Comparative Design-Choice Analysis of Color Refinement Algorithms Beyond the Worst Case

Markus Anders
TU Kaiserslautern, Germany
TU Darmstadt, Germany

Pascal Schweitzer
TU Kaiserslautern, Germany
TU Darmstadt, Germany

Florian Wetzels
TU Kaiserslautern, Germany

Abstract

Color refinement is a crucial subroutine in symmetry detection in theory as well as practice. It has further applications in machine learning and in computational problems from linear algebra.

While tight lower bounds for the worst case complexity are known [Berkholz, Bonsma, Grohe, ESA2013] no comparative analysis of design choices for color refinement algorithms is available.

We devise two models within which we can compare color refinement algorithms using formal methods, an online model and an approximation model. We use these to show that no online algorithm is competitive beyond a logarithmic factor and no algorithm can approximate the optimal color refinement splitting scheme beyond a logarithmic factor.

We also directly compare strategies used in practice showing that, on some graphs, queue based strategies outperform stack based ones by a logarithmic factor and vice versa. Similar results hold for strategies based on priority queues.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Online algorithms

Keywords and phrases Color refinement, Online algorithms, Graph isomorphism, Lower bounds

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.15

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2103.10244`

Funding Received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (EngageS: grant agreement No. 820148).

1 Introduction

Color refinement, also known as 1-dimensional Weisfeiler-Leman algorithm, is a crucial cornerstone of symmetry detection in theory as well as practice. It emerged as a subroutine for algorithms solving the graph isomorphism problem and its efficiency remains to date one of the determining factors for the running time of practical isomorphism solvers. Modern, highly efficient implementations are based on Hopcroft's algorithm for automata minimization [8], which was first adapted to color refinement by McKay in his widely used tool NAUTY [11]. A more recent but also in the meantime large application area of color refinement can be found in machine learning. Specifically, color refinement is used in the Weisfeiler-Leman Kernel for graph classifications as a measure for similarity [15] and as the foundation of graph neural networks [13]. The algorithm can also be applied to effectively reduce the size of linear equation systems [7].

Given a graph, color refinement iteratively recolors the vertices producing increasingly fine partitions of vertices into color classes. Starting with an initial, usually monochromatic coloring, in each iteration the colors of the vertices are chosen to depend on the colors of the neighbors and their multiplicities. If vertices differ in the number of neighbors they have in some color class, the algorithm *splits* up the vertices accordingly by assigning them distinct colors. This is done exhaustively until no further splits are possible.

The applications mentioned above depend on highly engineered implementations of the algorithm. This is the reason why modern implementations meticulously optimize the color refinement subroutine treating many special cases with tailored code [1, 9, 12]. Especially in machine learning applications it is crucial to achieve scalability for big data inputs [15]. Overall, demand for fast implementations of color refinement is high. Since color refinement has a quasilinear worst case running time, even small logarithmic or constant factors can have a crucial impact.

Indeed, the best known implementation of color refinement runs in time $\mathcal{O}(m \log(n))$ (see [4, 10]). Remarkably, within a model with modest assumptions, a tight lower bound construction matching this upper bound was given in 2015 [4]. This result tells us that there are graphs for which color refinement, no matter how it is implemented, runs in $\Omega(m \log(n))$. However, the result does not make any comparative statements between various ways to implement color refinement. In fact, there are dramatic differences in the various implementations of color refinement. While all color refinement algorithms depend on performing the aforementioned splits, there is a lot of freedom as to which order we perform the splits in. A *worklist* is usually employed to determine in what order these splits are performed. Common choices include a stack, queue, priority queue or combinations of these.

So far however, there has been no rigorous analysis as to whether one worklist choice is superior over another – or how significant the order of splits actually is. Going one step further, a natural question is whether there are efficient optimal solutions. If not the case, maybe there are at least solutions that are competitive with all other methods.

Contribution. This paper performs an in-depth comparative analysis of design choices for color refinement algorithms. The first challenge is to actually find a model within which we can compare color refinement algorithms with formal methods. We employ a two-pronged approach. We distinguish (1) algorithms that may only use information realistically collected during the color refinement process itself, and (2) algorithms that are allowed to compute additional information about the underlying graph. Remarkably, our results in the two orthogonal models concur in their conclusion. Namely, that there is no design choice that is competitive beyond a logarithmic factor.

More specifically, in (1) we model algorithms that may only access information explored during the color refinement process itself. For this we define a formal online model within which, in fact, all practical algorithms operate. In this model, the algorithmic decisions of when to refine with respect to what may solely depend on this information. We prove that this information does not suffice to make optimal or even competitive choices, no matter the amount of computational power used. Specifically, we show no online algorithm is within a logarithmic factor of the offline optimum. We also investigate the direct relationship between practical (online) color refinement strategies. Each of the strategies stack, queue, and priority queue, is outperformed by another of the strategies by a logarithmic factor on some graphs.

For (2), we define an "offline" version of the problem, which is essentially to compute an optimal split order for a given graph. Through a reduction from the set cover problem we prove an approximation hardness result. Specifically, unless $\mathsf{P} = \mathsf{NP}$, no approximation factor

Algorithm 1 A typical rendition of color refinement.

```
1  function ColorRefinement($G$, $\pi$)
       Input  : graph $G$, coloring $\pi$
       Output : refined coloring $\pi$
2      initialize empty worklist $W$;
3      put all cells of $\pi$ into $W$;
4      while $W$ is non-empty do
5          take a cell $C$ from $W$;
6          for each cell $X$ containing a neighbor of a vertex in $C$ do
7              for each vertex in $X$ count its neighbors in $C$ ;
8              split $X$ into $X_1, \dots, X_k$ in $\pi$, according to neighbor counts;
9              let $X_i$ be one of the largest cells of $X_1, \dots, X_k$;
10             put all sets $X_1, \dots, X_k$ except $X_i$ into $W$;
11             if $X \in W$ then replace $X$ in $W$ with $X_i$;
12     return $\pi$
```

in $o(\log(n))$ can be achieved by polynomial-time algorithms. This proves that unless $\mathsf{P} = \mathsf{NP}$, even when collecting more information about the underlying graph than current algorithms actually do, computing a competitive let alone optimal order of splits is intractable.

Overall, our results demonstrate that while the choice of worklist can indeed make a crucial difference, there is no clear optimal color refinement strategy. We conclude that users need to adapt color refinement algorithms to the specific type of graphs encountered in the algorithmic application area in mind.

