Tight Lower and Upper Bounds for the Complexity of Canonical Colour Refinement

Christoph Berkholz · Paul Bonsma · Martin Grohe

Published online: 27 June 2016
© The Author(s) 2016. This article is published with open access at Springerlink.com

Abstract An assignment of colours to the vertices of a graph is stable if any two vertices of the same colour have identically coloured neighbourhoods. The goal of colour refinement is to find a stable colouring that uses a minimum number of colours. This is a widely used subroutine for graph isomorphism testing algorithms, since any automorphism needs to be colour preserving. We give an \(O((m + n) \log n)\) algorithm for finding a canonical version of such a stable colouring, on graphs with \(n\) vertices and \(m\) edges. We show that no faster algorithm is possible, under some modest assumptions about the type of algorithm, which captures all known colour refinement algorithms.

Keywords Graph isomorphism · Colour refinement · Partition refinement · Canonical labelling

Supported by the European Community’s Seventh Framework Programme (FP7/2007-2013), grant agreement nº 317662.

Supported by the German Research Foundation DFG Koselleck Grant GR 1492/14-1.

An extended abstract of this paper has appeared in the proceedings of ESA’13, LNCS 8125, pp 145–156 (2013).

Paul Bonsma
p.s.bonsma@ewi.utwente.nl

1 Institut für Informatik, Humboldt-Universität zu Berlin, Unter den Linden 6, 10099, Berlin, Germany

2 Faculty of EEMCS, University of Twente, PO Box 217, 7500 AE Enschede, The Netherlands

3 RWTH Aachen, Lehrstuhl für Informatik 7, Ahornstr. 55, 52074 Aachen, Germany


1 Introduction

Colour refinement (also known as naive vertex classification) is a very simple, yet extremely useful algorithmic routine for graph isomorphism testing. It classifies the vertices by iteratively refining a colouring of the vertices as follows. Initially, all vertices have the same colour. Then in each step of the iteration, two vertices that currently have the same colour get different colours if for some colour $c$ they have a different number of neighbours of colour $c$. The process stops if no further refinement is achieved, resulting in a stable colouring of the graph. To use colour refinement as an isomorphism test, we can run it on the disjoint union of two graphs. Any isomorphism needs to map vertices to vertices of the same colour. So, if the stable colouring differs on the two graphs, that is, if for some colour $c$, the graphs have a different number of vertices of colour $c$, then we know they are nonisomorphic, and we say that colour refinement distinguishes the two graphs. Babai, Erdős, and Selkow [2] showed that colour refinement distinguishes almost all graphs (in the $G(n, 1/2)$ model). In fact, they proved the stronger statement that the stable colouring is discrete on almost all graphs, that is, every vertex gets its own colour. On the other hand, colour refinement fails to distinguish any two regular graphs with the same number of vertices, such as a 6-cycle and the disjoint union of two triangles.

Colour refinement is not only useful as a simple isomorphism test in itself, but also as a subroutine for more sophisticated algorithms, both in theory and practice. For example, Babai and Luks’s [1, 3] $O(2^{\sqrt{n \log n}})$-algorithm — this is still the best known worst-case running time for isomorphism testing — uses colour refinement as a subroutine, and most practical graph isomorphism tools (for example, [11, 18, 20, 22]), starting with McKay’s “Nauty” [20, 21], are based on the individualisation refinement paradigm (see also [23]). The basic idea of the latter algorithms is to recursively compute a canonical labelling of a given graph, which may already have an initial colouring of its vertices, as follows. We run colour refinement starting from the initial colouring until a stable colouring is reached. If the stable colouring is discrete, then this already gives us a canonical labelling (provided the colours assigned by colour refinement are canonical, see below). If not, we pick some colour $c$ with more than one vertex. Then for each vertex $v$ of colour $c$, we modify the stable colouring by assigning a fresh colour to $v$ (that is, we “individualise” $v$) and recursively call the algorithm on the resulting vertex-coloured graph. Then for each $v$ we get a canonically labelled version of our graph, and we return the lexicographically smallest among these. (More precisely, each canonical labelling of a graph yields a canonical string encoding, and we compare these strings lexicographically.) To turn this simple procedure into a practically useful algorithm, various heuristics are applied to prune the search tree. They exploit automorphisms of the graph found during the search. However, crucial for any implementation of such an algorithm is a very efficient colour refinement procedure, because colour refinement is called at every node of the search tree.

Colour refinement can be implemented to run in time $O((n + m) \log n)$, where $n$ is the number of vertices and $m$ the number of edges of the input graph. To our knowledge, this was first proved by Cardon and Crochemore [9]. Later Paige and
Tarjan [24, p.982] sketched a simpler algorithm. Both algorithms are based on the partitioning techniques introduced by Hopcroft [15] for minimising finite automata. However, an issue that is neglected in the literature is that, at least for individualisation refinement, we need a version of colour refinement that produces a canonical colouring. That is, if $f$ is an isomorphism from a graph $G$ to a graph $H$, then for all vertices $v$ of $G$, $v$ and $f(v)$ should get the same colour in the respective stable colourings of $G$ and $H$. However, none of the aforementioned algorithms seem to produce canonical colourings. We present an implementation of colour refinement that computes a canonical stable colouring in time $O((n + m) \log n)$. Ignoring the canonical part, our algorithmic techniques are similar to known results: like [24] and various other papers, we use Hopcroft’s strategy of ‘ignoring the largest new cell’, after splitting a cell [15]. Our data structures have some similarities to those described by Junttila and Kaski [18]. Nevertheless, since [18] contains no complexity analysis, and [17, 24] omit various (nontrivial) implementation details, it seems that the current paper gives the first detailed description of an $O((n + m) \log n)$ algorithm that uses this strategy. On a high level, our algorithm is also quite similar to McKay’s canonical colour refinement algorithm [20, Alg. 2.5], but with a few key differences which enable an $O((n + m) \log n)$ implementation. McKay [20] gave an $O(n^2 \log n)$ implementation using adjacency matrices. Our algorithm is described and analysed in Section 3. In Section 3.4, we discuss extensions: We show how the algorithm can be applied to directed, undirected and edge coloured graphs, and how the complexity bound in fact applies to an entire branch of an individualisation refinement algorithm.

Now the question arises whether colour refinement can be implemented in linear time. After various attempts, we started to believe that it cannot. Of course with currently known techniques one cannot expect to disprove the existence of a linear time algorithm for the standard (RAM) computation model, or for similar general computation models. Instead, we prove a tight lower bound for a restricted, but very broad class of algorithms. In this sense, our result is comparable to the lower bounds for comparison based sorting algorithm. Actually, our class of partition-refinement based algorithms captures all known colour refinement algorithms, and in fact every reasonable algorithmic strategy we could think of. We use the following assumptions. (See Sections 2 and 4 for precise definitions.) Colour refinement algorithms start with a unit partition (which has one cell $V(G)$), and iteratively refine this until a stable partition, or colouring, is obtained. This is done using refining operations: choose a union of current partition cells as refining set $R$, and choose another (possibly overlapping) union of partition cells $S$. Cells in $S$ are split up if their neighbourhoods in $R$ provide a reason for this. (That is, two vertices in a cell in $S$ remain in the same cell only if they have the same number of neighbours in every cell in $R$.) This operation requires considering all edges between $R$ and $S$, so the number of such edges is a very reasonable and modest lower bound for the complexity of such a refining step; we call this the cost of the operation. We note that a naive algorithm might choose $R = S = V(G)$ in every iteration. This then requires time $\Omega(mn)$ on graphs that require a linear number of refining operations, such as paths. Therefore, all fast algorithms are based on choosing $R$ and $S$ smartly (and on implementing refining steps efficiently).
For our main lower bound result, we construct a class of instances such that any possible sequence of refining operations that yields the stable partition has total cost at least $\Omega((m+n) \log n)$. Note that it is surprising that a tight lower bound can be obtained in this model. Indeed, cost upper bounds in this model would not necessarily yield corresponding algorithms, since firstly we allow the sets $R$ and $S$ to be chosen nondeterministically, and secondly, it is not even clear how to refine $S$ using $R$ in time proportional to the number of edges between these classes. However, as we prove a lower bound, this makes our result only stronger. An alternative formulation of our lower bound result is to model the class of nondeterministic partition-refinement based algorithms as a “proof system” and then prove lower bounds on the length of derivations (see the first author’s PhD-thesis [4] for details). We formulate the lower bound result for undirected graphs and non-canonical colour refinement, so that it also holds for digraphs, and canonical colour refinement. These results are presented in Section 4. Our construction also yields corresponding lower bounds for the problems of computing the bisimilarity relation on a transition system and for computing the equivalence classes induced by the 2-variable fragment of first-order logic $L^2$ on a structure (see Section 4.4).

Colour refinement has a natural “higher dimensional” generalisation known as the Weisfeiler-Leman algorithm. In the $k$-dimensional version, which we call $k$-WL in the following, we colour $k$-tuples of vertices instead of single vertices. For a detailed description of the algorithm and its history, we refer the reader to [8]. Using similar tricks as for colour refinement, one can implement $k$-WL to run in time $O(n^k \log n)$ [17]. The question arises if this is optimal. At the time of writing this paper we did not know a better than linear lower bound for $k$-WL. However, recently the first author of this paper jointly with Nordström [5] proved an $n/\Omega(k/\log k)$ lower bound for the number of refinement rounds $k$-WL requires. This implies an $n/\Omega(k/\log k)$ lower bound in the partition refinement model considered here. But of course there remains a gap between this lower bound and the $O(n^k \log n)$ upper bound.

### 2 Preliminaries

For an undirected (simple) graph $G = (V, E)$, $N(v)$ denotes the set of neighbours of $v \in V(G)$, and $d(v) = |N(v)|$ its degree. For a digraph, $N^+(v)$ and $N^-(v)$ denote the out- and in-neighbourhoods, and $d^+(v) = |N^+(v)|$ resp. $d^-(v) = |N^-(v)|$ the out- and in-degree, respectively. A partition $\pi$ of a set $V$ is a set $\{S_1, \ldots, S_k\}$ of pairwise disjoint nonempty subsets of $V$, such that $\bigcup_{i=1}^k S_i = V$. The sets $S_i$ are called cells of $\pi$. The order of $\pi$ is the number $|\pi|$ of cells. A partition $\pi$ is discrete if every cell has size 1, and unit if it has exactly one cell. Given a partition $\pi$ of $V$, and two elements $u, v \in V$, we write $u \equiv_{\pi} v$ if and only if there exists a cell $S \in \pi$ with $u, v \in S$. We say that a set $V' \subseteq V$ is $\pi$-closed if it is the union of a number of cells of $\pi$. In other words, if $u \equiv_{\pi} v$ and $u \in V'$ then $v \in V'$. For any subset $V' \subseteq V$, $\pi$ induces a partition $\pi[V']$ on $V'$, which is defined by $u \equiv_{\pi[V']} v$ if and only if $u \equiv_{\pi} v$, for all $u, v \in V'$.

