume without loss of generality that it is between w'_1 and w'_2. In this case, w'_1 is connected to two new nodes, let us call one of them w''_1, w'_2 is connected to two new nodes, so at least one of them (let us call it w''_2) is not w''_1, and w'_3 is connected to three new nodes, so at least one of them (let us call it w''_3) is neither w''_1 nor w''_2. This would form the pattern X_4 over edges (w', w_4), (w_4, w_5), (w', w'_1), (w'_1, w''_1), (w', w'_2), (w'_2, w''_2), (w', w'_3) and (w'_3, w''_3). So there are exactly two edges between the nodes w'_1, w'_2 and w'_3. Assume without loss of generality that they connect w'_1 to w'_2, and w'_2 to w'_3.

Since w'_1 is of degree 4, it is connected to four nodes. Two of them are w' and w'_2, the other two, w'_4 and w'_5 are new. We now have the situation depicted in Fig. 7. Suppose that the last incident edge of w'_2 connects it to a new node w''_2 that is neither w'_4 nor w'_5. We know that w'_3 is connected to two more nodes, so at least one of them (let us call it w''_3) is not w''_2, and edges (w', w_4), (w_4, w_5), (w', w'_1), (w'_1, w'_4) (or (w'_1, w'_5) if w''_3 is w'_4), (w', w'_2), (w'_2, w''_2), (w', w'_3) and (w'_3, w''_3) form X_4. So w'_2 is connected to either w'_4 or w'_5. Assume without loss of generality that w'_2 is connected to w'_5.

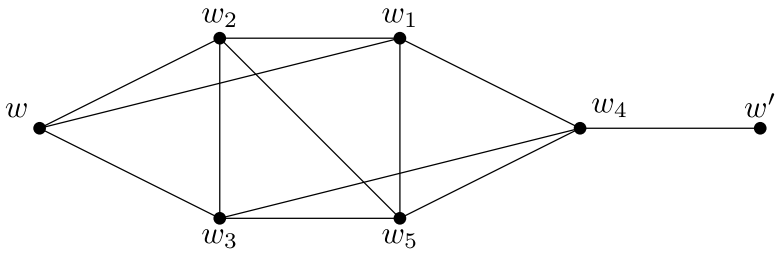


Fig. 6. The gadget *Link*.

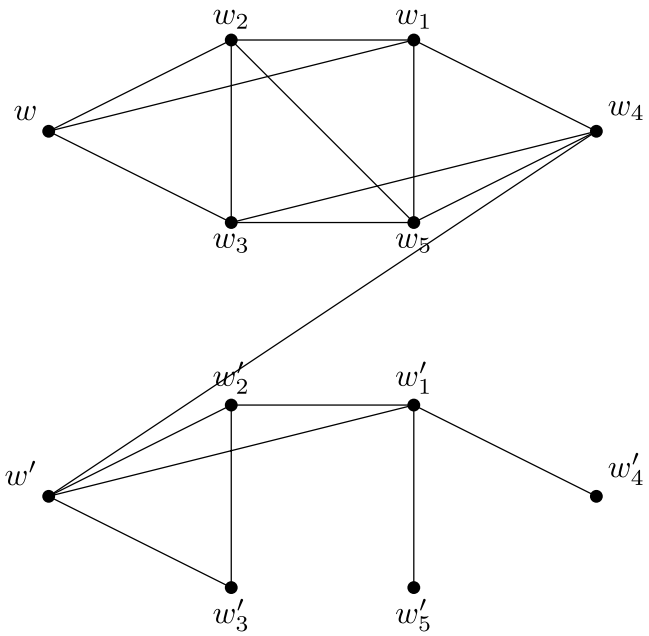


Fig. 7. Extending *Link*.

If w'_3 is connected to a new node w''_3 that is neither w'_4 nor w'_5, then edges (w', w_4), (w_4, w_5), (w', w'_1), (w'_1, w'_4), (w', w'_2), (w'_2, w'_5), (w', w'_3) and (w'_3, w''_3) form X_4. Since w'_3 is connected to two more nodes, it is therefore connected to both w'_4 and w'_5.

We already know that w'_5 is connected to w'_1, w'_2 and w'_3. Let w''_5 be the fourth node incident to w'_5. There are two more nodes incident to w'_4, so at least one of them (let us call it w'') is not w''_5. If w''_5 is not w'_4, then edges (w'_1, w'), (w', w_4), (w'_1, w'_2), (w'_2, w'_3), (w'_1, w'_4), (w'_4, w''), (w'_1, w'_5) and (w'_5, w''_5) form X_4. So w'_5 is connected to w'_4 and we have a copy of the gadget *Link*.

So if $\Delta = 4$, then *Link* can only be connected to a copy of itself. Since G is finite, then an occurrence of *Link* in G implies the existence of a closed chain of copies of *Link*. The gadget is 3-colorable, in fact we can even have a coloring that labels w and w' with the same color: w, w' and w_5 red, w_2 and w_4 green, and w_1 and w_3 blue. So every time X_4 appears in G, we have a 3-colorable chain of gadgets disconnected from the rest of G that we can remove.