2 Color Refinement

All graphs in this paper are finite, simple, undirected graphs, unless stated otherwise. The neighborhood of a vertex v is denoted $N(v)$. For a set of vertices $V' \subseteq V(G)$ the *neighborhood* is the set $N[V'] \coloneqq (\cup_{v \in V'} N(v)) \setminus V'$. A *coloring* of a graph G is a map $\pi \colon V(G) \to \mathcal{C}$ from the vertices to some set of colors. A (color) *class* is a set $\pi^{-1}(c)$ of vertices of the same color.

We begin with a discussion of the color refinement algorithm itself. Algorithm 1 describes a typical rendition of color refinement. The basic idea is as follows. If two vertices in some class X have a different number of neighbors in some class C then X can be split by partitioning it according to neighbor counts in C. Whenever we split up a class X according to its connections to another class C in such a fashion (see Line 7 and Line 8) we say that we *refine* X *with respect to* C. Specifically this means that after the split, two vertices have the same color precisely if they had the same color before the split and they have the same number of neighbors in C. We repeatedly split classes with respect to other classes until no further splits are possible. A partition not admitting further splits is called *equitable*.

Algorithm 1 maintains the classes with respect to which refinements still have to be performed in a worklist W. Note that the algorithm does not fully specify the internals of the worklist. Specifically, it does not state in Line 5 which cell is extracted from the worklist next. We should emphasize that the final partition into color classes is independent of the choices of cells that are extracted, however the overall running time may depend on it. Typical implementations use a stack, queue, priority queue or a similar data structure. All of these choices result in the same worst case running time of $\Theta((n+m)(\log n))$ (see [4]). To achieve this running time it is crucial to prevent one largest cell (Line 9) from being added to the worklist. Splits with respect to this class are already covered by the other classes.

Overall, the main design choice of the algorithm is the choice of when to split which class with respect to which other class. To describe a general framework for the possible strategies of what to split when, we first need to understand what information is available to the algorithm for making its decision.

2.1 Partial Quotient Graphs

For an equitable partition, quotient graphs capture the information of how many neighbors vertices from one class have in another class. They are used in individualization refinement algorithms as pruning invariants (see [12]). Typically, the quotient graph is computed on the fly during the execution of a color refinement algorithm.

We now introduce the concept of *partial quotient graphs*. These graphs are a tool to formalize the information gathered up to a certain point during the execution of color refinement algorithms. As we cannot precisely say which information an algorithm collects, the quotient graphs give an overapproximation of the available information and model all information that could have possibly been gathered. For the purpose of our lower bounds, overapproximating can only strengthen the conclusions.

The partial quotient graph of a colored graph (G, π) is denoted by $P(G, \pi)$. Quotient graphs are directed and contain self-loops. They include vertex labels l_V as well as edge labels l_E. The vertex set of $P(G, \pi)$ is the set of all sets of colors of (G, π), i.e., $V(P(G, \pi)) := 2^{\pi(V(G))}$. A set of colors also represents the union of the respective color classes.

Vertices of the partial quotient graph are labeled with the size of their corresponding set of vertices in G, i.e., for all sets of colors $c \in 2^{\pi(V(G))}$ we define $l_V(P(G, \pi))(c) := |\pi^{-1}(c)|$, where by $\pi^{-1}(c)$ we denote the vertices whose color is in c. The edge set contains all connections between (unions of) color classes that would not cause a split. Thus there is an edge from c_1 to c_2 if $\pi^{-1}(c_2)$ does not split $\pi^{-1}(c_1)$. Formally, this means

$$E(P(G, \pi)) := \{(c_1, c_2) \mid c_1, c_2 \in 2^{\pi(V(G))}, \forall v, w \in \pi^{-1}(c_1) : \ d_{\pi^{-1}(c_2)}(v) = d_{\pi^{-1}(c_2)}(w)\}.$$

Edges only exist whenever the connection between unions of color classes are regular on one side, so we can label each edge with the corresponding degree, i.e., $l_E(P(G, \pi))((c_1, c_2)) := d_{\pi^{-1}(c_2)}(v)$, where $v \in \pi^{-1}(c_1)$ is arbitrary.

Let us justify the definition with an example. Suppose we split in a monochromatic graph the class of all vertices with itself. Then the new coloring partitions the vertices precisely by degree. That is, classes contain vertices of the same degree. An algorithm would know this degree, since it has counted the edges incident with each vertex, but it would not know how many neighbors a vertex has within a current color class. In the partial quotient graph, there is an edge from each new color classes to the union of all color classes.

The definition of partial quotient graphs contains many more vertices and edges and information on these than would truly be available while executing color refinement. In fact, partial quotient graphs grow exponentially in size, since all possible unions of color classes are considered. Common color refinement algorithms clearly gather much less information. Firstly, only connections of classes that are involved in a refinement are actually considered. Secondly, only information about unions of colors that occurred as a color class in a previous step of the refinement is known. Thus, usually color refinement algorithms only uncover a small, polynomial-sized portion of the partial quotient graphs defined above.

However, for our lower bounds, we assume that algorithms have access to the entire partial quotient graphs. We show that even if we generously allow such access, the information is not sufficient to derive a strategy with constant competitive ratio. For upper bounds, we only use information of the aforementioned polynomial-sized portion of partial quotient graphs. In fact, the upper bounds are based on a stack-based approach akin to Algorithm 1.

Algorithm 2 Corresponding color refinement for a strategy W.

```
1 function ColorRefinement($G$, $\pi$)
      Input  : graph $G$, coloring $\pi$
      Output : refined coloring $\pi$
2     create list $S$ containing $P(G, \pi)$;
3     while $\pi$ is not equitable do
4         $(C, X) := W(S)$;
5         for each vertex in $X$ count its neighbors in $C$;
6         split $X$ into $X_1, \dots, X_k$ in $\pi$, according to neighbor counts;
7         append $P(G, \pi)$ to $S$;
8     return $\pi$
```

2.2 Online Model

We now define a model that bases the choice of which color classes to use for the next refinement solely on the information available through partial quotient graphs. Practical implementations such as a queue or a stack are naturally captured by this, but the model even allows for much more powerful choices. The goal is then to prove that no strategy based solely on information of partial quotient graphs is sufficient to make optimal choices.

Let us start by defining the concept of a *strategy* $W : \mathcal{P}^* \to (2^{\mathbb{N}})^2$. A strategy is a function mapping a sequence of quotient graphs $P = P_1 \cdots P_k \in \mathcal{P}^*$ to two vertices of the last quotient graph $(C, X) \in V(P_k)^2$, that is, two unions of color classes. The sequence of graphs P denotes all partial quotient graphs observed during execution of the algorithm up to step k. The pair (C, X) denotes the choice of colors with which the algorithm continues in the next step: in step $k + 1$, the algorithm refines X with respect to C.