Let $G = (V, E)$ be a graph. A partition $\pi$ of $V$ is stable for $G$ if for every pair of vertices $u, v \in V$ with $u \equiv_{\pi} v$ and $R \in \pi$, it holds that $|N(u) \cap R| = |N(v) \cap R|$. If $G$
is a digraph, then $|N^+(u) \cap R| = |N^+(v) \cap R|$ should hold. For readability, all further definitions and propositions in this section are formulated for (undirected) graphs, but the corresponding statements also hold for digraphs (replace degrees/neighbourhoods by out-degrees/out-neighbourhoods). One can see that if $\pi$ is stable and $d(u) \neq d(v)$, then $u \not\approx_{\pi} v$, which we will use throughout.

A partition $\rho$ of $V$ refines a partition $\pi$ of $V$ if for every $u, v \in V$, $u \approx_{\rho} v$ implies $u \approx_{\pi} v$. (In other words: every cell of $\pi$ is $\rho$-closed.) If $\rho$ refines $\pi$, we write $\pi \preceq \rho$. Note that $\preceq$ is a partial order on all partitions of $V$.

**Definition 1** Let $G$ be a graph, and let $\pi$ and $\pi'$ be partitions of $V(G)$. For vertex sets $R, S \subseteq V(G)$ that are $\pi$-closed, we say that $\pi'$ is obtained from $\pi$ by a refining operation $(R, S)$ if
- for every $S' \in \pi$ with $S' \cap S = \emptyset$, it holds that $S' \in \pi'$, and
- for every $u, v \in S$: $u \approx_{\pi'} v$ if and only if $u \approx_{\pi} v$ and for all $R' \in \pi$ with $R' \subseteq R$, $|N(u) \cap R'| = |N(v) \cap R'|$ holds.

Note that if $\pi'$ is obtained from $\pi$ by a refining operation $(R, S)$, then $\pi \preceq \pi'$. We say that the operation $(R, S)$ is effective if $\pi < \pi'$. In this case, at least one cell $C \in \pi$ is split, which means that $C \notin \pi'$. Note that an effective refining operation exists for $\pi$ if and only if $\pi$ is unstable. In addition, the next proposition says that if the goal is to obtain a (coarsest) stable partition, then applying any refining operation is safe.

**Proposition 2** Let $\pi'$ be obtained from $\pi$ by a refining operation $(R, S)$. If $\rho$ is a stable partition with $\pi \preceq \rho$, then $\pi \preceq \pi' \preceq \rho$.

**Proof** $\pi \preceq \pi'$ follows immediately from the definitions. Now consider $u, v$ with $u \approx_{\rho} v$, and thus $u \approx_{\pi} v$. Then for any $R' \in \pi$, $d_{R'}(u) = d_{R'}(v)$. This holds because $R'$ is a union of sets in $\rho$, and for all these sets this property holds since $\rho$ is stable. Therefore, $u \approx_{\pi'} v$. \qed

A partition $\pi$ is a coarsest partition for a property $P$ if $\pi$ satisfies $P$, and there is no partition $\rho$ with $\rho < \pi$ that also satisfies property $P$.

**Proposition 3** Let $G = (V, E)$ be a graph. For every partition $\pi$ of $V$, there is a unique coarsest stable partition $\rho$ that refines $\pi$.

**Proof** For any partition $\pi$, the discrete partition refines $\pi$ and is stable, so there exists a stable partition that refines $\pi$. Because $\preceq$ is a partial order, there exists then at least one coarsest stable partition that refines $\pi$. Now suppose there exists a partition $\pi$ for which there exist at least two distinct coarsest stable partitions $\rho_1$ and $\rho_2$ that refine $\pi$. Choose such a partition $\pi$ so that $|\pi|$ is maximum. Clearly, $\pi$ is not stable (otherwise $\rho_1 = \pi = \rho_2$). So there exists at least one effective refining operation
(R, S) that can be applied to \( \pi \). For the resulting partition \( \pi' \), \( |\pi'| > |\pi| \) holds. By Proposition 2, both \( \rho_1 \) and \( \rho_2 \) refine \( \pi' \) as well. But since \( |\pi'| > |\pi| \), this contradicts the choice of \( \pi \).

3 A Fast Canonical Colour Refinement Algorithm

3.1 Canonical Colouring Methods

A colouring of a (di)graph \( G \) is a function \( \alpha : V(G) \to \mathbb{Z} \). (Note that adjacent vertices may receive the same colour.) It is a \( k \)-colouring if for every \( v \in V(G) \), \( \alpha(v) \in \{1, \ldots, k\} \). Given a colouring \( \alpha \) of \( G \) and \( i \in \mathbb{Z} \), we denote \( C_{\alpha}^i = \{v \in V(G) \mid \alpha(v) = i\} \). The set \( C_{\alpha}^i \) is called colour class \( i \). If the colouring is clear from the context, we also omit the superscript. For any colouring \( \alpha \) of a (di)graph \( G \), the set \( \{C_{\alpha}^i \mid i \in \mathbb{Z}, C_{\alpha}^i \neq \emptyset \} \) is a partition of \( V(G) \), which we will denote by \( \pi_{\alpha} \). We will call \( \alpha \) unit or stable if \( \pi_{\alpha} \) is unit or stable, respectively.

Given two (di)graphs \( G \) and \( G' \), with respective colourings \( \alpha \) and \( \alpha' \), an isomorphism \( h : V(G) \to V(G') \) is colour preserving for \( \alpha \) and \( \alpha' \) if for all \( v \in V(G) \), \( \alpha(v) = \alpha'(h(v)) \) holds. A colouring method is a method for obtaining a colouring \( \beta \) of a (di)graph \( G \), given an initial \( k \)-colouring \( \alpha \). (This method can be an algorithm, or simply a definition. Often, the initial colouring \( \alpha \) is chosen to be the unit colouring.) A colouring method (or algorithm) is called canonical if for any two isomorphic (di)graphs \( G \) and \( G' \) with initial colourings \( \alpha \) resp. \( \alpha' \) and isomorphism \( h : V(G) \to V(G') \), the following holds: if \( h \) is colour preserving for \( \alpha \) and \( \alpha' \), then \( h \) is colour preserving for the resulting colourings \( \beta \) and \( \beta' \). The resulting colouring \( \beta \) itself is also called a canonical colouring of \( G \), starting from \( \alpha \). If \( \alpha \) is the unit colouring, \( \beta \) is simply called a canonical colouring of \( G \).

For instance, for simple undirected graphs \( G \), the degree function \( d \), which assigns the colour \( d(v) = |N(v)| \) to each \( v \in V(G) \), yields canonical colouring of \( G \), because every isomorphism maps vertices to vertices of the same degree. (In other words: degrees are isomorphism invariant.) Obviously, a canonical colouring method is useful for deducing information about possible isomorphisms between two graphs, especially when the resulting partition \( \pi_{\beta} \) refines the initial partition \( \pi_{\alpha} \). For details on isomorphism testing algorithms based on this idea, we refer to [20, 23].

In this section we give a fast canonical algorithm that for any (di)graph \( G \) and colouring \( \alpha \) of \( G \), yields a colouring \( \beta \) of \( V(G) \) such that \( \pi_{\beta} \) is the coarsest stable partition that refines \( \pi_{\alpha} \). For ease of presentation, we require that the initial colouring \( \alpha \) is a surjective \( \ell \)-colouring for some value \( \ell \) (so every colour in \{1, \ldots, \ell\} occurs at least once). The resulting colouring \( \beta \) will then again be a surjective \( k \)-colouring for some value \( k \). In particular, if we choose \( \alpha \) to be the unit colouring, then \( \beta \) is a canonical colouring of \( G \) such that \( \pi_{\beta} \) is the unique coarsest stable partition of \( G \). To obtain the most general result, we formulate the algorithm for digraphs. Variants and extensions are discussed in Section 3.4.
3.2 High-level Description and Correctness Proofs

In Algorithm 1, we give a high-level description of our canonical colour refinement algorithm. This is not yet the fast implementation, and in fact, because we do not yet specify which data structures are used to represent the various mathematical objects (sets and functions), no sharp complexity bound can be concluded from it. In the next section, we give a detailed implementation of this algorithm, describe the data structures in detail, and prove the desired complexity bound. Here, we first focus on proving correctness of the algorithm.

In our algorithms, the scope of for loops, while loops and if-then-else statements is indicated by the indentation of blocks; because of space considerations we omit ‘end for’, ‘end while’ and ‘end if’ statements.

Algorithm 1 A canonical colour refinement algorithm

\textbf{Input}: A digraph $G$ on vertex set $\{1, \ldots, n\}$, with surjective $\ell$-colouring $\alpha$, and a sufficient refining colour set $S \subseteq \{1, \ldots, \ell\}$.

\textbf{Output}: A surjective canonical $k$-colouring $\beta$ of $G$, starting from $\alpha$, such that $\pi_\beta$ is the coarsest stable partition that refines $\pi_\alpha$.

1: For $i \in \{1, \ldots, n\}$:
2: \hspace{1cm} $C_i := \{v \in V(G) \mid \alpha(v) = i\}$
3: \hspace{1cm} $k := \ell$
4: $S_{\text{refine}} := \text{a stack containing all elements of } S \text{ in increasing order}$
5: \hspace{1cm} While $S_{\text{refine}}$ is not empty:
6: \hspace{2cm} $r := \text{pop}(S_{\text{refine}})$
7: \hspace{2cm} For every vertex $v \in V(G)$:
8: \hspace{3cm} $d_{\ell}^+(v) := |N^+(v) \cap C_r|$  
9: \hspace{2cm} $\text{Colors}_{\text{split}} := \{c \in \{1, \ldots, k\} \mid \exists v, w \in C_c \hspace{1mm} d_{\ell}^+(v) \neq d_{\ell}^+(w)\}$
10: \hspace{2cm} For all $s \in \text{Colors}_{\text{split}}$, in increasing order:
11: \hspace{3cm} $\text{maxcdeg} := \max_{v \in C_s} d_{\ell}^+(v)$
12: \hspace{3cm} For $i \in \{0, \ldots, \text{maxcdeg}\}$:
13: \hspace{4cm} $\text{numcdeg}(i) := |\{v \in C_s \mid d_{\ell}^+(v) = i\}|$
14: \hspace{3cm} $D := \{i \in \{0, \ldots, \text{maxcdeg}\} \mid \text{numcdeg}(i) \geq 1\}$
15: \hspace{3cm} $I := \{s\} \cup \{k + 1, \ldots, k + |D| - 1\}$
16: \hspace{3cm} Construct a bijection $f : D \rightarrow I$ such that $\forall i, j \in D$:
17: \hspace{4cm} $i < j \Rightarrow f(i) < f(j)$
18: \hspace{2cm} For every $v \in C_s$:
19: \hspace{3cm} if $f(d_{\ell}^+(v)) \neq s$ then
20: \hspace{4cm} remove $v$ from $C_s$
21: \hspace{4cm} add $v$ to $C_{f(d_{\ell}^+(v))}$
22: If $s \in S_{\text{refine}}$ then
23: \hspace{2cm} For $c \in I \setminus \{s\}$, in increasing order:
24: \hspace{3cm} $\text{Push}(S_{\text{refine}}, c)$
25: else
26: \hspace{2cm} $b := \min \{i \in \{0, \ldots, \text{maxcdeg}\} \mid \forall j \hspace{1mm} \text{numcdeg}(i) \geq \text{numcdeg}(j)\}$
27: \hspace{2cm} For $c \in I$, in increasing order:
28: \hspace{3cm} if $c \neq f(b)$ then push($S_{\text{refine}}, c$)
29: \hspace{2cm} $k := k + |I| - 1$
30: For $i \in \{1, \ldots, k\}$:
31: \hspace{2cm} For $v \in C_i$:
32: \hspace{3cm} $\beta(v) := i$
33: Return the colouring $\beta$ of $G$
The input to Algorithm 1 is a digraph $G = (V, E)$, with $V = \{1, \ldots, n\}$. For every vertex $v \in V$, the sets of out-neighbours $N^+(v)$ and in-neighbours $N^-(v)$ are given. (Alternatively, these can be computed in linear time from the edge list.) In addition, an $\ell$-colouring $\alpha$ of $G$ and a set $S \subseteq \{1, \ldots, \ell\}$ are given. The set $S$ should be a sufficient refining colour set for $\alpha$, which is a set that satisfies the following property: for any colour class $C^\alpha_i$ and two vertices $u, v \in C^\alpha_i$, if there exists a colour class $C^\alpha_j$ with $|N^+(u) \cap C^\alpha_j| \neq |N^+(v) \cap C^\alpha_j|$, then there exists a $j' \in S$ such that $|N^+(u) \cap C^\alpha_j| \neq |N^+(v) \cap C^\alpha_j|$. Note that $\{1, \ldots, \ell\}$ trivially forms a sufficient refining colour set for any $\ell$-colouring, but that smarter choices of $S$ may give a faster algorithm (which will be necessary in Section 3.4).