To prove the theorem, we show using a computer proof (available in the supplementary material) that if X_3 appears in G then one of the following is true:

- G contains a disconnected component with at most 60 nodes that can be solved separately and then removed from G.
- G contains a clique of size 4, and therefore is not 3-colorable.
- G contains X_4.
- $\Delta = 4$ and *Link* appears in G.

Since we have already shown that X_3 is tractable for the Regular 3-Coloring problem, this concludes the proof. □

We summarize how our results advance the state of the art in Tables 1 and 2. Our two new NP-hard classes are both strict subsets of two distinct NP-hard classes from the literature, while our main tractability result, in addition of

Table 1
Previously known complexity results for graph coloring.

	Class	Complexity	Reference
KR1	Regular graphs	NP-hard	[8,14,19]
KR2	Triangle-free graphs	NP-hard	[18]
KR3	*Crab*-free graphs (Fig. 3)	NP-hard	[13]
KR4	X_4-free graphs (Fig. 5)	NP-hard	[13]
KR5	Regular X_2-free graphs (Fig. 5)	Tractable	[13]

Table 2
Our new complexity results for graph coloring.

Class	Complexity	Type of result
Regular Triangle-free graphs	NP-hard (Theorem 1)	Smaller NP-hard class than both KR1 and KR2.
Regular *Crab*-free graphs	NP-hard (Theorem 2)	Smaller NP-hard class than both KR1 and KR2.
Regular X_4-free graphs	Tractable (Theorem 3)	Larger tractable class than KR5. in the complexity frontier between the smaller tractable KR5 and the larger NP-hard KR4.

generalizing a known tractable class, illustrates an example of an NP-Complete pattern that becomes tractable when requiring all nodes to have the same degree.

4. Generating difficult coloring instances

In this section, we use the insight obtained from our first NP-Completeness result to provide four related ways to create graph coloring instances that are hard to solve. All our generators are fully parameterizable with regard to the numbers of nodes, edges, and colors.

4.1. Previous generators

Other attempts have been made in the past to create difficult Coloring instances. The problem was used to help in computing derivative matrices [15]. The authors comment that some of the graphs obtained during the process form relatively hard coloring instances. However, because their structure is so closely linked to the matrices, it is not possible to finely tune parameters like number of nodes and number of edges, unlike with our own generators.

Hard Coloring instances have also been built by combining minimal unsolvable graphs [22]. When using the same solvers that we use in our experiments, their instances are actually more difficult to solve than ours, but because their generator relies heavily on a small number of gadgets, it suffers from three major drawbacks compared to ours. First, all of the unsolvable graphs are composed of a fixed number of edges, with each one containing between 1.70 and 1.85 times as many edges as nodes, so the number of edges in the instances that they obtain cannot be parameterized. Second, knowledge of their algorithm, and in particular of the list of seven unsolvable patterns, can be exploited to make their instances easy to solve, for example by looking for occurrences of these gadgets in the instance. In comparison, because of Theorem 1, knowing that our algorithms rely on regular and/or triangle-free graphs cannot help much in solving the instances that they create. Third, their focus is entirely on the 3-Coloring problem, while our instances can be given as input of a k-Coloring problem for any number of colors.

Avoiding three-edge patterns to find difficult Coloring instances has been tried before [26], although once again only for the $k = 3$ case. Instead of triangles, the author aims to eliminate paths of length 3. Note that 3-Coloring is in P when such paths are forbidden [13], so there exists an algorithm that can solve these instances in polynomial time. This is not the case for our instances (unless P=NP), because our generators are based on an NP-Completeness class.

4.2. Our generators

One of our goals when designing our generators was to make them parameterizable, meaning that they could create graphs of any size (number of nodes n) and density (number of edges m), and not just graphs that follow some specific structure. To this end, we followed principles adopted by Satisfiability (SAT) generators that have achieved this goal. For example, Balanced SAT [25] can output Conjunctive Normal Form (CNF) instances for any desired combination of values for the number of variables, number of clauses, and arity.

Balanced SAT manages to generate difficult CNF instances, but not all Balanced SAT instances are hard to solve. For a fixed number of variables, increasing the number of clauses leads eventually to a sharp transition from a phase where most instances are satisfiable and solving them is easy to a phase where most instances are so over-constrained that

proving unsatisfiability is easy. Additionally, instances around the threshold are often found very hard to solve. This phase transition behavior is common to all decision problems [3], including SAT [21] and the k-Coloring problem [7].

The Balanced SAT algorithm has been subsequently modified, resulting in No-Triangle CNF, a CNF instance generator that can provide difficult CNF instances for parameter values corresponding to trivially under- or over-constrained Balanced SAT instances [11]. Note that neither of these two generators can be considered intrinsically better than the other, because the hardest instances built by each are comparable in terms of difficulty, they are instead complementary in terms of the constrainedness regions that they cover.