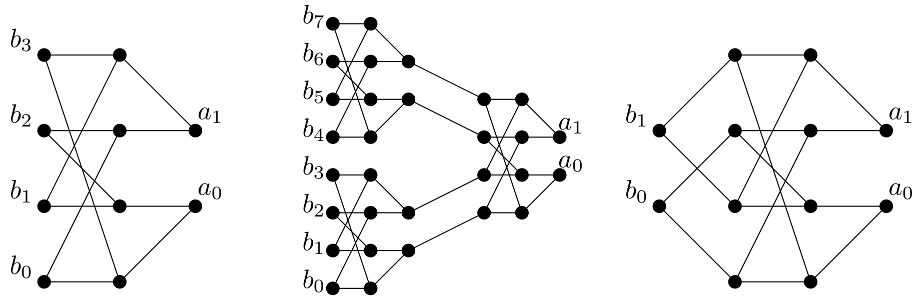
For a strategy W we now define a *corresponding color refinement implementation.* Assume we are working on G and have already refined up to a coloring π_k within k steps. Furthermore, let $P_1, \dots, P_k$ denote the partial quotient graphs corresponding to the execution. Next, we compute $(C, X) = W(P_1 \cdots P_k)$ and refine X with respect to C. The algorithm terminates whenever π_k is equitable. A formal definition is given in Algorithm 2. We call W a *valid* strategy if the corresponding color refinement implementation is correct, i.e., if it terminates with an equitable partition in finite time on all finite graphs.

Throughout this paper, we measure the *cost* of the strategy W, denoted $\mathrm{cost}(W, G)$, in terms of the number of edges that need to be considered to execute the refinements. Specifically, when refining X with respect to C, we charge the algorithm the number of edges connecting X with C. This is the same model as used in [4] reflecting the actual running time of practical implementations (see [11, 12]). We use the terms cost and time interchangeably.

3 Graph Gadgets

Throughout the paper we construct graphs that cause color refinement to behave in particular manners. These graphs are mostly built using three types of graph gadgets, described next.

And gadgets. Let us first discuss the AND_i gadgets as used by Berkholz et al. [4]. There is set B of 2^i in-vertices that come in pairs and 2 out-vertices. The goal of the gadget is that whenever all pairs of *in-vertices* have been split, a split of two *out-vertices* a_0 and a_1 is induced, but not before.



(a) An AND$_2$ gadget. **(b)** An AND$_3$ gadget. **(c)** The unidirectional gadget.

Figure 1 Basic gadget constructions as used throughout the paper. Vertices labeled with b_i always denote in-vertices, while a_i denotes out-vertices.

The AND$_2$ gadget (see Figure 1) is the well known CFI-gadget [5], where two gates form the in-vertices B and the third one the out-vertices a_0, a_1.

The AND$_i$ gadget is constructed recursively using AND$_2$ gadgets. For $i > 2$, the AND$_i$ gadget is constructed by taking the union of two AND$_{i-1}$ and one AND$_2$ gadget. The four out-vertices of the AND$_{i-1}$ gadgets are then connected to the four in-vertices of the AND$_2$ gadget. Figure 1 shows how the AND$_3$ gadget can be constructed using three AND$_2$ gadgets.

The important property is that in an AND$_i$ gadget, all pairs b_{2j}, b_{2j+1} with $j \in \{0, ..., 2^{i-1}\}$ need to be distinguished to induce a split of a_0 and a_1. We should also record a property for the opposite direction: if a_0 and a_1 are distinguished, no split on B should be induced.

Unidirectional gadgets. We now describe the *undirectional gadget*. As the name suggests, it blocks the continuation of a split of pairs in one direction but allows it in the opposite direction. Figure 1 illustrates the gadget.

The gadget behaves as follows. Consider in-vertices b_0, b_1 and out-vertices a_0 and a_1. Distinguishing b_0 and b_1 should induce a split of a_0 and a_1. However, distinguishing a_0 and a_1 should *not* cause a split of b_0 and b_1. The gadget is obtained through a modification of the AND$_2$ gadget. We use the fact that a split of out-vertices in AND$_2$ does not cause a split of the pairs of in-vertices. Therefore, by connecting the in-vertices to new vertices a_0 and a_1, such that the AND$_2$ gadget is activated by any of the two singletons, we get the desired property.

Interestingly, the unidirectional gadget has also been used as a crucial building block in [3] and [6] to study the complexity of various problems closely related to color refinement.

Concealer gadgets. We conclude our discussion of gadgets with the *concealer gadgets*. Similar to the AND$_i$ gadget, a concealer gadget C_i of level i has 2^i in-vertices B and 2 out-vertices a_0, a_1. Whereas in the AND gadget, *all* input pairs need to be distinguished, the concealer gadget only includes *one* specific pair that causes a split of the out-vertices. We call the pair causing the split of out-vertices the *correct pair*, while all other pairs not causing the split are called *dead end pairs*.

The idea is that the correct pair can not be located easily by color refinement algorithms. Hence, the gadget *conceals* where refinement can be continued.

To achieve this behavior, the gadget consists of 2^{i-1} unidirectional gadgets and the out-vertices a_0, a_1. We modify all but one of the unidirectional gadgets so that the connection of the in-gate agrees with the one of the out-gate. This causes these gadgets to become dead ends – activating any of these gadgets has no effect on the out-vertices. The last, unmodified unidirectional gadget is the only one that can actually split the out-vertices and is therefore the only correct gadget.

The out-vertices of the entire concealer gadget are then connected to the out-vertices of all the unidirectional gadgets so that activating the correct pair causes a split of the out-vertices. Figure 2 shows a concealer gadget C_3.

Since we did not specify which of the pairs is the correct pair, there are several concealer gadgets for each $i \in \mathbb{N}$. Abusing notation we denote all of them by C_i. The concealer gadgets have two crucial properties. First, as long as the correct pair has not been split (and the neighbors of a correct pair have not been split) the partial quotient graphs of two concealer gadgets on the same size are isomorphic. Second, the correct pair can only be split from outside the gadget. We formalize these properties in the following.