Throughout, the algorithm maintains an (ordered) partition $(C_1, \ldots, C_k)$ of $V(G)$, starting with the partition $(C^\alpha_1, \ldots, C^\alpha_k)$ (Lines 1–3). We also view this partition as a colouring, so the sets $C_i$ will be called colour classes, and indices $i \in \{1, \ldots, k\}$ will be called colours. In the main while-loop (Line 5), this partition is iteratively refined using refining operations of the form $(R, V)$, where $R = C_r$ for some $r \in \{1, \ldots, k\}$. We will show that when the algorithm terminates, no effective refining operations are possible on the resulting partition. So the resulting partition is the unique coarsest stable partition of $G$ that refines $\pi_\alpha$ (Propositions 2, 3). The next colour $r$ that is used as refining colour is chosen using a stack (sequence) $S_{\text{refine}}$ (Line 6), which contains all colours that still need to be considered. For a given refining colour class $C_r$ and any $v \in V$, call $d^+_r(v) := |N^+(v) \cap C_r|$ the colour degree of $v$ (with respect to colour $r$). Then every colour $s \in \{1, \ldots, k\}$ will be split up according to colour degrees (in the for-loop of Line 10). We only consider colours that actually split up, in increasing order. When splitting up colour class $C_s$, the new colours will be $s$ and $k + 1, \ldots, k + d - 1$, where $d$ is the number of different colour degrees that occur in $C_s$. These new colours are assigned to the vertices in $C_s$ according to increasing colour degrees (Lines 11–12).

It remains to explain how newly introduced colours are added to the stack $S_{\text{refine}}$. Initially, $S_{\text{refine}}$ contains all colours in $S$, in increasing order (Line 4). (To be precise, this means that the highest colour is on top of the stack.) Whenever new colours are introduced during the splitting of a colour class $C_s$, these are pushed onto the stack $S_{\text{refine}}$, in increasing order (Lines 21–27). There are however exceptions: for instance, if we have already used the vertex set $C_s$ as refining colour class before, and this set is split up into $d$ new colours, then it is not necessary to use all of these new colours as refining colours later; one colour $b$ may be omitted from $S_{\text{refine}}$ (Line 27). To obtain a good complexity, we choose $b$ such that the size of the corresponding colour class is maximised, in order to minimise the sizes of the refining colour sets used later during the computation. (This is Hopcroft’s trick [15], which was also used by e.g. [24].)

Informally, this algorithm is canonical since at every point, both the (colourings given by the) ordered partition $(C_1, \ldots, C_k)$ and stack $S_{\text{refine}}$ remain canonical; new colours that we assign to vertices, and the order in which colours are considered in the various loops of the algorithm, are completely determined by isomorphism-invariant values such as colour degrees and colour numbers. The order in which vertices of $G$ or neighbour lists are given in the input is irrelevant. A formal proof is given in Lemma 6 below. We first prove that Algorithm 1 returns the unique coarsest stable partition, which requires the following invariant.
Proposition 4  At the end of every iteration of the for-loop in Line 10 of Algorithm 1, \( \{C_1, \ldots, C_k\} \) is a partition of \( V(G) \) into nonempty sets, and the set of colours in \( S_{\text{refine}} \) is a sufficient refining colour set for the corresponding \( k \)-colouring of \( G \).

Proof  Since new colours correspond to colour degrees that actually occur (Lines 11–16), every new colour class will be nonempty. Lines 19 and 20 show that every vertex of \( G \) remains part of exactly one colour class. So the algorithm maintains a partition of \( V(G) \).

By definition, the set of colours in \( S_{\text{refine}} \) is a sufficient refining colour set before the first iteration. We prove that this invariant is maintained during any iteration of the for-loop, where colour class \( C_s \) for \( s \in \{1, \ldots, k\} \) is split up (by colour \( r \)), into the new colour classes \( C_{\sigma_1}, \ldots, C_{\sigma_p} \). Denote \( S = C_s \), as it is at the start of the iteration (so \( S = C_{\sigma_1} \cup \ldots \cup C_{\sigma_p} \)). Because the new colour classes form a partition of the old colour class \( S \), for every \( z \in V(G) \), it holds that

\[
|N^+(z) \cap S| = \sum_{j \in \{1, \ldots, p\}} |N^+(z) \cap C_{\sigma_j}|. \tag{1}
\]

Consider two vertices \( u, v \in V(G) \) that are in the same colour class after the refining operation, and therefore also before the refining operation. If \( |N^+(u) \cap S| \neq |N^+(v) \cap S| \), then there exists an \( i \in \{1, \ldots, p\} \) such that \( |N^+(u) \cap C_{\sigma_i}| \neq |N^+(v) \cap C_{\sigma_i}| \) (because of (1)). So if \( s \in S_{\text{refine}} \), then the invariant is maintained after splitting up the colour, since every new colour is added to \( S_{\text{refine}} \) (Lines 22–23), and \( s \) remains in \( S_{\text{refine}} \).

So now assume \( s \notin S_{\text{refine}} \). Then every colour in \( \{\sigma_1, \ldots, \sigma_p\} \) is added to \( S_{\text{refine}} \), except for \( i = f(b) \) (Line 27). Then we need to consider the case that \( |N^+(u) \cap C_{\sigma_i}| \neq |N^+(v) \cap C_{\sigma_i}| \), but \( |N^+(u) \cap C_{\sigma_j}| = |N^+(v) \cap C_{\sigma_j}| \) for all \( j \in \{1, \ldots, p\} \setminus \{i\} \). But then also \( |N^+(u) \cap S| \neq |N^+(v) \cap S| \) (because of (1)). Since \( s \notin S_{\text{refine}} \), and the invariant held before the refining operation, there exists another colour \( j^* \in S_{\text{refine}} \) such that \( |N^+(u) \cap C_{\sigma_{j^*}}| \neq |N^+(v) \cap C_{\sigma_{j^*}}| \). Since this colour remains in \( S_{\text{refine}} \), the invariant is also maintained in this case.

Using the above proposition, we can prove that Algorithm 1 computes a coarsest stable colouring, provided that \( S \) is a sufficient refining colour set. Recall that this condition is certainly satisfied when choosing \( S = \{1, \ldots, \ell\} \).

Lemma 5  Let \( G \) be a digraph, \( \alpha \) be a surjective \( \ell \)-colouring of \( G \), and let \( S \subseteq \{1, \ldots, \ell\} \) be a sufficient refining colour set for \( \alpha \). Then Algorithm 1 computes a surjective \( k \)-colouring \( \beta \) of \( G \) such that \( \pi_\beta \) is the coarsest stable partition that refines \( \pi_\alpha \).

Proof  Let \( \omega \) be the coarsest stable partition of \( V(G) \) that refines \( \pi_\alpha \). The partition \( \pi_\beta \) given by the algorithm is refined by \( \omega \) because it is obtained from \( \pi_\alpha \) using refining operations (Proposition 2). The stack \( S_{\text{refine}} \) is empty when the algorithm terminates, so the empty set is a sufficient refining colour set at this point (Proposition 4), and therefore \( \pi_\beta \) is stable. It follows that \( \pi_\beta \) is equal to \( \omega \) (Proposition 3). At any point, the
sets $C_i$ for $i \in \{1, \ldots, k\}$ are nonempty (Proposition 4), so the resulting $k$-colouring $\beta$ is surjective.

**Lemma 6** Algorithm 1 is a canonical colouring algorithm.

**Proof** Consider two digraphs $G$ and $G'$, with $\ell$-colourings $\alpha$ resp. $\alpha'$, and $S \subseteq \{1, \ldots, \ell\}$. For $i \in \mathbb{N}$, let $C_j^G,i$ (resp. $C_j^{G',i}$) denote the set $C_j$ as it is at the start of the $i$-th iteration of the while-loop in Line 5, when running Algorithm 1 with input $G, \alpha, S$ (resp. $G', \alpha', S$). Let $S_{\text{refine}}^G,i$ (resp. $S_{\text{refine}}^{G',i}$) denote the stack $S_{\text{refine}}$ as it is at the start of iteration $i$ of the while-loop in Line 5, when running Algorithm 1 with input $G, \alpha, S$ (resp. $G', \alpha', S$).

To show that Algorithm 1 is canonical, we prove by induction over $i$ that for every isomorphism $h : V(G) \to V(G')$ that is colour-preserving for $\alpha$ and $\alpha'$, the following properties are maintained: $S_{\text{refine}}^G,i = S_{\text{refine}}^{G',i}$, and for all $c$ and $v \in C_c^G,i$, it holds that $h(v) \in C_c^{G',i}$. For $i = 1$, the claim follows immediately from how $S_{\text{refine}}$ is initialised (Line 4), and how the sets $C_c$ are initialised (Line 2). We now consider the places in the algorithm where these sets and stacks are modified. In Line 6, the last element of both $S_{\text{refine}}^G,i$ and $S_{\text{refine}}^{G',i}$ is removed, so these sequences stay the same. Furthermore, it follows that the same colour is used as refining colour for both $G$ and $G'$ in this iteration. The induction assumption shows that $h$ is a colour preserving isomorphism for the colourings given by the various sets $C_c^G,i$ and $C_c^{G',i}$. So the isomorphism $h$ shows that for every $c$ and every $d$, $C_c^G,i$ and $C_c^{G',i}$ contain the same number of vertices with colour degree $d$. Hence the set $\text{Colors}_{\text{split}}$ is the same for both $G$ and $G'$, and for each colour $c \in \text{Colors}_{\text{split}}$, the values $\text{max cdeg}(j)$ (for every $j$) are the same. Therefore, in every iteration of the for-loop in Line 10, the sets $D, I$ will be the same for both $G$ and $G'$. The choice of the bijection $f$ in line 16 is unique because of the monotonicity; hence $f$ will be the same for $G$ and $G'$ as well. It follows that when in Lines 19 and 20, a vertex $v \in V(G)$ is moved from colour class $C_s^G,i$ to colour class $C_f(d_r^+(v))$, the vertex $h(v) \in V(G')$ is also moved from $C_s^{G',i}$ to $C_f(d_r^+(v))$, since $d_r^+(v) = d_r^+(h(v))$. Hence $h$ remains colour preserving for the new partition. From the previous observations it also follows that in Line 25, $b$ is chosen to be the same value for both $G$ and $G'$. Therefore, in Lines 27 and 23, the stack $S_{\text{refine}}$ is modified in the same way for both $G$ and $G'$ (note that in both cases, the colours are added in increasing order). This shows that the claimed properties are maintained in one iteration of the while-loop in Line 5, so by induction, $h$ is also a colour preserving isomorphism for the final colouring $\beta$ that is returned in Line 32.