When deciding on which variable to add to a clause, Balanced SAT follows one main heuristic: balancing the number of occurrences of each variable. The most intuitive graph equivalent would be to balance the degree of the nodes when adding an edge. No-Triangle CNF follows the same heuristic as Balanced SAT, as well as a second one: minimizing the number of constraint triangles introduced in the CNF instance. This corresponds to minimizing the number of triangles (cliques of size 3) introduced in the graph being generated. Depending on which heuristic is used and, in the case where both are, on the order that they are applied, we obtain four different graph generators.

For a given number of nodes n and a given number of edges m, all four generators build a k-Coloring instance by starting from a graph with n nodes and no edges, then adding m edges one by one following one or both of the two heuristics. These heuristics aim to get as close as possible to their respective associated property, but the resulting graph is not guaranteed to always be regular and/or triangle-free. For example, depending on the divisibility of the number of edges by the number of nodes, it might not be actually possible to attain regularity. However, as we will show in Section 4.3, in practice the most difficult instances either do reach these ideals or end up close.

We call **RegGraph** the generator that picks the edge connecting the two nodes with the lowest combined sum of degrees, with random tie-breaking. This method aims solely for regular graphs, and can be considered as the graph equivalent of Balanced SAT. As we will show in Section 4.4, RegGraph can build difficult k-Coloring instances when the desired number of edges is low.

We call **TFGraph** the generator that picks the edge that introduces the fewest triangles to the graph, with random tie-breaking. This method aims solely for triangle-free graphs. While TFGraph instances do not appear to be as difficult to solve on average as RegGraph ones, their phase transition occurs at a different place, and therefore TFGraph is not dominated by RegGraph.

We call **RegTFGraph** the generator that picks the edge connecting the two nodes with the lowest combined sum of degrees, with ties broken by minimizing the number of triangles introduced to the graph. This method aims first for regular graphs, and second for triangle-free graphs. It can be considered as the graph equivalent of No-Triangle CNF. As the experiments will show, the locations of hard RegGraph and RegTFGraph instances coincide for $k = 3$, but drift further and further apart as the number of colors increases, making the two generators complementary.

We call **TFRegGraph** the generator that picks the edge that introduces the fewest triangles to the graph, with ties broken by prioritizing pairs of nodes with lowest combined sum of degrees. This method switches the order of RegTFGraph heuristics, aiming first for triangle-free graphs, and second for regular graphs. This generator behaves in a similar way as RegTFGraph, except when using one particular solver, in which case TFRegGraph instances appear to be consistently harder to solve than RegTFGraph ones.

We designed our generators around our triangle-free result, but not around our *Crab*-free one, because counting the number of triangles in a graph can be done in an efficient manner, unlike counting the number of occurrences of *Crab*. Indeed, the former can be naively done in n^3, and in Lemma 2 we show that when exploiting the natural symmetry found in triangles, counting them only affects the total runtime of the algorithm by a multiplicative factor of n. On the other hand, the pattern *Crab* contains 13 vertices, so keeping track of how many times it appears is not feasible in practice in a generator of even moderately sized instances.

Lemma 2. *RegGraph can build a coloring instance with n nodes and m edges in $O(m \times n^2)$ time. TFGraph, RegTFGraph and TFRegGraph can all build a coloring instance with n nodes and m edges in $O(m \times n^3)$ time.*

Proof. For each generator, the data structure used to store the graph is a two-dimensional boolean array g with n^2 cells, such that $g[i, j] \Leftrightarrow$ edge (v_i, v_j) is in the graph. This allows us to add an edge, or to check that two given nodes are connected, in $O(1)$ time, and to list all neighbors of a given node in $O(n)$.

Depending on the generator, we also have up to two additional integer arrays *ScReg* and *ScTF* that store the score of each graph element for the desired metric. For regularity, the score $ScReg[i]$ of node v_i is its degree. The regularity score of an edge (v_i, v_j) can then be computed in $O(1)$ time by simply summing $ScReg[i]$ and $ScReg[j]$. For triangle-freeness, the score $ScTF[i, j]$ of edge (v_i, v_j) is the number of triangles that would be completed if (v_i, v_j) was added to the graph, or equivalently the number of nodes in the graph that are already connected to both v_i and v_j.

The time complexity of building an instance is the number of edges m times the time complexity of adding an edge. The latter is in turn the time complexity of looking for the empty edge (v_i, v_j) with the best score ($O(n^2)$ by iterating through all n^2 edges) times the time complexity of updating the scores. For RegGraph, this is done by incrementing the degrees of v_i and v_j in *ScReg* by 1, giving a total runtime of $O(m \times n^2)$ for this generator. For the other three algorithms, we can increment $ScTF[i, k]$ (respectively $ScTF[k, j]$) by 1 for all k such that v_k is connected to v_j (respectively v_i). As mentioned before, listing all nodes incident to an edge can be done in $O(n)$, so the total time complexity of the three generators that are concerned with triangle-freeness is $O(m \times n^3)$. □

Table 3
Difference between the highest and lowest degrees at the peak of difficulty (m edges) for instances with n nodes when solved by Color6.