Consider two colored concealer gadgets $(C_i, \pi), (C_i', \pi')$ of the same order. Suppose $\{b_s, b_{s+1}\}$ is the correct pair in (C_i, π) and $\{b_t, b_{t+1}\}$ is the correct pair in (C_i', π'). We say the two graphs still *concur* if the colors for the vertices agree (note that the two graphs have the same vertex set) and in both graphs neither the correct pairs nor their neighbors have been split. Specifically, we require that

- the vertex colorings agree, (i.e., $\pi(v) = \pi'(v)$ for every $v \in V(C_i) = V(C_i')$),
- the correct pairs have not been distinguished (i.e., $\pi(b_s) = \pi(b_{s+1})$ and $\pi'(b_t) = \pi'(b_{t+1})$),
- the neighbors of the correct pairs have not been distinguished (i.e., $\pi(v) = \pi(v')$ for all $v, v' \in N_{C_i}(b_s) \cup N_{C_i}(b_{s+1})$ and $\pi(v) = \pi(v')$ for all $v, v' \in N_{C_i'}(b_t) \cup N_{C_i'}(b_{t+1})$).

▶ **Lemma 1.** *Suppose (C_i, π) and (C_i', π') are colored concealer gadgets that concur. Then the graphs have the same partial quotient graphs, i.e., $P(C_i, \pi) = P(C_i', \pi')$.*

Proof. Suppose for a vertex v we want to count the number of neighbors that v has in a union of color classes X. We claim that this number is the same in C_i and C_i'. Indeed, we only need to consider edges incident with v that have one endpoint in $M = \{b_s, b_{s+1}, b_t, b_{t+1}\}$ and one endpoint in $N[M]$ (the neighborhood of M). Let E' be the set of these edges and let E_v' be the set of these edges incident with v.

Note that for each of the four sets $\{b_s, b_{s+1}\}$, $\{b_t, b_{t+1}\}$, $N[\{b_s, b_{s+1}\}]$, and $N[\{b_t, b_{t+1}\}]$ either X contains the set entirely or not at all.

If v is in M then either all edges of E_v' have an endpoint in X or no such edge does.

Likewise if v is in $N[M]$ then either all edges of E_v' have an endpoint in X or no such edge does.

Moreover, in either case, whether all such edges are or no such edge is contained does not depend on whether we consider C_i or C_i'.

This implies that the number of edges counted in the refinement (i.e., those incident with v and having an endpoint in X) is the same in C_i and C_i'. ◀

▶ **Lemma 2.** *For concealer gadgets (C_i, π) and (C_i', π') suppose $\pi = \pi'$ so that*

- *vertices in an input pair that is correct in one of the graphs have the same color and*
- *all vertices that are not in an input pair have the same color.*

Then (C_i, π) and (C_i', π') concur. After an arbitrary sequence of splits to both graphs the resulting graphs still concur and neither correct input pairs nor the out pair are split.

Proof. This follows by induction on the number of steps observing that the functionality of the unidirectional gadget ensures that the output pair is never split, and thus vertices inside correct gadgets are never split. ◀

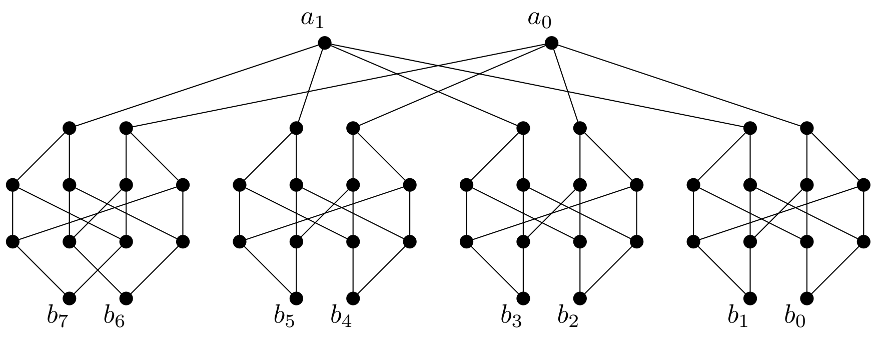


Figure 2 A concealer gadget C_3. Vertices b_6, b_7 form the correct pair; other pairs are dead ends.

The two lemmas show that unless a correct pair is split, the gadgets always concur and an algorithm in the online model will have to perform splits consistently on both graphs. Moreover, the output pair is never split.

Intuitively this means that in the online model, an algorithm can only guess which pair is the correct pair. Therefore, when faced with a concealer gadget, the algorithm potentially has to try all input pairs.

4 Competitive Ratio

We prove the non-existence of a c-competitive strategy in the online model. In particular, in this section, we prove the following theorem:

▶ **Theorem 3.** *For every strategy W of the online model, there is an infinite family of graphs G_k ($k \in \mathbb{N}$) such that $\mathrm{cost}(W, G_k) \in \Omega(\mathrm{opt}(G_k) \cdot \log(\mathrm{opt}(G_k)))$, where $\mathrm{opt}(G_k) \in \Theta(|G_k|)$ is the minimal cost of a strategy on G_k.*

The theorem implies that the information provided by partial quotient graphs is not sufficient to make competitive let alone optimal choices in color refinement algorithms.

Towards this goal, we first define the class of *concealer graphs*, which we denote with $\mathcal{G}_k$ ($k \in \mathbb{N}$). Concealer graphs resemble the graphs of the lower bound construction in [4] closely. Essentially, we swap out AND_i gadgets in the original construction for concealer gadgets C_i. A concealer graph of $\mathcal{G}_4$ is illustrated in Figure 3.

The main idea is that we can then speed-up or slow-down particular strategies by changing the position of the correct pairs within the concealer gadgets. This forces one strategy to extensively search for the correct pairs, while another strategy finds them immediately.

In the rest of this section we provide formal arguments for the above claims. We start with a precise description of concealer graphs. Then, we show that for every concealer graph there exists a fast strategy. Contrarily, we then provide a slow concealer graph for every strategy. Together these two statements prove Theorem 3.

4.1 Concealer Graphs

The first ingredient for the concealer graphs is a "splitting scheme" that results in the worst case running time of $\Omega(m \log(n))$. Consider a vertex set of size $n = 2^k$, on which the following refinements are performed. First, we split the set in halves, then quarters, then eighths and so on, until all vertices have their own distinct color. This gives us $\log(n)$ rounds of refinements, each with a cost of $\Omega(n)$. This results in total costs of $\Omega(n \log(n))$. By ensuring that sufficiently many edges are involved, the running time can be increased to $\Omega(m \log(n))$.