### 3.3 Implementation and Complexity Bound

We now describe a fast implementation of Algorithm 1. The main idea of the complexity proof is the following: one iteration (of the main while-loop; Line 5 of Algorithm 1) consists of popping a refining colour $r$ from the stack $S_{\text{refine}}$, and
applying the refining operation \((R, V)\), with \(R = C_r\). Below we give implementation details and prove the following lemma:

**Lemma 7** Algorithm 1 can be implemented such that one iteration, in which a refining operation \((R, V)\) is applied, takes time

\[ O(|R| + D^-(R) + k \log k), \]

where \(D^-(R) = \sum_{v \in R} d^-(v)\) and \(k\) is the number of new colours that are introduced in this iteration. This implementation requires an initialisation step with complexity \(O(n)\).

Using the above lemma, we can prove the desired complexity bound. (The main idea is again based on Hopcroft’s idea [15].)

**Lemma 8** Algorithm 1 has an implementation with complexity \(O((n + m) \log n)\), where \(n = |V(G)|\) and \(m = |E(G)|\) for the input digraph \(G\).

**Proof** Consider a vertex \(v \in V(G)\). Let \(R^v_1, \ldots, R^v_q\) denote the refining colour classes \(C_r\) with \(v \in C_r\) that are considered throughout the computation, in chronological order. Then we observe that for all \(i \in \{1, \ldots, q - 1\} \), \(|R^v_i| \geq 2|R^v_{i+1}|\) holds. This holds because whenever a set \(S = C_s\) is split up into \(C_{\sigma_1}, \ldots, C_{\sigma_p}\), where \(s\) has been considered earlier as a refining colour (so it is not in \(S_{\text{refine}}\) anymore), then for all new colours \(\sigma_i\) that are added to the stack \(S_{\text{refine}}\), \(|C_{\sigma_i}| \leq \frac{1}{2} |S|\) holds, since the largest colour class is not added to \(S_{\text{refine}}\). Note that if a colour class \(C_{\sigma_i}\) is subsequently split up before \(\sigma_i\) is considered as refining colour, the bound of course also holds. It follows that every \(v \in V(G)\) appears at most \(\log_2 n\) times in a refining colour class. Then we can write

\[ \sum_R |R| + D^-(R) \leq \sum_{v \in V(G)} (1 + d^-(v)) \log_2 n = (n + m) \log_2 n, \]

where the first summation is over all refining colour classes \(R = C_r\) considered during the computation. In addition, the total number of new colours that is introduced is at most \(n\), since every colour class, after it is introduced, remains nonempty throughout the computation. So we may write

\[ \sum_i k_i \log k_i \leq \sum_i k_i \log n \leq n \log n, \]

where \(k_i\) denotes the number of colours introduced during iteration \(i\). Combining these bounds with Lemma 8 shows that the total complexity of the algorithm can be bounded by \(O(n) + O((n + m) \log n) + O(n \log n) \subseteq O((n + m) \log n)\).
Combining Lemmas 5, 6 and 8 (using \(S = \{1, \ldots, \ell\}\)), we obtain our main theorem:

**Theorem 9** For any digraph \(G\) on \(n\) vertices and \(m\) edges, with surjective \(\ell\)-colouring \(\alpha\), in time \(O((n + m) \log n)\) a canonical surjective \(k\)-colouring \(\beta\) of \(G\) can be computed such that \(\pi_{\beta}\) is the coarsest stable partition that refines \(\pi_{\alpha}\).

**Implementation Details** It remains to prove Lemma 7. In Algorithm 2 and its subroutine Algorithm 3, the detailed, fast implementation of Algorithm 1 is given. The colour classes \(C_i\) are represented by doubly linked lists \(C[i]\), indexed by \(i \in \{1, \ldots, n\}\). (So \(C\) is an array containing (pointers to) doubly-linked lists, indexed by colour numbers \(1, \ldots, n\).) For all lists \(L\), we keep track of their length, which we denote by \(|L|\).

The first challenge is how to compute the colour degrees \(d_i^+(v)\) efficiently for every \(v \in V(G)\) (Lines 7 and 8 of Algorithm 1), with respect to the refining colour \(r\), and corresponding colour class \(R\). For this we use an array \(cdeg[v]\) of integers, indexed by \(v \in \{1, \ldots, n\}\). We use the following invariant: at the beginning of every iteration, \(cdeg[v] = 0\) for all \(v\). Then we can compute these colour degrees by looping over all in-neighbours \(w\) of all vertices \(v \in R\), and increasing \(cdeg[w]\). At the same time, we compute the maximum colour degree for every colour \(c\), using an array \(\max cdeg\) (this is an array of integers indexed by \(c \in \{1, \ldots, n\}\)), we compute a list \(\text{Colors}_{\text{adj}}\) of colours \(i\) that contain at least one vertex \(w \in C_i\) with \(cdeg[w] \geq 1\), and for every such colour \(i\), we compute a list \(A[i]\) of all vertices \(w\) with \(cdeg[w] \geq 1\). (So \(A\) is an array containing (pointers to) lists, indexed by colour numbers \(1, \ldots, n\).) None of these lists contain duplicates. See Lines 47–54 of Algorithm 2. This implementation is correct because we also maintain the following invariant: at the beginning of every iteration, \(\max cdeg[c] = 0\) and \(A[c]\) is an empty list, for every \(c\), \(\text{Colors}_{\text{adj}}\) is an empty list, and flags are maintained for colours to keep track of membership in \(\text{Colors}_{\text{adj}}\). To maintain this invariant, we reset all of these data structures again at the end of every iteration (Lines 68–73). Note that it suffices to only reset \(cdeg[v]\) for vertices \(v\) that occur in some list \(A[c]\) (Lines 69–70).

Next, we address how we can consider all colours that split up in one iteration, in canonical (increasing) order (see Lines 9,10 of Algorithm 1 and Lines 55–66). To this end, we compute a new list \(\text{Colors}_{\text{split}}\), which represents the subset of \(\text{Colors}_{\text{adj}}\) containing all colours that actually split up. This is necessary since this list needs to be sorted, in order to consider the colours in canonical order (in the for-loop in Line 66). By ensuring that all colours in \(\text{Colors}_{\text{split}}\) split up, we have that \(|\text{Colors}_{\text{split}}| \leq k\) (where \(k\) is the number of colours introduced in this iteration), and therefore we can afford to sort this list. This can be done using any list sorting algorithm of complexity \(O(k \log k)\), such as merge sort. To compute which colours split up, we compute for every colour \(c \in \text{Colors}_{\text{adj}}\) the maximum colour degree \(\max cdeg[c]\) and minimum colour degree \(\min cdeg[c]\). The values \(\max cdeg[c]\) were computed before. Observe that we have \(\min cdeg[c] = 0\) if \(|A[c]| < |C[c]|\). Otherwise, we can afford to compute \(\min cdeg[c]\) by iterating over \(A[c]\) (see Lines 55–61).
Finally, we need to show how a single colour class $S = C[s]$ can be split up efficiently, and how the appropriate new colours can be added to the stack $S_{\text{refine}}$ in the proper order (Lines 10–28 of Algorithm 1). The details of this procedure are given in Algorithm 3. Firstly, for every relevant $d$, we compute how many vertices in $C[s]$ have colour degree $d$. These values are stored in an array $\text{numcdeg}[d]$, indexed by $d \in \{0, \ldots, \text{max cdeg}[s]\}$ (Lines 76–80). Using this array $\text{numcdeg}$, we can easily compute the (minimum) colour degree $b$ that occurs most often in $S$ (Lines 81–83), which
corresponds to the new colour that is possibly not added to $S_{\text{refine}}$. Using numcdeg, we can also easily construct an array $f$, indexed by $d \in \{0, \ldots, \max \text{cdeg}[s]\}$, which represents the mapping from colour degrees that occur in $S$ to newly introduced colours, or to the current colour $s$ (Lines 85–93). Finally, we can move all vertices $v \in A[s]$ from $C[s]$ to $C[i]$, where $i = f[\text{cdeg}[v]]$ is the new colour that corresponds to the colour degree of $v$ (Lines 94–98). Note that looping over $A[s]$ suffices, because if there are vertices in $C[s]$ with colour degree 0, then these keep the same colour, and thus do not need to be addressed. This fact is essential since the number of such vertices may be too large to consider, for our desired complexity bound. In conclusion, Algorithms 2 and 3 are indeed implementations of Algorithm 1. We now prove Lemma 7 by analysing the complexity.

**Algorithm 3 Subroutine SPLITUPCOLOUR($s$)**

75:  maxcdeg := maxcdeg[$s$]
76:  For $i \in [1, \ldots, \max \text{cdeg}]$:
77:      numcdeg[$i$] := 0
78:  numcdeg[0] := $|C[s]| - |A[s]|$
79:  For $v \in A[s]$:
80:      numcdeg[\text{cdeg}[v]] := numcdeg[\text{cdeg}[v]] + 1
81:  $b := 0$
82:  For $i \in [1, \ldots, \max \text{cdeg}]$:
83:      If numcdeg[$i$] > numcdeg[$b$] then $b := i$
84:      If $s \in S_{\text{refine}}$ then instack := 1 else instack := 0 // maintain flag for this test
85:  For $i \in [0, \ldots, \max \text{cdeg}]$:
86:      If numcdeg[$i$] $\geq$ 1 then
87:          If $i = \min \text{cdeg}[s]$ then
88:              $f[i] := s$
89:              If instack = 0 and $b \neq i$ then push($S_{\text{refine}}$, $f[i]$)
90:          else
91:              $k := k + 1$
92:              $f[i] := k$
93:              If instack = 1 or $i \neq b$ then push($S_{\text{refine}}$, $f[i]$)
94:  For $v \in A[s]$:
95:      If $f[\text{cdeg}[v]] \neq s$ then
96:          Delete $v$ from $C[s]$
97:      Append $v$ to $C[f[\text{cdeg}[v]]]$
98:  colour[$s$] := $f[\text{cdeg}[v]]$

**Proof of Lemma 7** The given implementation uses a number of arrays of length $n$, either containing integers (cdeg, colour, max cdeg, numcdeg, $f$), or containing (pointers to) lists/doubly linked lists ($C$, $A$). All of these arrays can be initialised in time $O(n)$. In general, the initialisation steps (Lines 33–44) take time $O(n)$ (for Line 43, use bucket sort).