Colors	Instances			Difference		
	n	Generator	m	Minimum	Median	Maximum
3	225	RegGraph	580	1	1	1
		TFGraph	530	9	11	14
		RegTFGraph	590	1	1	1
		TFRegGraph	590	1	1	1
	250	RegGraph	650	1	1	1
		TFGraph	600	9	11	14
		RegTFGraph	650	1	1	1
		TFRegGraph	650	1	1	1
	275	RegGraph	710	1	1	1
		TFGraph	660	9	12	14
		RegTFGraph	720	1	1	1
		TFRegGraph	720	1	1	1
4	130	RegGraph	580	1	1	1
		TFGraph	610	10	12	15
		RegTFGraph	620	1	1	1
		TFRegGraph	620	1	1	1
	150	RegGraph	670	1	1	1
		TFGraph	700	11	13	17
		RegTFGraph	710	1	1	1
		TFRegGraph	710	1	1	1
	170	RegGraph	770	1	1	1
		TFGraph	790	10	13	17
		RegTFGraph	800	1	1	1
		TFRegGraph	810	1	1	1
5	90	RegGraph	580	1	1	1
		TFGraph	780	6	8	10
		RegTFGraph	710	1	1	1
		TFRegGraph	750	3	4	6
	100	RegGraph	650	0	0	2
		TFGraph	810	6	8	11
		RegTFGraph	770	1	1	1
		TFRegGraph	790	1	1	2
	110	RegGraph	720	1	1	1
		TFGraph	850	7	10	14
		RegTFGraph	830	1	1	1
		TFRegGraph	850	1	1	1

The last three generators aim to create difficult instances by, as much as possible, forbidding triangles. It has been shown that in the related Constraint Satisfaction Problem, doing the opposite, that is forbidding incomplete incompatibility triangles instead of complete ones, leads to a tractable set of instances [5]. Both their results and ours intuitively complement each other.

Additionally, we call RandGraph the control generator that follows no heuristic and simply adds edges randomly. The graphs obtained by this method are generally much easier to solve than the ones from more elaborate constructions, because their imbalance can be exploited to greatly simplify the instance, in some cases even contain small patterns that are trivially unsolvable, like for example cliques of size $k + 1$.

4.3. Efficiency of the heuristics

While our generators seek to balance the degree of the nodes and/or minimize the number of triangles, they are not guaranteed to return graphs that are exactly regular and triangle-free. To gauge how close they are of this ideal, we present for instances at each peak of difficulty the difference between the highest and lowest degrees of their nodes (Table 3) and their number of triangles (Table 4).

A regular graph has a difference of 0 between the highest and lowest degree. If the number of edges is not a multiple of half the number of variables, then the smallest possible difference becomes 1. As shown in Table 3, this is achieved by hard instances created by RegGraph and RegTFGraph, the two generators that prioritize regularity. For the most part, this minimal difference is also reached by difficult instances created by TFRegGraph, even though this generator favors avoiding triangles over balancing the degree of the nodes. For comparison purposes, we included data for TFGraph, the generator without a regularity heuristic. It shows that when not trying to balance degrees at all, there is a wide difference between the degrees of the most and least connected nodes.

Table 4
Number of triangles at the peak of difficulty (m edges) for instances with n nodes when solved by Color6.

Colors	Instances			#triangles		
	n	Generator	m	Minimum	Median	Maximum
3	225	RegGraph	580	5	11	20
		TFGraph	530	0	0	0
		RegTFGraph	590	0	0	1
		TFRegGraph	590	0	0	0
	250	RegGraph	650	4	13	20
		TFGraph	600	0	0	0
		RegTFGraph	650	0	0	1
		TFRegGraph	650	0	0	0
	275	RegGraph	710	2	12	19
		TFGraph	660	0	0	0
		RegTFGraph	720	0	0	1
		TFRegGraph	720	0	0	0
4	130	RegGraph	580	65	82	104
		TFGraph	610	0	0	0
		RegTFGraph	620	0	2	5
		TFRegGraph	620	0	0	0
	150	RegGraph	670	72	82	107
		TFGraph	700	0	0	0
		RegTFGraph	710	0	2	4
		TFRegGraph	710	0	0	0
	170	RegGraph	770	69	88	105
		TFGraph	790	0	0	0
		RegTFGraph	800	0	1	4
		TFRegGraph	810	0	0	0
5	90	RegGraph	580	251	284	320
		TFGraph	780	0	0	16
		RegTFGraph	710	17	29	38
		TFRegGraph	750	0	0	5
	100	RegGraph	650	262	295	325
		TFGraph	810	0	0	0
		RegTFGraph	770	11	23	33
		TFRegGraph	790	0	0	0
	110	RegGraph	720	264	304	330
		TFGraph	850	0	0	0
		RegTFGraph	830	10	18	27
		TFRegGraph	850	0	0	0

Table 4 shows that the second heuristic used by our generators, minimizing the number of triangles, is just as successful. Hard instances created by our two generators that specifically seek to avoid triangles, TFGraph and TFRegGraph, have a median number of triangles of 0 for all configurations. In fact, for eight out of nine configurations (the exception being small 5-Coloring instances), not a single TFGraph or TFRegGraph instance at the peak of difficulty has even one triangle.