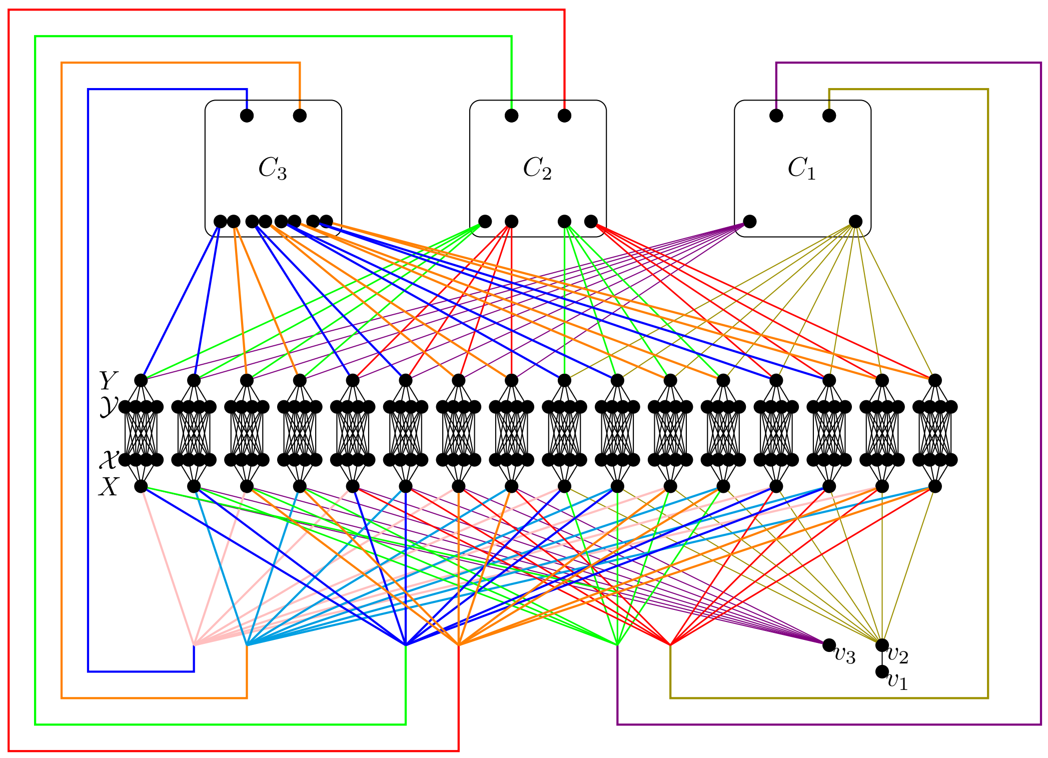


Figure 3 A concealer graph from the class $\mathcal{G}_4$.

Concealer graphs can be used to cause the splitting scheme just described. The graphs contain *middle layers* $(X, \mathcal{X}, \mathcal{Y}, Y)$ (see Figure 3) in which the splitting scheme can be forced. The graph is constructed in a way such that splitting Y into halves, quarters, eighths and so on, causes the next halving refinement on X. The edge colors in Figure 3 indicate the splitting scheme. While the halves (yellow and purple) of Y lead to a split of X into quarters (red and green), the quarters of Y lead to eighths (blue and orange) of X and so on. By initially splitting X in halves, any color refinement algorithm needs to cycle through these layers until X is fully discrete.

The core idea of the general lower bound construction in [4] is that the AND_i gadget enforces refinements with respect to *every* block of level i, which in turn ensures costs of $2^k \cdot k^2 \in \Omega(m)$ for every level.

We modify the construction to suit our purposes as follows. In the concealer graphs, we swap for each i the AND_i gadget for a concealer gadget C_i. On a particular graph, the worst case behavior is therefore not enforced for all refinement strategies anymore. However, a deterministic online algorithm cannot choose for *all* possible concealer gadgets the correct pair in level i to allow it to continue with level $i + 1$. Hence, an adversary can construct a graph that makes a specific color refinement slow, while keeping a “shortcut” for other algorithms that choose the correct pair directly.

We now formally define the class $\mathcal{G}_k$ of concealer graphs. Note that for every $k \in \mathbb{N}$, we define a set of graphs $\mathcal{G}_k$. Essentially, we describe a graph $G_k \in \mathcal{G}_k$ based on concealer gadgets, and the set $\mathcal{G}_k$ then simply consists of all possible instantiations (i.e., positions of the correct pairs) for the included concealer gadgets.

At its core, a graph $G_k \in \mathcal{G}_k$ consists of the four middle layers of vertices $(X, \mathcal{X}, \mathcal{Y}, Y)$, that are interconnected using additional gadgets. Formally, the vertex set of G_k includes $X = \{x_0, ..., x_{2^k-1}\}$, $\mathcal{X} = \{x_i^j \mid 0 \leq i < 2^k, 0 \leq j < k\}$, $\mathcal{Y} = \{y_i^j \mid 0 \leq i < 2^k, 0 \leq j < k\}$,

$Y = \{y_0, ..., y_{2^k-1}\}$, a simple starting gadget induced by only three vertices v_1, v_2, v_3 and $k-1$ concealer gadgets. For $0 \leq l \leq k$ and $0 \leq q \leq 2^l - 1$ let $\mathcal{B}^l_q = \{q2^{k-l}, ..., (q+1)2^{k-l}-1\}$ be the q-th binary block of level l. We use this notation on all sets of size 2^k for some $k \in \mathbb{N}$.

Every x_i is connected to a corresponding y_i via a complete bipartite graph of size k consisting of vertices in $\mathcal{X}$ and $\mathcal{Y}$ (see Figure 3). Formally, each x_i is connected to all x^j_i, y_i to all y^j_i and x^j_i to all $y^{j'}_i$. For each level $l \in \{1, ..., k-1\}$, the i-th binary block of level l is connected to the i-th in-vertex of the l-th concealer gadget. Furthermore, for each gadget C_l, we connect a_0 to all X^l_i with i even and a_1 to all X^l_i with i odd. The starting construction splits X into the blocks X^0_0 and X^0_1. We refer to the i-th in-vertex of the l-th concealer gadget as b^l_i and to the i-th out-vertex as a^l_i.

Let us generally consider how a refinement strategy has to operate on G_k. The algorithm starts with the monochromatic coloring of G_k. The first refinement always distinguishes vertices by their degree, meaning we get the individualized starting gadget $\{v_1\}, \{v_2\}, \{v_3\}$, the distinct layers in the middle $X, \mathcal{X} \cup \mathcal{Y}, Y$, the in- and out-vertices of the concealer gadgets $\bigcup_{l \in \{1,...,k-1\}} \{b^l_i, a^l_j \mid i \in \{0, ..., 2^l\}, j \in \{0, 1\}\}$, and the union of the inner vertices of the concealer gadgets. Next the middle layers are split in half. From this point onwards the splits that are possible depend on finding the correct pair in the gadgets. This can lead to fast or slow refinements, as discussed next.