We first consider the complexity of the subroutine SPLITUPCOLOUR($s$), given in Algorithm 3. We prove that it terminates in time $O(D_R^+(S))$, where $R = C[r]$ denotes the refining colour class, $S = C[s]$ denotes the class to be split up, and $D_R^+(S) = \sum_{v \in S} |N^+(v) \cap R|$. Every (non-loop) line takes constant time. For the list deletion (Line 96), this requires a proper implementation of doubly linked lists. The test in Line 84 whether $s \in S_{\text{refine}}$ can be done in constant time by maintaining a 0/1 flag for every colour, which indicates whether the colour is in $S_{\text{refine}}$. Since colours are added to and deleted from the stack $S_{\text{refine}}$ one by one, maintaining these flags is no problem. All for-loops in Algorithm 3 are repeated either max cdeg[$s$] times or
\(|A[s]|\) times. Both values are bounded by \(D^+_R(S)\). So the total complexity of one call to the subroutine can be bounded by \(O(D^+_R(S))\).

Now consider the complexity of one while loop iteration of Algorithm 2. The first two (nested) for loops (Lines 47–54) take time \(O(|R| + D^-(R))\). This holds because in total, \(D^-(R)\) choices of \(w\) are considered, and the operations for every such choice take constant time. The test in Line 51 can be implemented in constant time using a 0/1 flag that keeps track of whether a colour appears in Colors\(_{adj}\). Since elements are added to and deleted from Colors\(_{adj}\) one by one (Lines 52, 73), maintaining these flags is again no problem.

Since \(|\text{Colors}_{adj}| \leq D^-(R)\), the complexity of the for loops in Lines 55 and 63 can be bounded by \(O(D^-(R))\). Sorting \(\text{Colors}_{\text{split}}\) takes time \(O(k \log k)\), when using e.g. merge sort, since \(|\text{Colors}_{\text{split}}| \leq k\) (every colour in \(\text{Colors}_{\text{split}}\) will split up and thus introduce at least one new colour). One call to the subroutine \(\text{SPLITUPCOLOUR}(s)\) takes time \(O(D^+_R(S))\), with \(S = C[s]\), as shown above. Since

\[
\sum_{s \in \text{Colors}_{\text{split}}} D^+_R(C[s]) \leq D^-(R),
\]

the complexity of the for-loop in Line 66 can be bounded by \(O(D^-(R))\). The complexity of the last for-loop (Line 68) can also be bounded by \(O(D^-(R))\). Note in particular that in total, at most \(D^-(R)\) choices of \(v\) are considered in Line 70. This shows that the complexity of one iteration of the while-loop can be bounded by \(O(|R| + D^-(R) + k \log k)\). \(\square\)

### 3.4 Extensions, Generalisations and Variants

#### Stack vs. queue
In our algorithm, we use a stack to select the next colour that should be used for the next refining operation, whereas previous similar algorithms use a queue [20, 24]. Firstly, we remark that if we replace the stack by a queue, it can easily be checked that all of the claims proved in the previous sections still hold. So the best choice is determined by other concerns, which we now shortly discuss.

Using a queue gives the nice property that during the algorithm execution, all of the following ‘standard’ partitions will be generated: given an initial partition \(\pi = \pi_0\) of the vertices \(V\) of a graph \(G\), for every \(i \geq 0\) one can define \(\pi_{i+1}\) to be the partition obtained from \(\pi_i\) using the refining operation \((V, V)\). The coarsest stable partition of \(G\) that refines \(\pi\) is now the first partition \(\pi_i\) with \(\pi_i = \pi_{i+1}\). This characterisation is sometimes used as an alternative definition of coarsest stable partitions. One can verify that when using a queue, for every \(i\) a colouring \(\alpha\) with \(\pi_{\alpha} = \pi_i\) will be generated during the execution of the algorithm.

When using a stack, the behaviour of the algorithm seems somewhat less predictable. Nevertheless, this yields a ‘depth-first’ type of strategy that tends to give very small colour classes much quicker, which seems an advantage. In our own (limited) computational studies, we observed that using a stack was never worse than using a queue, and in some cases significantly better. Furthermore, we had an earlier
lower bound example construction that required time $\Omega((n + m) \log n)$ for a queue-based algorithm, but could be solved in time $O(n + m)$ using a stack-based algorithm. For these reasons, we would recommend using a stack.

Algorithm 4 Iterative Colour Refinement

**Input:** A digraph $G$ with $V(G) = \{1, \ldots, n\}$.

99: Compute a surjective canonical coarsest stable $k$-colouring $\beta$ of $G$.
100: While $k < n$:
101: \hspace{1em} $\alpha := \beta$
102: \hspace{1em} $\ell := k$
103: \hspace{1em} Choose a vertex $v$ with a non-unique colour in $\alpha$.
104: \hspace{1em} $\ell := \ell + 1$
105: \hspace{1em} $\alpha(v) := \ell$
106: \hspace{1em} Compute a surjective canonical $k$-colouring $\beta$ of $G$ such that $\pi_\beta$ is the coarsest stable partition that refines $\pi_\alpha$.
107: Return $\beta$

The Complexity of Iterative Refinement

Consider Algorithm 4. This algorithm takes as input a digraph $G$ on $n$ vertices, and returns a discrete colouring $\beta$ of $G$, or more precisely: a surjective $n$-colouring of $G$. This colouring is not canonical, since in Line 103, an arbitrary vertex is chosen to be *individualised*, that is, to receive a unique colour. So by itself this algorithm is not very interesting (there are easier ways to obtain an arbitrary discrete colouring of $G$). However, it corresponds to one recursion branch of various state of the art canonical labelling algorithms, based on the algorithm introduced by McKay [20]. We now shortly sketch how one should modify this algorithm (into a recursive algorithm) to obtain such a canonical labelling algorithm: In Line 103, instead choose a colour class $C^\alpha_i$ of the current colouring $\alpha$ with $|C^\alpha_i| \geq 2$. We branch on this colour class, as follows: for every $v \in C^\alpha_i$, continue with a separate branch of the algorithm where $v$ is individualised (Line 105), and a new stable colouring is computed (as shown in Line 106). Continuing recursively this way, one obtains a number of discrete colourings of $G$; one for every leaf of the recursion tree. A canonical discrete colouring of $G$ can be obtained by choosing one of these colourings that maximises some value. For instance, consider the adjacency matrix representation of $G$ where rows and columns are ordered according to the colour numbers, and view this as a binary number in the straightforward way. This is the basic algorithm; by keeping track of automorphisms of the graph, there are various ways to speed up the algorithm by pruning the recursion tree. For more details, we refer to [11, 18, 20–22].

The algorithm for obtaining a canonical discrete colouring $\beta$ for a digraph $G$ sketched above does not terminate in polynomial time for all graphs $G$. (If it did, this would yield a polynomial time isomorphism test: for two digraphs $G$ and $G'$, compute canonical discrete $n$-colourings $\beta$ and $\beta'$, respectively. Since $\beta$ and $\beta'$ are discrete $n$-colourings, they define a unique colour preserving bijection $h : V(G) \to V(G')$. Since $\beta$ and $\beta'$ are canonical, $G$ and $G'$ are isomorphic if and only if $h$ is an isomorphism.) Examples are known where such an algorithm will consider an exponential number of branches [23]. Nevertheless, a single branch of this algorithm (as
shown in Algorithm 4) terminates quickly. From [20] it follows that Algorithm 4 has an implementation that terminates in time $O(n^2 \log n)$. Using our results, we can show that it has an $O((n + m) \log n)$ implementation.

**Theorem 10** Algorithm 4 can be implemented such that it terminates in time $O((n + m) \log n)$, where $n = |V(G)|$ and $m = |E(G)|$ for the input graph $G$.

**Proof** The main part of the computation occurs in Lines 99 and 106, where we compute a surjective canonical $k$-colouring $\beta$ such that $\pi_\beta$ refines $\pi_\alpha$, for a given surjective $\ell$-colouring $\alpha$ (in Line 99, the unit colouring is chosen for $\alpha$). For this we use the fast implementation of Algorithm 1, given in Section 3.3. To obtain the desired complexity, we make the following simple changes, compared to Algorithm 1: we do not initialise the sets $C_i$ and stack $S\text{refine}$ every time we call the algorithm (Lines 1–4), and do not explicitly compute the new colouring $\beta$ (Lines 29–32). In addition, we do not actually copy the the colouring $\beta$ (Line 101 of Algorithm 4). Instead, we initialise these sets once, keep working with the same sets $\{C_1, \ldots, C_k\}$ throughout different iterations of the while-loop in Algorithm 4, and only compute the corresponding colouring $\beta$ at the very end of the algorithm. Whenever we individualise a vertex $v$ by assigning it a new colour (Line 105 of Algorithm 4), we move $v$ from its previous colour class $C_i$ to the new colour class $C_{\ell + 1}$. In addition, we update the stack $S\text{refine}$, which is currently empty, to contain the single colour $\ell + 1$. (This can both be done in constant time.)

We now argue that for computing the next stable colouring $\beta$ (Line 106), it is sufficient that $S\text{refine}$ contains only the colour $\ell + 1$. Denote by $\alpha_1$ the stable $\ell$-colouring before this step (with $\alpha_1(v) = i$), and by $\alpha_2$ the new ($\ell + 1$)-colouring (with $\alpha_2(v) = \ell + 1$). Consider the colour classes $C_i^{\alpha_1}$, $C_i^{\alpha_2}$ and $C_i^{\ell + 1}$, so $\{C_i^{\alpha_2}, C_i^{\ell + 1}\}$ is a partition of $C_i^{\alpha_1}$. Consider any two vertices $u$, $v$ with $\alpha_2(u) = \alpha_2(v)$. If $|N^+(u) \cap C_i^{\alpha_2}| \neq |N^+(v) \cap C_i^{\alpha_2}|$, then $|N^+(u) \cap C_i^{\ell + 1}| \neq |N^+(v) \cap C_i^{\ell + 1}|$, since $|N^+(u) \cap C_i^{\alpha_2}| = |N^+(v) \cap C_i^{\alpha_2}|$ (because $\alpha_1$ is stable). We conclude that $(\ell + 1)$ is a sufficient refining colour set for $\alpha_2$, so Algorithm 1 will compute the desired stable colouring $\beta$ when $S\text{refine}$ is initialised like this (Lemma 5).

We can now use the same argument as in the proof of Theorem 9 to show that the total complexity of all calls to Algorithm 1 (without the initialisation steps, as described above) is bounded by $O((n+m) \log n)$. Indeed, for every vertex $v \in V(G)$, if $R_1^v, \ldots, R_q^v$ denote the refining colour classes $C_r$ with $v \in C_r$ that are considered throughout the entire computation, in chronological order, then again for all $i \in \{1, \ldots, q - 1\}$ it holds that $|R_i^v| \geq 2|R_{i+1}^v|$. If $v$ is the vertex that is individualised in Line 105 (of Algorithm 4), then this holds because the next refining colour class that contains $v$ has size one, whereas the previous colour class that contained $v$ had size at least two (because $v$ was chosen with a non-unique colour in Line 103). In all other cases, the argument given in the proof of Theorem 9 applies. Following that proof, this shows that the total complexity of all refining operations done in Algorithm 4 can be bounded by $O((n + m) \log n)$.