Even when avoiding triangles is second to aiming for regularity, as in our RegTFGraph algorithm, the number of triangles in difficult instances is still close to none for 3- and 4-Coloring. A few triangles do appear in 5-Coloring RegTFGraph instances, but they become rarer as the size increases, even though more edges are added.

We again included the generator that does not follow the relevant heuristic, this time RegGraph, for reference. Unsurprisingly, hard RegGraph instances contain many more triangles than hard instances from our other generators. It is interesting to note that the number of triangles in 3-Coloring RegGraph instances is relatively low, and increases with the number of colors. This would seem to indicate that 3-Coloring RegGraph instances are still similar in some aspects to 3-Coloring instances built by the other algorithms, but become more distinct when the number of colors is increased.

4.4. Comparison of the generators

To study the relative difficulty of instances produced by the generators presented in Section 4.2, we used each of them to create k-Coloring instances for three different numbers of colors: 3, 4 and 5. To generate RandGraph instances, which are equivalent to random graphs, we used the Erdős–Rényi model [10], which assigns to each graph with n nodes and m edges the same probability of being selected.

For $k = 3$, we generated instances with 225, 250 and 275 nodes. For each number of nodes n, we generated instances with m edges, with m varying from $2n$ to $4n$ with an increment of 10. For $k = 4$, we generated instances with 130, 150

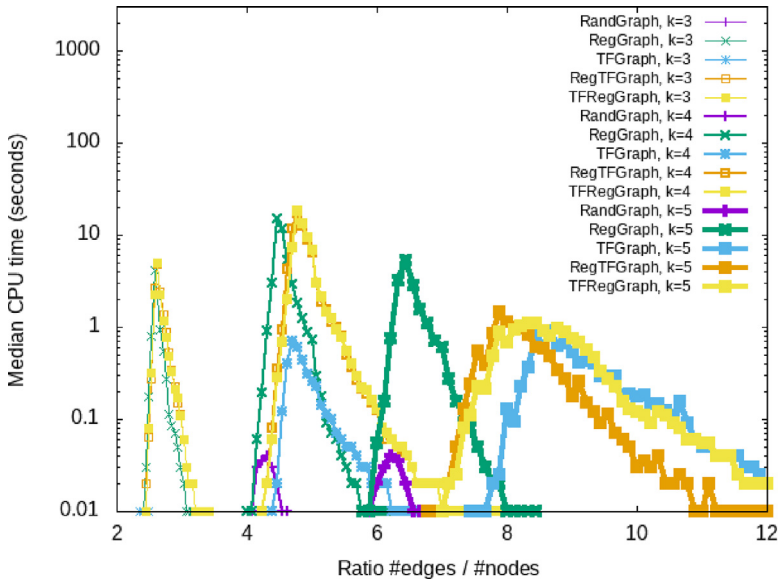


Fig. 8. Comparison of the median runtimes for 3-Coloring instances with 225 nodes, 4-Coloring instances with 170 nodes and 5-Coloring instances with 90 nodes.

and 170 nodes, with the number of edges varying from $3n$ to $8n$, with the same increment of 10. For $k = 5$, the numbers of nodes are 90, 100 and 110, and the number of edges varies from $5n$ to $12n$, again with an increment of 10. In the graph coloring problem, the number of edges of an instance with a fixed number of nodes represents its constrainedness. Adding edges decreases the number of solutions (or at least cannot increase it), and removing edges increases it (or at least cannot decrease it). For each combination of the number of colors and the number of nodes, the range of values for the number of edges was chosen so that it captures the phase transitions for all five generators.

For each combination of k, n and m appearing in our experiments, we generated 250 instances, 50 for each method. To solve the instances, we used the Color6 [27] and Smallk [7] coloring solvers, the GNU Linear Programming Kit (GLPK) 5.0 [20] and CPLEX 22.1 [16] linear programming solvers, as well as the Gecode [24] constraint solver. For this last one, we converted the coloring instances to the standard Minizinc [23] constraint model. Because of the high difficulty of our instances, and the prohibitively long runtimes this leads to, not all five solvers were able to be run on all instance sizes. GLPK, CPLEX and Gecode, probably because they are general-purpose solvers and not specialized coloring ones, could only solve the smallest sizes for each number of colors. CPLEX performed the best of these three, followed by Gecode. Smallk managed to solve higher sizes, but was still unable to solve the largest size ($n = 110$, 170 and 275) for each color. Color6 turned out to be the best performing of the five and was run on all instances. Apart from performance, the results were similar for all solvers (with one minor exception concerning Smallk and two of the generators, that we precise below), both in terms of relative difficulty of the generator outputs, and in terms of the locations of the different peaks of difficulty. For this reason, we only show the results for the best performing solver, Color6.

Experiments were done on a Dell PowerEdge R410 with an Intel Xeon E5620 processor. We present the median runtimes for small instances in Fig. 8, for medium instance in Fig. 9, and for large instances in Fig. 10. All plots in Figs. 8, 9 and 10 have the ratio $\frac{m}{n}$ range from 2 to 12. This allows us to examine how the locations of the peaks of difficulty move as the number of colors increases. We can see that for all generation methods, hard instances are more constrained when there are more colors.