4.2 A Fast Strategy for Every Concealer Graph

We now show that for every *fixed* concealer graph $G_k \in \mathcal{G}_k$ we can define a linear time strategy. We show this by providing an appropriate sequence of refinements.

For each concealer gadget C_l in G_k, let $b^l_{i_l}, b^l_{i_l+1}$ be the correct pair. Now consider an online refinement strategy on such a graph. After the first (and fixed) refinement, we refine X with respect to $\{v_2\}$ or $\{v_3\}$. We choose one half of $X^1_{i_1}$ for the next refinement and then $\mathcal{X}^1_{i_1}$, $\mathcal{Y}^1_{i_1}$ and $Y^1_{i_1}$ while propagating the split through the middle layers. The important property is that $Y^1_{i_1}$ always splits the correct pair of the next concealer gadget. The concealer gadget then in turn splits X into quarters. Now, we continue with the quarters $X^2_{i_2}$, $\mathcal{X}^2_{i_2}$, $\mathcal{Y}^2_{i_2}$ and $Y^2_{i_2}$, such that the second concealer gadget is activated. This splits X in eighths.

We now repeat this scheme, such that for each level we only propagate the blocks corresponding to correct pairs through the layers and immediately continue with the next level after activating the concealer gadget. When X is discrete, we get the equitable coloring by refining with respect to each level k block of X, $\mathcal{X}$, $\mathcal{Y}$ and Y.

Now consider the cost of this strategy. While cycling through the layers, the most expensive refinements are those with respect to the blocks of $\mathcal{X}$ and $\mathcal{Y}$. On level l, they have cost $2^{k-l} \cdot k^2$, which means the total cost for all levels is $2^k \cdot k^2 = \Theta(m)$. Once X is discrete the cost of the final refinements of $\mathcal{X}$, $\mathcal{Y}$ and Y is also in $\Theta(m)$.

Overall, the cost for an optimal solution for G_k is linear, i.e., $\text{opt}(G_k) \in \Theta(m)$. Note that since refinement is always continued with color classes that have just been created, the scheme actually follows a depth-first approach and can be implemented using a stack.

4.3 A Slow Concealer Graph for Every Strategy

For a *fixed* strategy W, we now provide an infinite family of concealer graphs G_k on which this strategy is slow, i.e., incurs super-linear cost. The family is constructed by choosing for every $k \in \mathbb{N}$ one specific concealer graph $G_k \in \mathcal{G}_k$.

We start with an arbitrary graph $G_k \in \mathcal{G}_k$. We run W on G_k and observe which color classes are split within the concealer gadgets. Say we are looking at concealer gadget C_i. If W distinguishes the correct pair in G_k, but there are still dead ends that have not been distinguished, then we replace G_k by the graph $G'_k \in \mathcal{G}_k$ obtained from G_k by replacing the gadget C_i with another one so that a dead end not yet investigated becomes the correct pair. Due to Lemma 1 and Lemma 2 we know that up until the point where W finds the correct pair in C_i for graph G_k, the strategy W performs the same sequence of splits when executed on G'_k as on G_k. Thus, by doing these transformations exhaustively, we ensure W distinguishes all correct pairs in all the concealer gadgets last. This causes $2^k \cdot k^2$ cost per level and hence $2^k \cdot k^3 = \Theta(m \log(n))$ total cost.

Since the optimal solution for fixed G_k only has linear cost (see Section 4.2), we in turn get that $\mathrm{cost}(W, G_k) \in \Omega(\mathrm{opt}(G_k) \cdot \log(\mathrm{opt}(G_k)))$. This in turn proves Theorem 3. Further details and a more formal reasoning can be found in the full version [2].

5 Comparison of Practical Worklists

We now compare specific, practical worklist data structures. First, we compare stacks with queues. We show that either of the two can asymptotically outperform the other. We can also show the same result for priority queues (see the full version [2]).

5.1 Stack Advantage over Queue

To see how a stack worklist might outperform a queue, recall the fast strategies for concealer graphs of Section 4. The specific, fast split scheme discussed there is realized by a worklist maintained as a stack. Indeed, whenever possible we continue with a "newest" class.

We conclude from Theorem 3 that there is a class of graphs on which a stack based worklist asymptotically outperforms a queue based worklist by a logarithmic factor.

We should remark that it is possible to prove the same result with a simpler construction that does not rely on concealer gadgets. We should also remark that the construction does not apply to all stack based worklists. However, it is possible to modify the construction such that a particular stack based worklist is optimal. For example this can be done for the worklist that choses smallest color classes first (see the full version [2]).

5.2 Queue Advantage over Stack

Now, we construct a graph class, called the *queue graphs*, for which a queue based worklist outperforms a stack based one by a logarithmic factor. This complements the result of the previous section. The construction is also based on the graph class of Berkholz et al. [4]. It is an extension of these graphs, which allows queue worklists to finish quickly but maintains the slow behavior for stacks. We provide an intuitive description. A formal definition and a detailed analysis is given in the full version [2].

The starting gadget. We use a starting gadget (see Figure 4) that forces a stack worklist to perform certain splits before others, while a queue worklist behaves differently.

Consider the gadget together with the coloring indicated in the figure. Any color refinement eventually splits the pairs $\{p_1^{1,3}, p_1^{2,3}\}$, $\{p_2^{1,3}, p_2^{2,3}\}$ and $\{a_0, a_1\}$. However, a stack based worklist splits $\{p_1^{1,3}, p_1^{2,3}\}$ or $\{p_2^{1,3}, p_2^{2,3}\}$ *before* $\{a_0, a_1\}$, while a queue based one splits $\{a_0, a_1\}$ before the other two pairs.

Graph class construction. We start with the graphs from [4] as a main building block. Recall that these graphs are the graphs from Section 4 where the concealer gadgets are replaced by AND gadgets. As argued in [4] a worst case behavior is enforced for every refinement strategy: any refinement on these graphs has a cost of $\Omega(2^k \cdot k^3) = \Omega(m \log(n))$.

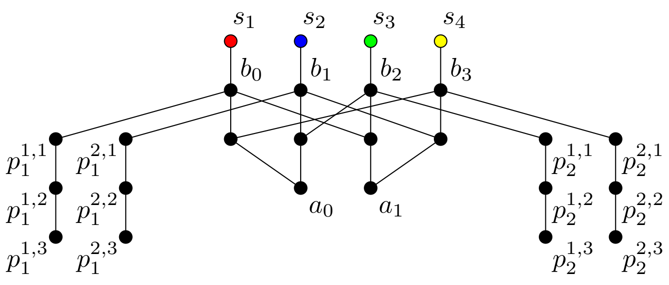


Figure 4 An extension of AND_2 gadget, which will be used as the starting gadget of the new construction. The starting vertices $s_1, ..., s_4$ have been individualized.