It remains to bound the complexity of the other steps of Algorithm 4. As described above, the various sets are initialised only once, and the final colouring $\beta$ is computed only once, so this only adds a term $O(n)$ to the complexity. In addition, all steps in
the while-loop (Line 100) other than the stable colouring computation in Line 106 can be done in constant time, since we do not actually copy the colouring (Line 101). For the selection of the vertex \( v \) in Line 103, this claim is not entirely obvious, but one can observe that during the computation, one can maintain a doubly linked list that contains the colours of all colour classes of size at least two. This list can be updated in constant time whenever vertices are recoloured (so it does not change the total asymptotic complexity), and it can be used to select a vertex in Line 103 in constant time. The while-loop in Line 100 terminates after at most \( n \) iterations. In total, this shows that Algorithm 4 has an implementation with complexity \( O(n) + O(n) + O((n + m) \log n) \subseteq O((n + m) \log n) \).

We remark that in practice, one might wish to use smarter methods to select the vertex \( v \) to be individualised (Line 103), or more generally, to select the nontrivial colour class on which the recursive canonical labelling algorithm should branch. For instance, one can always branch on the smallest nontrivial colour class, or on the largest colour class\(^1\). In that case, an efficient heap-based priority queue implementation (see e.g. \cite{13}) can be used instead of a doubly-linked list to keep track of the sizes of colour classes, to attain the above complexity.

**Alternative Stability Criteria** We formulated our results only for digraphs, with stability defined only in terms of out-neighbours. We now summarise how our results should be modified to accommodate alternative stability criteria.

**Theorem 11** For any undirected graph \( G \) on \( n \) vertices with \( m \) edges, in time \( O((n + m) \log n) \) a canonical coarsest stable colouring can be computed.

**Proof** For an undirected graph \( G \), denote by \( G^* \) the digraph with \( V(G^*) = V(G) \), constructed by replacing every undirected edge by two directed edges in both directions. Observe that a colouring \( \alpha \) is stable for \( G \) if and only if it is stable for \( G^* \), so we can use the fast implementation of Algorithm 1 on input \( G^* \) to compute a coarsest stable colouring of \( G \). Next, observe that a bijection \( h : V(G) \to V(H) \) is an isomorphism from \( G \) to \( H \) if and only if it is a (digraph) isomorphism from \( G^* \) to \( H^* \). It follows that the computed colouring is a canonical coarsest stable colouring. \( \square \)

For a positive integer \( p \), we define a \( p \)-edge coloured digraph \( G \) to be a tuple \((V, E, c)\) where \((V, E)\) is a digraph that may have parallel edges and/or loops, and \( c : E \to \{1, \ldots, p\} \) is an edge colouring of \( G \). For \( e \in E \), we write \( e = (u, v) \) to denote that \( e \) is an edge from \( u \) to \( v \). For \( j \in \{1, \ldots, p\} \), \( v \in V \) and \( C \subseteq V \), denote \( d^+_j(v, C) = |\{e \in E \mid \exists w \in C e = (v, w) \wedge c(e) = j\}| \) (the number of edges of colour \( j \), leaving \( v \), with head in \( C \)). A (vertex) \( \ell \)-colouring \( \alpha \) of \( G \) is called **edge-colour stable** if for all \( u, v \in V \) with \( \alpha(u) = \alpha(v) \), all \( j \in \{1, \ldots, p\} \) and all \( i \in \{1, \ldots, \ell\} \), it holds that \( d^+_j(u, C_{\alpha}^i) = d^+_j(v, C_{\alpha}^i) \). For two \( p \)-edge coloured digraphs \( G = (V, E, c) \) and \( G' = (V', E', c') \), a bijection \( h : V \to V' \) is called

---

\(^1\)Our own computational tests showed, somewhat surprisingly, that branching on the **largest** colour class is clearly the best strategy of these two.
an isomorphism if for all \( j \in \{1, \ldots, p\} \) and \( u, v \in V \) (possibly the same), it holds that the number of edges of colour \( j \) from \( u \) to \( v \) equals the number of edges of colour \( j \) from \( h(u) \) to \( h(v) \). Using this notion of isomorphism, canonical colouring methods/canonical colourings for edge coloured digraphs are defined the same as before.

**Theorem 12** Let \( G = (V, E, c) \) be an edge coloured digraph with \( n = |V| \) and \( m = |E| \). In time \( O((n + m) \log(n + m)) \), a canonical coarsest edge-colour stable colouring can be computed for \( G \).

**Proof** In time \( O(n + m) \), we can construct the following digraph \( G' \) from \( G \), with vertex colouring \( \alpha \): Start with the vertex set \( V \) (the original vertices), and for every edge \( e \in E \) with \( e = (u, v) \), add a vertex \( v_e \) (the new vertices) and two edges \( (u, v_e) \) and \( (v_e, v) \). Assign colour \( \alpha(v_e) = c(e) \) to the new vertices, and colour \( \alpha(v) = 0 \) to the original vertices \( v \in V \).

We will now show that a colouring \( \beta \) of \( G \) is edge-colour stable if and only if there exists a stable colouring \( \beta' \) for \( G' \) that refines \( \alpha \), that coincides with \( \beta \) on \( V \).

Consider an edge-colour stable colouring \( \beta \) of \( G \). We extend it to a colouring \( \beta' \) of \( G' \), as follows: for each new vertex \( v_e \) that corresponds to an edge \( e = (u, v) \), assign the tuple \( (c(e), \beta(v)) \). Extend \( \beta \) by assigning new colours to the new vertices, according to the lexicographical order of these tuples. (So two new vertices receive the same colour if and only if they are assigned the same tuple, and a new vertex and an original vertex never receive the same colour.) The resulting colouring \( \beta' \) of \( G' \) clearly refines \( \alpha \), and is stable for \( G' \): for every vertex colour \( i \) used by \( \beta \), vertex \( u \in V \) and edge colour \( j \in \{1, \ldots, p\} \), the number \( d^+(u, C^\beta_i) \) (with respect to \( G \)) equals the number of out-neighbours of \( u \) in \( G' \) that have the colour corresponding to the tuple \((j, i)\). For the new vertices of \( G' \), the stability criterion follows easily.

For the other direction, consider a stable colouring \( \beta' \) of \( G' \) that refines \( \alpha \), and define \( \beta \) to be the restriction of \( \beta' \) to \( V \). We argue that \( \beta \) is edge-colour stable for \( G \). For two new vertices \( v_e \) and \( v_f \) of \( G' \), with respective out-neighbours \( x \) and \( y \), we have that \( \beta'(v_e) = \beta'(v_f) \) implies \( \beta'(x) = \beta'(y) \) and \( \alpha(v_e) = \alpha(v_f) \), so \( c(e) = c(f) \). This can be used to conclude that for any two vertices \( u, v \in V \), colour \( i \) and edge colour \( j \), if \( \beta(u) = \beta(v) \) then \( d^+_j(u, C^\beta_i) = d^+_j(v, C^\beta_i) \). So \( \beta \) is edge-colour stable for \( G \).

It follows that a coarsest edge-colour stable colouring \( \beta \) of \( G \) corresponds to a coarsest stable colouring \( \beta' \) of \( G' \) that refines \( \alpha \). Since we can compute such a colouring \( \beta' \) in a canonical way, we can compute such a colouring \( \beta \) in a canonical way (Theorem 9). It remains to consider the complexity. The graph \( G' \) and colouring \( \alpha \) can be constructed from \( G \) in time \( O(n + m) \). It has \( n + m \) vertices, and \( 2m \) edges. So \( \beta' \) can be computed in time \( O((n + 3m) \log(n + m)) = O((n + m) \log(n + m)) \) time, by Theorem 9.

We remark that for any class of edge-coloured digraphs where the number of edges is polynomially bounded in the number vertices (so they satisfy \( m \in O(n^d) \) for a constant \( d \)), we can write \( \log(n + m) \in O(\log n^d) = O(\log n) \). So for such a graph
class, the above lemma shows that a canonical coarsest edge-colour stable colouring can again be computed in time \( O((n + m) \log n) \).

The above theorem can be used for various stronger isomorphism tests. We now give details for one of these. For digraphs, we defined stability only considering out-neighbourhoods. Nevertheless, an isomorphism \( h \) between two digraphs not only maps the out-neighbourhood of a vertex \( v \) bijectively to the out-neighbourhood of \( h(v) \), but does the same with the in-neighbourhoods. So for the purpose of digraph isomorphism testing, the following stronger stability criterion is more useful: a \( k \)-colouring \( \alpha \) of a digraph \( G \) is \textit{bi-stable} if for every pair of vertices \( u, v \in V(G) \) with \( \alpha(u) = \alpha(v) \) and every colour \( c \in \{1, \ldots, k\} \), both \( |N^+(u) \cap C^a_c| = |N^+(v) \cap C^a_c| \) and \( |N^-(u) \cap C^a_c| = |N^-(v) \cap C^a_c| \) hold.

**Theorem 13** For any digraph \( G \) on \( n \) vertices with \( m \) edges, in time \( O((n + m) \log n) \) a canonical coarsest bi-stable colouring can be computed.

**Proof** Let \( V = V(G) \). Construct a 2-edge coloured digraph \( G' = (V, E, c) \) on the same vertex set as \( G \) as follows: for every edge \( (u, v) \in E(G) \), add an edge \( e = (u, v) \) to \( E \) with \( c(e) = 1 \), and an edge \( f = (v, u) \) to \( E \) with \( c(f) = 2 \). (Note that this may introduce parallel edges.) Observe that a colouring \( \alpha : V \to \{1, \ldots, k\} \) is edge-colour stable for \( G' \) if and only if it is bi-stable for \( G \), and that a canonical colouring method for \( G' \) is a canonical colouring method for \( G \). So Theorem 12 can be applied. We use that \( G' \) has \( 2m \in O(n^2) \) edges, which yields the complexity bound \( O((n + m) \log n) \).

\[ \square \]

4 Complexity Lower Bound

We shall prove our lower bound for undirected graphs; this makes it as general as possible. The **cost** of a refining operation \( (R, S) \) in a graph \( G \) is

\[ \text{cost}(R, S) := |\{(u, v) \mid uv \in E(G), u \in R, v \in S\}|. \]

This is basically the number of edges between \( R \) and \( S \), except that edges with both ends in \( R \cap S \) are counted twice. For a partition \( \pi \) that admits a refining operation \( (R, S) \), denote by \( \pi(R, S) \) the partition that results from this operation.

**Definition 14** Let \( G = (V, E) \) be a graph, and \( \pi \) be a partition of \( V \).

- If \( \pi \) is stable, then \( \text{cost}(\pi) := 0 \).
- Otherwise, \( \text{cost}(\pi) := \min_{R, S} \text{cost}(\pi(R, S)) + \text{cost}(R, S) \), where the minimum is taken over all effective refining operations \( (R, S) \) that can be applied to \( \pi \).