The results show that by combining the two generators RegGraph and TFRegGraph we can obtain difficult coloring instances for a large and diverse (in terms of constrainedness) set of graphs:

- RegGraph generates difficult instances when the number of edges is relatively low.
- TFRegGraph generates difficult instances with more edges than the difficult RegGraph instances. For k=3, difficult instances built by these two algorithms have about the same constrainedness, although TFRegGraph ones are slightly harder to solve, especially in the over-constrained region.
- The behavior of RegTFGraph is close to TFRegGraph, and one could almost substitute the former for the latter. However, one particular solver (Smallk) struggles more with TFRegGraph instances than with RegTFGraph ones. Also, TFRegGraph is slightly better at finding difficult instances for highly constrained regions of the 5-Coloring problem.
- For most constrainedness values, there is always at least one generator that creates instances that are more difficult to solve than TFGraph ones. There are some exceptions (see 5-Coloring instances with high constrainedness) but even they are fairly easy to solve anyway.

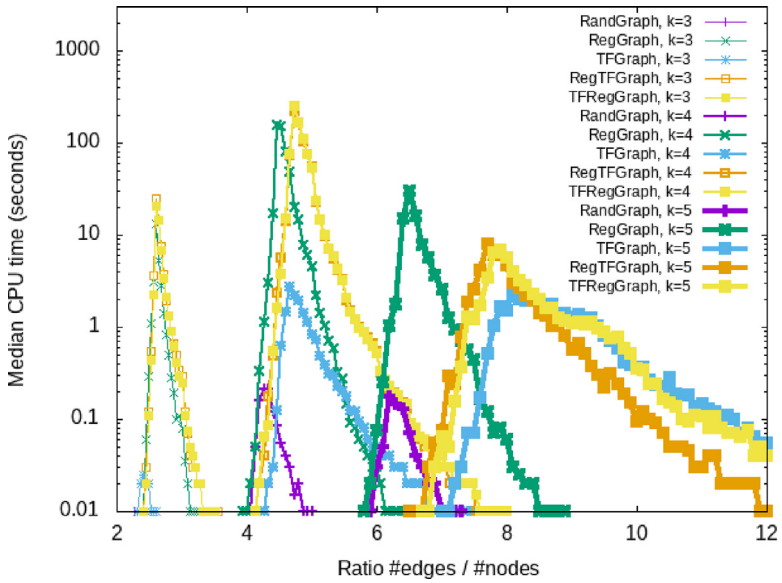


Fig. 9. Comparison of the median runtimes for 3-Coloring instances with 250 nodes, 4-Coloring instances with 150 nodes and 5-Coloring instances with 100 nodes.

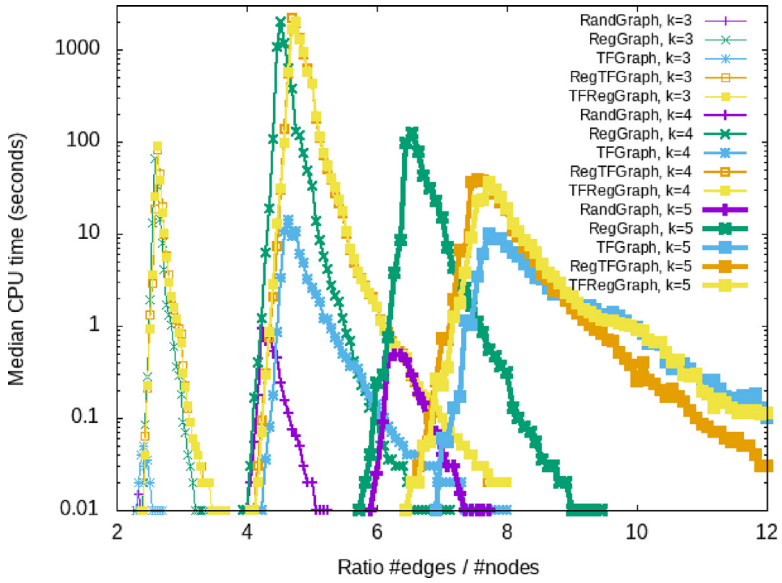
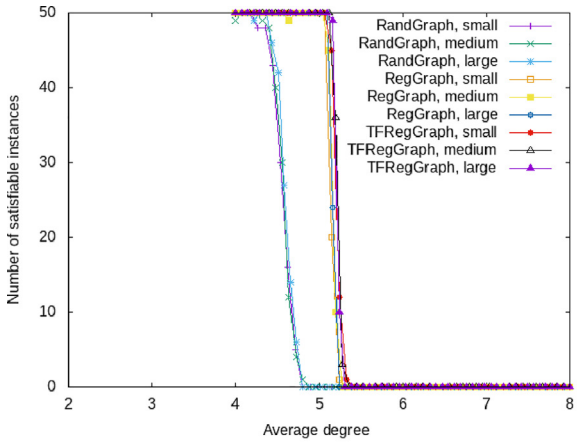


Fig. 10. Comparison of the median runtimes for 3-Coloring instances with 275 nodes, 4-Coloring instances with 130 nodes and 5-Coloring instances with 110 nodes.