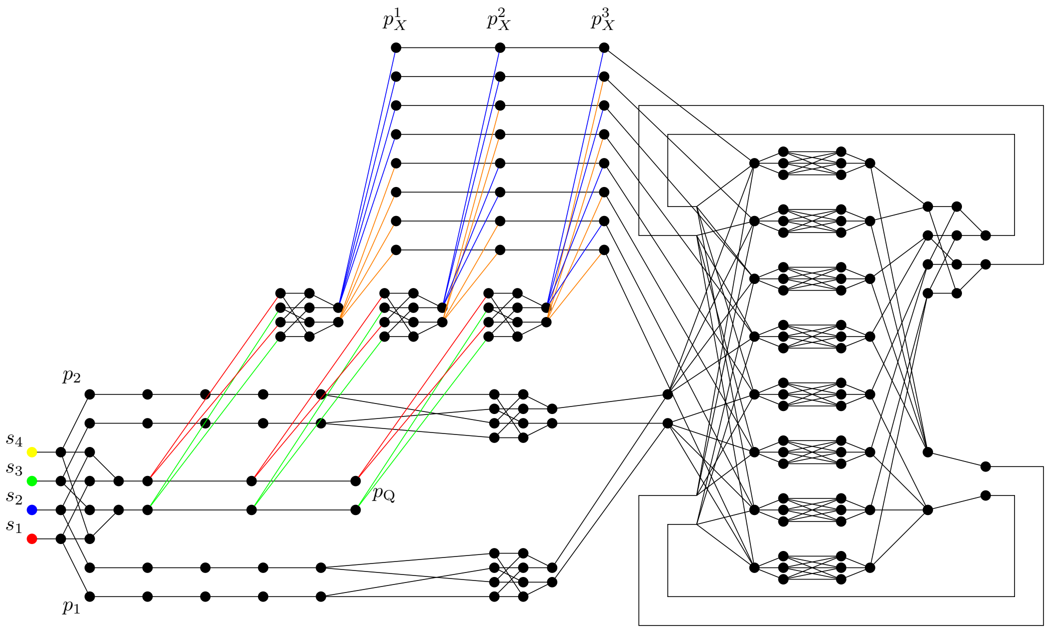
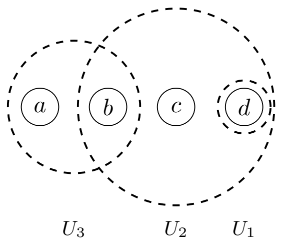


Figure 5 The graph G_3^{Q} for the queue advantage.

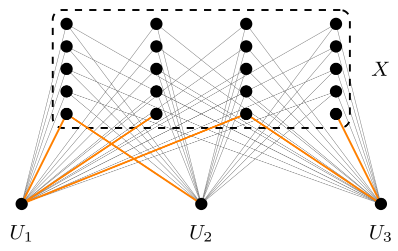
We now add "shortcuts" that allow queue based algorithms to bypass the construction. The core idea is to ensure the queue algorithm refines the set X into a discrete set within a single level of its breadth first behavior. This causes $\mathcal{X}$, $\mathcal{Y}$ and Y to completely split in subsequent rounds, thereby preventing the cycling behavior that causes superlinear cost. Indeed, if $\mathcal{X}$ and $\mathcal{Y}$ are handled only once, then the total cost is in $\mathcal{O}(2^k \cdot k^2) = \mathcal{O}(m)$.

Simultaneously we force the stack into the typical cycling behavior. We do so by forcing it to make the same splits of X as the simple starting gadget from Section 4 would.

We apply the following changes to define queue graphs G_k^{Q} (see Figure 5): we connect each vertex in X to a path of length k. We add the new starting gadget described above. We extend the paths p_1 and p_2 to a length of $k+2$ and connect the ends to the old starting vertices through unidirectional gadgets. We also attach a third path p_{Q} of length k to a_0 and a_1 and connect the i-th pair to the level-i blocks of the i-th vertices of the X-paths, again through unidirectional gadgets. Note that the graph has still a size of $\mathcal{O}(2^k \cdot k^2)$.



(a) Set cover instance.



(b) Result of the reduction.

Figure 6 Reduction of the set cover instance $S = \{a, b, c, d\}$ and $\mathcal{U} = \{\{d\}, \{b, c, d\}, \{a, b\}\}$. Orange lines indicate connections to elements of S, all other edges are connections to dummy elements.

Queue Behavior. Consider a queue based color refinement. It splits the pairs within the paths p_1, p_2, p_Q layer by layer. The splits of the i-th pair of p_Q induce a split of the i-th vertices of the X-paths into the binary blocks of level i. After $k+7$ many rounds, this leads to a split of X into the blocks of level k. Note that at the same time $\{p^0_\mathrm{end}\}$ or $\{p^1_\mathrm{end}\}$ will be able to split X. Therefore, we know that X will be fully discrete before any subset of $\mathcal{X}$ can be considered by the worklist.

Stack Behavior. Any stack based color refinement running on G^Q_k splits one of the paths p_1, p_2 in the starting gadget before splitting the path p_Q. This induces the worst case cycling behavior of the construction from [4]. The unidirectional gadgets and depth first strategy hinder the algorithm from distinguishing anything else in the starting gadget before the rest of the graph has been distinguished. Therefore, no "shortcut" can be applied and a stack based color refinement on G^Q_k has costs of at least $\Omega(m \log(n))$.

6 Approximation Hardness

Complementing our previous results, we now provide an approximation hardness result for computing optimal color refinement strategies. We begin by defining the optimal refinement worklist problem:

▶ **Problem** (Refinement Worklist Problem). *Given a colored graph (G, π), compute a minimal cost sequence of pairs of color classes $W = (C_1, X_1), \dots, (C_t, X_t)$ such that:*

1. *Refining with respect to W results in the stable coloring π^∞.*
2. *For all prefixes $(C_1, X_1), \dots, (C_s, X_s)$, the partial quotient graph obtained after refining C_i w.r.t. X_i for $i = 1, \dots, s-1$ contains C_s and X_s (as unions of color classes).*

The cost of a sequence W is the sum of the costs for refining with respect to all $(C_i, X_i) \in W$.