Note that this is well-defined; if \( \pi \) is unstable, then there exists at least one effective elementary refining operation \( (R, S) \), and for any such operation, \( |\pi(R, S)| > |\pi| \). We can now formulate the main result of this section.
Theorem 15 For every integer $k \geq 2$, there is a graph $G_k$ with $n \in O(2^kk)$ vertices and $m \in O(2^kk^2)$ edges, such that $\text{cost}(\alpha) \in \Omega((m + n) \log n)$, where $\alpha$ is the unit partition of $V(G_k)$.

Note that this theorem implies a complexity lower bound for all partition-refinement based algorithms for colour refinement, as discussed in the introduction. We will first prove some basic observations related to the above definitions, then give the construction of the graph, and finally prove Theorem 15.

4.1 Basic Observations

We start with two basic properties of stable partitions. The first proposition follows easily from the definitions.

Proposition 16 Let $G = (V, E)$ be a graph, and $\pi$ be a stable partition of $V$. For any $\pi$-closed subset $S \subseteq V$, $\pi[S]$ is a stable partition for $G[S]$.

Proposition 17 Let $G = (V, E)$ be a graph, and $\pi$ be a stable partition of $V$. For any $\pi$-closed set $S$ and vertices $u, v \in V$: if the distance from $u$ to $S$ is different from the distance from $v$ to $S$, then $u \not\approx_{\pi} v$.

Proof Denote the distance from a vertex $x$ to $S$ by $\text{dist}(x, S)$. W.l.o.g. we may assume that $\text{dist}(u, S) < \text{dist}(v, S)$, so in particular $\text{dist}(u, S)$ is finite. We prove the statement by induction over $\text{dist}(u, S)$. If $\text{dist}(u, S) = 0$ then $u \in S$ but $v \not\in S$. Since $S$ is $\pi$-closed, this implies $u \not\approx_{\pi} v$. Otherwise, $u$ is adjacent to a vertex $w$ with $\text{dist}(w, S) = \text{dist}(u, S) - 1$, but $v$ is not. Let $R \in \pi$ be the cell with $w \in R$. Then by induction, $|N(v) \cap R| = 0$, so $u \not\approx_{\pi} v$, since $\pi$ is stable.

For a partition $\pi$ of $V$, denote by $\pi_\infty$ the coarsest stable partition of $V$ that refines $\pi$.

Proposition 18 Let $\pi$ and $\rho$ be partitions of $V$ such that $\pi \preceq \rho \preceq \pi_\infty$. Then $\text{cost}(\pi) \geq \text{cost}(\rho)$.

Proof Let $(R, S)$ be a refining operation that can be applied to $\pi$, which yields $\pi'$. Then it can be observed that the operation $(R, S)$ can also be applied to $\rho$, and that for the resulting partition $\rho'$, it holds again that $\pi' \preceq \rho' \preceq \pi_\infty$ (Proposition 2 shows that $\rho' \preceq \pi_\infty$).

An induction proof based on this observation shows that a minimum cost sequence of refining operations that refines $\pi$ to $\pi_\infty$ can also be applied to $\rho$, to yield the stable partition $\pi_\infty$, at the same cost. Therefore, $\text{cost}(\pi) \geq \text{cost}(\rho)$.

A refining operation $(R, S)$ on $\pi$ is elementary if both $R \in \pi$ and $S \in \pi$. The next proposition shows that adding the word ‘elementary’ in Definition 14 yields an equivalent definition.
Proposition 19  Let $\pi$ be an unstable partition of $V(G)$. Then
\[
\cost(\pi) = \min_{R,S} \cost(\pi(R, S)) + \cost(R, S),
\]
where the minimum is taken over all effective elementary refining operations $(R, S)$ that can be applied to $\pi$.

Proof  Let $(R, S)$ an nonelementary refining operation for $\pi$, and let $\rho_1$ be the result of applying $(R, S)$ to $\pi$. We shall prove that there is a sequence of elementary refining operations of total cost at most $\cost(R, S)$ that, when applied to $\pi$, yields a partition $\rho_2$ that refines $\rho_1$. The claim follows by Proposition 18.

Suppose that $R$ consists of the cells $R_1, \ldots, R_q$ and $S$ consists of the cells $S_1, \ldots, S_p$. We apply the elementary refining operations $(R_i, S_j)$ for all $i \in \{1, \ldots, q\}$, $j \in \{1, \ldots, p\}$ in an arbitrary order and let $\rho_2$ be the resulting partition. The cost of these elementary refinements is
\[
\sum_i \sum_j \cost(R_i, S_j) = \sum_{i,j} |\{(u, v) | uv \in E(G), u \in R_i, v \in S_j\}|
= |\{(u, v) | uv \in E(G), u \in R, v \in S\}| = \cost(R, S).
\]
It is easy to see that $\rho_2$ refines $\rho_1$. Indeed, if $u, v \in S$ belong to the same class of $\rho_2$, then they belong to the same class $S_j$, and for all classes $R_i$ they have the same number of neighbours in $R_i$. Hence they have the same number of neighbours in $R = \bigcup_i R_i$, and this means that they belong to the same class of $\rho_1$. \qed

4.2 Construction of the Graph

For $k \in \mathbb{N}$, denote $B_k = \{0, \ldots, 2^k - 1\}$. For $\ell \in \{0, \ldots, k\}$ and $q \in \{0, \ldots, 2^\ell - 1\}$, the subset $B^\ell_q = \{q2^{k-\ell}, \ldots, (q+1)2^{k-\ell} - 1\}$ is called the $q$-th binary block of level $\ell$. Analogously, for any set of vertices with indices in $B_k$, we also consider binary blocks. For instance, if $X = \{x_i | i \in B_k\}$, then $X^\ell_q = \{x_i | i \in B^\ell_q\}$ is called a binary block of $X$. For such a set $X$, a partition $\pi$ of $X$ into binary blocks is a partition where every $S \in \pi$ is a binary block. A key fact for binary blocks that we will often use is that for any $\ell$ and $q$, $B^\ell_q = B^{\ell+1}_{2q} \cup B^{\ell+1}_{2q+1}$.

For every integer $k \geq 2$, we will construct a graph $G_k$. (An example for $k = 3$ is given in Fig. 1.) In its core this graph consists of the vertex set $X = \{x_i | i \in B_k\}$, $X' = \{x_i^j | i \in B_k, j \in \{1, \ldots, k\}\}$, $Y = \{y_i^j | i \in B_k, j \in \{1, \ldots, k\}\}$ and $Y = \{y_i^j | i \in B_k\}$. Every vertex $x_i$ is adjacent to $x_i^j$ for all $j \in \{1, \ldots, k\}$ and every $y_i$ is adjacent to all $y_i^j$. Furthermore, for all $i$, $j_1$, $j_2$ there is an edge between $x_i^{j_1}$ and $y_i^{j_2}$. (For $X'$, binary blocks are subsets of the form $X'_q := \{x_i^j | i \in B^\ell_q, j \in \{1, \ldots, k\}\}$, and for $Y$ the definition is analogous.)

We add gadgets to the graph to ensure that any sequence of refining operations behaves as follows. After the first step, which distinguishes vertices according to their degrees, $X$ and $Y$ are cells of the resulting partition. Next, $X$ splits up into two binary blocks $X'_0$ and $X'_1$ of equal size. This causes $X'$ to split up accordingly into $X'_0$ and $X'_1$. One of these cells will be used to halve $Y$ in the same way. This refining
The graph $G_3$ operation $(R, S)$ is expensive because $[R, S]$ contains half of the edges between $\mathcal{X}$ and $\mathcal{Y}$. Next, $Y$ can be split up into $Y_0^1$ and $Y_1^1$. Once this happens, there is a gadget $\text{AND}_1$ that causes the two cells $X_0^1, X_1^1$ to split up into the four cells $X_q^2$, for $q = 0, \ldots, 3$. Again, this causes cells in $\mathcal{X}, \mathcal{Y}$ and $Y$ to split up in the same way and to achieve this, half of the edges between $\mathcal{X}$ and $\mathcal{Y}$ have to be considered. The next gadget $\text{AND}_2$ ensures that if both cells of $Y$ are split, then the four cells of $X$ can be halved again, etc. In general, we design a gadget $\text{AND}_\ell$ of level $\ell$ that ensures that if $Y$ is partitioned into $2^{\ell+1}$ binary blocks of equal size, then $X$ can be partitioned into $2^{\ell+2}$ binary blocks of equal size. By halving all the cells of $X$ and $Y$ $k = \Theta(\log n)$ times (with $n = |V(G_k)|$), this refinement process ends up with a discrete colouring of these vertices. Since every iteration uses half of the edges between $\mathcal{X}$ and $\mathcal{Y}$ (which are $\Theta(m)$), we get the cost lower bound of $\Omega(m \log n)$ (with $m = |E(G_k)|$).

We now define these gadgets in more detail. For every integer $\ell \geq 1$, we define a gadget $\text{AND}_\ell$, which consists of a graph $G$ together with two out-terminals $a_0, a_1$, and an ordered sequence of $p = 2^\ell$ in-terminals $b_0, \ldots, b_{p-1}$. For $\ell = 1$, the graph $G$ has $V(G) = \{a_0, a_1, b_0, b_1\}$, and $E(G) = \{a_0 b_0, a_1 b_1\}$. For $\ell = 2$, the graph $G$ is identical to the construction of Cai, F"urer and Immerman [8]. (See Fig. 2. The out-terminals $a_0, a_1$ and in-terminals $b_0, \ldots, b_3$ are indicated.) For $\ell \geq 3$, $\text{AND}_\ell$ is obtained by taking one copy $G^*$ of an $\text{AND}_2$-gadget, and two copies $G'$ and $G''$ of an $\text{AND}_{\ell-1}$-gadget, and adding four edges to connect the two pairs of in-terminals of $G^*$ with the pairs of out-terminals of $G'$ and $G''$, respectively. As out-terminals of the resulting gadget we choose the out-terminals of $G^*$. The in-terminal sequence is obtained by concatenating the sequences of in-terminals of $G'$ and $G''$. (See Fig. 3 for an example of $\text{AND}_3$.) For any $\text{AND}_\ell$-gadget $G$ with in-terminals $b_0, \ldots, b_{2^{\ell-1}}$, the in-terminal pairs are pairs $b_{2p}$ and $b_{2p+1}$, for all $p \in \{0, \ldots, 2^{\ell-1} - 1\}$.
The graph $G_k$ is now constructed as follows. Start with vertex sets $X, X', Y$ and $Y$, and edges between them, as defined above. For every $\ell \in \{1, \ldots, k - 1\}$, we add a copy $G$ of an AND$_\ell$-gadget to the graph. Denote the out- and in-terminals of $G$ by $a_0, a_1$ and $b_0, \ldots, b_{2^\ell - 1}$, respectively.

- For $i = 0, 1$ and all relevant $q$: we add edges from $a_i$ to every vertex in $X_{2q+i}^{\ell+1}$.
- For every $i$, we add edges from $b_i$ to every vertex in $Y_i^\ell$.