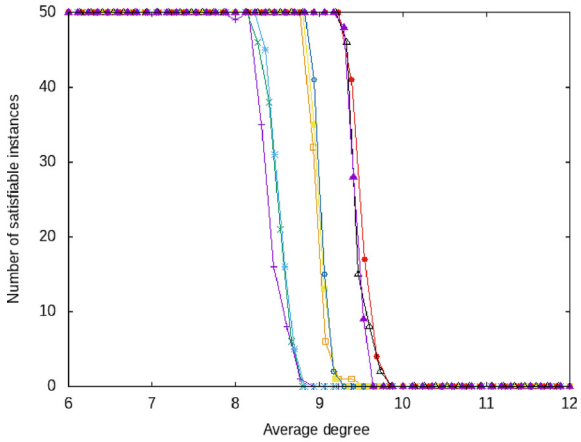
4.5. Coverage of the constrainedness map

For the combination of RegGraph and TFRegGrap to generate difficult instances for as many constrainedness values as possible, the phase transitions of the two generators need to occur at clearly distinct places. Indeed, the hardest to solve instances can be found around the phase transition [3,7], so phase transitions in different locations allow for the construction of difficult instances in different regions of the constrainedness map.

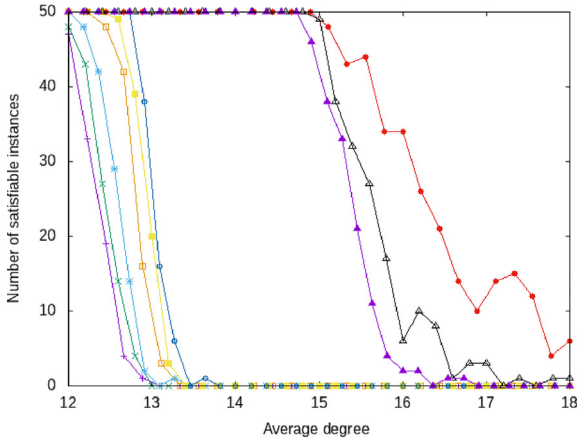
We present in Fig. 11 the relative positions of the phase transitions of RegGraph and TFRegGraph, as well as of RandGraph for perspective, for all graph sizes and number of colors. The X axis represents the average degree of the graphs, which is directly correlated to the constrainedness, because it is equal to twice the number of edges divided by the number of nodes. The Y axis represents the number of satisfiable instances, out of 50 generated for each configuration.



(a) $k = 3$.



(b) $k = 4$.



(c) $k = 5$.

Fig. 11. Locations of the phase transitions for k colors.

Figs. 11(b) and 11(c) uses the same legend as Fig. 11(a), where "small" means the smallest of the three sizes for each number of colors, "large" means the largest of the three sizes, and "medium" means the size in-between.

Note that the location of a phase transition is solver independent, because it is defined by satisfiability and not by difficulty. While the phase transitions occur at a higher average degree when increasing the number of allowed colors, the results show empirically that for a fixed generator and number of colors the phase transitions are found at the same constrainedness value, even when the sizes of the graphs change.

For $k = 3$, the phase transitions of RegGraph and TFRegGraph coincide, but when more colors are allowed they progressively drift apart. This means that no matter which solver is used, for $k > 3$ TFRegGraph can generate difficult instances for numbers of edges where RegGraph cannot (and vice versa).

5. Conclusion

We have determined the complexity of several hybrid graph coloring classes based on forbidden patterns and regularity. These results help to narrow down the possible location of the border between tractability and NP-hardness for regular pattern-free coloring. This separation can be found somewhere between X_4, which is tractable, and *Crab*, which is NP-Complete. Future work could thus focus on studying the complexity of the remaining patterns in the gap, the three graphs that strictly contain X_4 and are strict subgraphs of *Crab*.

We also have introduced a reliable method to produce difficult Coloring instances, based on preserving regularity and triangle-freeness in graphs. Our algorithms can build k-Coloring instances with n nodes and m edges for any combination of k, n and m. Our claim that they are able to create difficult instances has been backed with extensive empirical experiments.

Acknowledgments

This material is based upon works supported by the Science Foundation Ireland under Grant No. 12/RC/2289-P2 which is co-funded under the European Regional Development Fund. For the purpose of Open Access, the authors have applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.dam.2022.06.034.