The approximation hardness result is based on a reduction from the set cover problem. The set cover problem takes a finite universe S and a set of subsets of S, i.e., $\mathcal{U} \subseteq 2^S$. The decision variant then asks whether there exists a selection of k subsets in $\mathcal{U}$ whose union equals S. For simplicity, we assume $\bigcup_{U \in \mathcal{U}} U = S$. Set cover is well-known to be NP-complete.

The optimization variant requires a minimal selection of subsets that cover S, i.e., a solution that minimizes k. This problem is known to be NP-hard. More specifically, it is known that unless P = NP, polynomial-time algorithm can only reach an approximation factor of $\Omega(\log(n))$ [14].

▶ **Theorem 4.** *Unless* $\mathsf{P} = \mathsf{NP}$*, polynomial-time algorithms may only reach an approximation factor of* $\Omega(\log(n))$ *for the refinement worklist problem.*

Proof. We reduce the optimization variant of the set cover problem to the refinement worklist problem. More specifically, we reduce it in a manner which allows control of the parameters, so that the approximation hardness result of set cover immediately transfers to refinement worklists. The reduction is illustrated in Figure 6.

Given a set cover instance $(S,\mathcal{U})$ we define a related colored graph (G, π). We create one large color class X containing all elements of the universe S, as well as n^2 *dummy elements* (where n is the size of the set cover instance). Hence, the size of X is $n^2 + |S|$.

We add a singleton color class for each subset $U \in \mathcal{U}$, i.e., we add vertex U with color U. We connect the vertex U with all vertices of X *except* for the elements that are contained in U. Formally, we define the edges $E(G) := \{\{U, x\} \mid x \in X \wedge x \notin U \}$. Note that U has $n^2 + |S| - |U|$ connections to X.

In the constructed graph, all elements of the universe are eventually distinguished from the dummy elements in X. Refining X with respect to X is not productive, since there are no edges present and no splits occur. The only way to distinguish elements of X is to refine X with respect to an element of $\mathcal{U}$. Doing so always distinguishes all the elements contained in $U \in \mathcal{U}$ from the dummy elements and other remaining elements of X. Overall, we need to refine X with a subset of $\mathcal{U}$ that forms a set cover of S.

After that, assuming all elements of S have been distinguished from the dummy elements, it might be possible to split the resulting classes further through their connections to $\mathcal{U}$. However, the total cost for these further refinements is bounded by $c \cdot n^2$ for some fixed constant c.

The cost for refining X with respect to U is $n^2 + m$, where m is the number of remaining elements of S in X *after* the elements of U have been removed. Since we need to choose at most $|S|$ subsets in a reasonable solution (otherwise we could remove redundant elements from the solution), and each time X gets smaller by at least one element, the cost incurred by m over all subsets is at most $|S|^2 \leq n^2$. Ignoring the cost of m, we get that each subset incurs additional cost of n^2 through the dummy elements.

Hence, the final cost is upper bounded by $c \cdot n^2 + (N_U + 1) \cdot n^2 = (N_U + c + 1)n^2$ and lower bounded by $N_U n^2$, where N_U is the number of chosen subsets.

We finish our arguments with a proof by contradiction. Assume there is a polynomial-time algorithm with an approximation factor in $o(\log(n))$. Given a set cover instance $(S,\mathcal{U})$, we apply the polynomial-time reduction stated above. Assume now we get an approximate solution with cost $x \cdot n^2$. We know that this implies a set cover solution with cost at most x.

The optimal set cover solution with cost x' would imply a worklist solution with cost at most $(x' + c)n^2$ (for a fixed c). Hence, we know that the worklist solution also approximates the optimal solution of the original set cover instance with a factor in $o(\log(n))$.

The set cover instance has a size in the 3rd root of the size of the refinement worklist problem. But since $o(\log(n)) = o(\log(\sqrt[3]{n}))$, we get a contradiction to the approximation hardness result of set cover. ◀

References

1 Markus Anders and Pascal Schweitzer. Engineering a fast probabilistic isomorphism test. In *2021 Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 73–84. SIAM, 2021. doi:10.1137/1.9781611976472.6.

2 Markus Anders, Pascal Schweitzer, and Florian Wetzels. Comparative design-choice analysis of color refinement algorithms beyond the worst case. *CoRR*, abs/2103.10244, 2021. full version of the paper. arXiv:2103.10244.

3 Vikraman Arvind, Frank Fuhlbrück, Johannes Köbler, Sebastian Kuhnert, and Gaurav Rattan. The parameterized complexity of fixing number and vertex individualization in graphs. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, volume 58 of *LIPIcs*, pages 13:1–13:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. `doi:10.4230/LIPIcs.MFCS.2016.13`.

4 Christoph Berkholz, Paul S. Bonsma, and Martin Grohe. Tight lower and upper bounds for the complexity of canonical colour refinement. *Theory Comput. Syst.*, 60(4):581–614, 2017. `doi:10.1007/s00224-016-9686-0`.

5 Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identifications. *Comb.*, 12(4):389–410, 1992. `doi:10.1007/BF01305232`.

6 Martin Grohe. Equivalence in finite-variable logics is complete for polynomial time. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 264–273. IEEE Computer Society, 1996. `doi:10.1109/SFCS.1996.548485`.

7 Martin Grohe, Kristian Kersting, Martin Mladenov, and Erkal Selman. Dimension reduction via colour refinement. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 505–516. Springer, 2014. `doi:10.1007/978-3-662-44777-2_42`.

8 J.E. Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In *Theory of Machines and Computations*, pages 189–196. Academic Press, 1971.

9 Tommi A. Junttila and Petteri Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of the Nine Workshop on Algorithm Engineering and Experiments, ALENEX 2007, New Orleans, Louisiana, USA, January 6, 2007*. SIAM, 2007. `doi:10.1137/1.9781611972870.13`.

10 Sandra Kiefer and Brendan D. McKay. The iteration number of colour refinement. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 73:1–73:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.ICALP.2020.73`.

11 Brendan D. McKay. Practical graph isomorphism. In *10th. Manitoba Conference on Numerical Mathematics and Computing (Winnipeg, 1980)*, pages 45–87, 1981.

12 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *J. Symb. Comput.*, 60:94–112, 2014. `doi:10.1016/j.jsc.2013.09.003`.

13 Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 4602–4609. AAAI Press, 2019. `doi:10.1609/aaai.v33i01.33014602`.

14 Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 475–484. ACM, 1997. `doi:10.1145/258533.258641`.

15 Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561, 2011. URL: `http://dl.acm.org/citation.cfm?id=2078187`.