References

[1] Preston Briggs, Keith D. Cooper, Linda Torczon, Improvements to graph coloring register allocation, ACM Trans. Program. Lang. Syst. 16 (3) (1994) 428–455.
[2] Clément Carbonnel, David A. Cohen, Martin C. Cooper, Stanislav Zivný, On singleton arc consistency for CSPs defined by monotone patterns, Algorithmica 81 (4) (2019) 1699–1727.
[3] Peter C. Cheeseman, Bob Kanefsky, William M. Taylor, Where the really hard problems are, in: John Mylopoulos, Raymond Reiter (Eds.), Proceedings of the 12th International Joint Conference on Artificial Intelligence. Sydney, Australia, August 24-30, 1991, Morgan Kaufmann, 1991, pp. 331–340.
[4] Martin C. Cooper, Guillaume Escamocher, Characterising the complexity of constraint satisfaction problems defined by 2-constraint forbidden patterns, Discrete Appl. Math. 184 (2015) 89–113.
[5] Martin C. Cooper, Stanislav Zivny, Tractable triangles, in: Jimmy Ho-Man Lee (Ed.), Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings, in: Lecture Notes in Computer Science, vol. 6876, Springer, 2011, pp. 195–209.
[6] Martin C. Cooper, Stanislav Zivný, The power of arc consistency for CSPs defined by partially-ordered forbidden patterns, Log. Methods Comput. Sci. 13 (4) (2017).
[7] Joseph C. Culberson, Ian P. Gent, Frozen development in graph coloring, Theoret. Comput. Sci. 265 (1–2) (2001) 227–264.
[8] David P. Dailey, Uniqueness of colorability and colorability of planar 4-regular graphs are NP-complete, Discret. Math. 30 (3) (1980) 289–293.
[9] Andreas Eisenblätter, Martin Grötschel, Arie M.C.A. Koster, Frequency planning and ramifications of coloring, Discuss. Math. Graph Theory 22 (1) (2002) 51–88.
[10] Paul Erdős, Alfréd Rényi, On random graphs I. Math, Debrecen 6 (1959) 290–297.
[11] Guillaume Escamocher, Barry O'Sullivan, Steven David Prestwich, Generating difficult CNF instances in Unexplored Constrainedness Regions, ACM J. Exp. Algorithmics 25 (1) (2020).
[12] Petr A. Golovach, Matthew Johnson, Daniël Paulusma, Jian Song, A survey on the computational complexity of coloring graphs with forbidden subgraphs, J. Graph Theory 84 (4) (2017) 331–363.
[13] Petr A. Golovach, Daniël Paulusma, Bernard Ries, Coloring graphs characterized by a forbidden subgraph, Discrete Appl. Math. 180 (2015) 101–110.
[14] Ian Holyer, The NP-completeness of edge-coloring, SIAM J. Comput. 10 (4) (1981) 718–720.
[15] Shahadat Hossain, Trond Steihaug, Graph coloring in the estimation of sparse derivative matrices: Instances and applications, Discrete Appl. Math. 156 (2) (2008) 280–288.
[16] IBM, CPLEX optimizer, 2022, https://www.ibm.com/analytics/cplex-optimizer.
[17] Richard M. Karp, Reducibility among combinatorial problems, in: Raymond E. Miller, James W. Thatcher (Eds.), Proceedings of a Symposium on the Complexity of Computer Computations, Held March 20-22, 1972, At the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA, in: The IBM Research Symposia Series, Plenum Press, New York, 1972, pp. 85–103.
[18] Daniel Král, Jan Kratochvíl, Zsolt Tuza, Gerhard J. Woeginger, Complexity of coloring graphs without forbidden induced subgraphs, in: Andreas Brandstädt, Van Bang Le (Eds.), Graph-Theoretic Concepts in Computer Science, 27th International Workshop, WG 2001, Boltenhagen, Germany, June 14-16, 2001, Proceedings, in: Lecture Notes in Computer Science, vol. 2204, Springer, 2001, pp. 254–262.
[19] Daniel Leven, Zvi Galil, NP completeness of finding the chromatic index of regular graphs, J. Algorithms 4 (1) (1983) 35–44.

[20] Andrew Makhorin, GNU linear programming kit, 2022, https://www.gnu.org/software/glpk/.
[21] David G. Mitchell, Bart Selman, Hector J. Levesque, Hard and easy distributions of SAT problems, in: William R. Swartout (Ed.), Proceedings of the 10th National Conference on Artificial Intelligence, San Jose, CA, USA, July 12-16, 1992, AAAI Press / The MIT Press, 1992, pp. 459–465.
[22] Kazunori Mizuno, Seiichi Nishihara, Constructive generation of very hard 3-colorability instances, Discrete Appl. Math. 156 (2) (2008) 218–229.
[23] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, Guido Tack, Minizinc: Towards a standard CP modelling language, in: Christian Bessiere (Ed.), Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings, in: Lecture Notes in Computer Science, vol. 4741, Springer, 2007, pp. 529–543.
[24] Christian Schulte, Guido Tack, View-based propagator derivation, Constraints Int. J. 18 (1) (2013) 75–107.
[25] Ivor Spence, Balanced random SAT benchmarks, in: Tomáš Balyo, Marijn Heule, Matti Järvisalo (Eds.), Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions, in: Department of Computer Science Series of Publications B, vol. B-2017-1, University of Helsinki, 2017, pp. 53–54.
[26] Romulus Dan Vlasie, Systematic generation of very hard cases for graph 3-colorability, in: Seventh International Conference on Tools with Artificial Intelligence, ICTAI '95, Herndon, VA, USA, November 5-8, 1995, IEEE Computer Society, 1995, pp. 114–119.
[27] Zhaoyang Zhou, Chu Min Li, Chong Huang, Ruchu Xu, An exact algorithm with learning for the graph coloring problem, Comput. Oper. Res. 51 (2014) 282–301.