Finally, we add a starting gadget to the graph, consisting of three vertices $v_0, v_1, v_2$, the edge $v_1v_2$, and edges $\{v_0x_i \mid i \in B_0^1\} \cup \{v_1x_i \mid i \in B_1^1\}$. See Fig. 1 for an example of this construction. (In the figure, we have expanded the terminals of AND$_2$ into edges, for readability. This does not affect the behaviour of the graph.)
Proposition 20  $G_k$ has $O(2^k k)$ vertices and $O(2^k k^2)$ edges.

Proof  An easy induction proof shows that the $\text{AND}_\ell$-gadget has $O(2^\ell)$ vertices and edges. So, all $\text{AND}_\ell$ gadgets together, for $\ell \in \{1, \ldots, k-1\}$, have at most $O(2^k)$ vertices and edges. Therefore, the bounds on the total number of vertices and edges of $G_k$ are dominated by the number of vertices and edges in $G_k[\mathcal{X} \cup \mathcal{Y}]$, which is $k^2 2^k + 1$ and $k^2 2^k$, respectively.

We now state and prove the key property for $\text{AND}_\ell$-gadgets. This requires the following definitions. For a graph $(V, E)$, a coarsest stable partition $\rho$ of $V$ and $E$. If $\psi$ is a partition of a subset $S \subseteq V$, then for short we say that a partition $\rho$ of $V$ refines $\psi$ if it refines $\psi \cup \{V \setminus S\}$. We say that $\rho$ agrees with $\psi$ if $\rho[S] = \psi$. (So if $V \setminus S \neq \emptyset$, one can choose $\rho$ such that it agrees with $\psi$ but does not refine $\psi$.) For two graphs $G$ and $H$, by $G \uplus H$ we denote the graph obtained by taking the disjoint union of $G$ and $H$. We say that a partition $\pi$ of $V$ distinguishes two sets $V_1 \subseteq V$ and $V_2 \subseteq V$ if there is a set $R \in \pi$ with $|R \cap V_1| \neq |R \cap V_2|$. This is often used for the case where $V_1 = N(u)$ and $V_2 = N(v)$ for two vertices $u$ and $v$. If $V_1 = \{x\}$ and $V_2 = \{y\}$, then we also say that $\pi$ distinguishes $x$ from $y$.

Lemma 21  Let $G$ be an $\text{AND}_\ell$-gadget with in-terminals $B = \{b_0, \ldots, b_{2^\ell - 1}\}$ and out-terminals $a_0, a_1$. Let $\psi$ be a partition of $B$ into binary blocks, and let $\rho$ be the coarsest stable partition $\rho$ of $V(G)$ that refines $\psi$. Then $\rho$ agrees with $\psi$. Furthermore, $\rho$ distinguishes $a_0$ from $a_1$ if and only if $\psi$ distinguishes all in-terminal pairs.

Proof  We prove the statement by induction over $\ell$. For $\ell = 1$, the statement is trivial. Now suppose $\ell = 2$. We only consider partitions of $\{b_0, \ldots, b_3\}$ into binary blocks. Because of the automorphisms of this gadget, it follows that it suffices to consider the following four partitions for $\psi$. For all of them, a corresponding partition $\rho$ is given; it can be verified that $\rho$ is the coarsest stable partition of $V(\text{AND}_2)$ that refines $\psi$. (The nonterminal vertices are labeled $c_0, \ldots, c_3$, as shown in Fig. 2.)

\[
\psi = \{b_0, b_1, b_2, b_3\} \quad \Rightarrow \quad \rho = \psi \cup \{c_0, c_1, c_2, c_3, \{a_0, a_1\}\},
\]
\[
\psi = \{b_0, b_1\}, \{b_2, b_3\} \quad \Rightarrow \quad \rho = \psi \cup \{c_0, c_1, c_2, c_3, \{a_0, a_1\}\},
\]
\[
\psi = \{b_0\}, \{b_1\}, \{b_2, b_3\} \quad \Rightarrow \quad \rho = \psi \cup \{c_0, c_2, \{c_1, c_3\}, \{a_0, a_1\}\},
\]
\[
\psi = \{b_0\}, \{b_1\}, \{b_2\}, \{b_3\} \quad \Rightarrow \quad \rho = \psi \cup \{c_0, \{c_1\}, c_2, \{c_3\}, \{a_0\}, \{a_1\}\}.
\]

We see that in all four cases, $\rho$ agrees with $\psi$ on $B$. Furthermore, $\rho$ distinguishes the out-terminals if and only if $\psi$ distinguishes all in-terminal pairs (which is only the case for the last $\psi$).

Now suppose $\ell \geq 3$. Recall that an $\text{AND}_\ell$-gadget $H$ is obtained by taking two copies $G'$ and $G''$ of an $\text{AND}_{\ell-1}$-gadget, and informally, putting a copy $G^*$ of an $\text{AND}_2$-gadget on top of those. Any partition $\psi$ of the in-terminal set $B$ of $H$ into binary blocks corresponds to partitions $\psi'$ and $\psi''$ of the in-terminal sets $B'$ and $B''$ of $G'$ and $G''$, respectively, again into binary blocks. So by induction, we have coarsest stable partitions $\rho'$ and $\rho''$ of $V(G')$ and $V(G'')$ that refine $\psi'$ and $\psi''$ and
agree with them on $B'$ and $B''$, respectively. Together, this yields a partition $\pi$ of $V(G') \cup V(G'')$, which is stable for $G' \sqcup G''$, refines $\psi$, and agrees with $\psi$ on $B$. (To be precise: if $\psi$ is not the unit partition, then we can simply take $\pi = \rho' \cup \rho''$, because $\psi$ is a partition into binary blocks, and thus distinguishes every single in-terminal of $G'$ from every single in-terminal of $G''$. Otherwise, every set in $\pi$ should be the union of the two corresponding sets in $\rho'$ and $\rho''$.) Then $\pi$ gives a partition of the out-terminals of $G'$ and $G''$, which yields a matching partition $\psi^*$ of the in-terminals $B^*$ of $G^*$, again into binary blocks. Applying the induction hypothesis to $G^*$, we obtain a coarsest stable partition $\rho^*$ of $V(G^*)$ that refines and agrees with $\psi^*$. Combining $\pi$ and $\rho^*$ yields a stable partition $\rho$ of the vertices $V(H)$ of the entire gadget.

Applying the induction hypothesis to $G'$ and $G''$ shows that at least one in-terminal pair of $G^*$ is not distinguished by $\psi^*$ if and only if at least one in-terminal pair of $G'$ or $G''$ is not distinguished by $\psi'$ or $\psi''$ respectively. Applying the induction hypothesis to $G^*$ then shows that $\rho$ does not distinguish the out-terminals of $H$ if $\psi$ does not distinguish at least one in-pair of $H$. This then also holds for the coarsest stable partition of $V(H)$ that refines $\rho$.

Finally, let $\psi$ be a partition into binary blocks of the in-terminals $B$ of $H$ that distinguishes every pair, and let $\rho$ be a coarsest stable partition that refines $\psi$. We prove that $\rho$ also distinguishes $a_0$ from $a_1$. By definition, $\rho$ distinguishes any vertex from $B$ from any vertex not in $B$. We conclude that for any two vertices $u, v \in V(H)$, if they have different distance to $B$, then $u \not\approx_{\rho} v$ (Proposition 17). So by Proposition 16, $\rho$ induces stable partitions $\rho'$ and $\pi$ for both $G^*$ and $G' \sqcup G''$, respectively. The graphs $G'$ and $G''$ are components of $G' \sqcup G''$, so we conclude that $\rho$ induces stable partitions $\rho'$ and $\rho''$ for both $G'$ and $G''$, respectively. By induction, it follows that $\rho'$ and $\rho''$ both distinguish the out-terminals of $G'$ and $G''$, respectively. (If this holds for the coarsest stable partition, then it holds for any stable partition.) Then $\psi := \rho[B^*]$ (where $B^*$ denotes again the in-terminal set of $G^*$) distinguishes all in-terminal pairs of $G^*$. So by induction, $\rho$ distinguishes $a_0$ from $a_1$. \hfill $\Box$

The following Corollary follows immediately from Lemma 21.

**Corollary 22** Let $\pi$ be a stable partition for an AND-gadget $G$ such that $\psi = \pi[B]$ is a partition of the in-terminals $B$ into binary blocks, and such that $B$ is $\pi$-closed. If $\pi$ does not distinguish the out-terminals, then at least one in-terminal pair is not distinguished.

**Proof** Since $B$ is $\pi$-closed, $\pi$ refines $\psi = \pi[B]$. Since $\pi$ is stable, it refines the coarsest stable partition $\rho$ of $V(G)$ that refines $\psi$. Now apply Lemma 21. \hfill $\Box$

### 4.3 Cost Lower Bound Proof

Intuitively, at level $\ell$ of the refinement process, the current partition contains all blocks $\mathcal{Y}_{q+1}^{\ell+1}$ of level $\ell + 1$ and for all $0 \leq q < 2^\ell$, either $\mathcal{Y}_{q}^{\ell}$ or the two blocks $\mathcal{Y}_{2q}^{\ell+1}$ and $\mathcal{Y}_{2q+1}^{\ell+1}$. In this situation one can split up the blocks $\mathcal{Y}_{q}^{\ell}$ into blocks $\mathcal{Y}_{2q}^{\ell+1}$ and $\mathcal{Y}_{2q+1}^{\ell+1}$.
\( \mathcal{Y}_{2q+1}^{\ell+1} \) using either refining operation \((\mathcal{X}_{2q}^{\ell+1}, \mathcal{Y}_{q}^{\ell})\) or \((\mathcal{X}_{2q+1}^{\ell+1}, \mathcal{Y}_{q}^{\ell})\). These operations both have cost \(2^{k-(\ell+1)}k^2\), and refining all the \(\mathcal{Y}_{q}^{\ell}\) cells in this way costs \(2^{k-1}k^2\).

Once \(\mathcal{Y}\) is partitioned into binary blocks of level \(\ell+1\), we can partition \(\mathcal{X}\) into blocks of level \(\ell + 2\) (using the AND\(_\ell\)-gadget), and proceed the same way. Since there are \(k\) such refinement levels, we can lower bound the total cost of refining the graph by \(2^{k-1}k^3 = \Omega(m \log n)\) and are done. What remains to show is that applying the refining operations in this specific way is the only way to obtain a stable partition. To formalise this, we introduce a number of partitions of \(\mathcal{X}\).

**Definition 23** For any \(\ell \in \{0, \ldots, k-1\}\), and nonempty set \(Q \subseteq B_\ell\), by \(\pi_{Q,\ell}\) we denote the partition of \(\mathcal{X} \cup \mathcal{Y}\) that contains cells

- \(\mathcal{X}_q^{\ell+1}\) for all \(q \in B_{\ell+1}\),
- \(\mathcal{Y}_q^{\ell}\) for all \(q \in Q\), and both \(\mathcal{Y}_{2q}^{\ell+1}\) and \(\mathcal{Y}_{2q+1}^{\ell+1}\) for all \(q \in B_\ell \setminus Q\).

\(\pi_{Q,\ell}\) denotes the coarsest stable partition for \(G'_k = G_k - [\mathcal{X}, \mathcal{Y}]\) that refines \(\tau_{Q,\ell}\).

We now show that for every \(\ell\) and \(Q\), there is also a stable partition of \(G'_k\) that partitions \(\mathcal{X}\) and \(\mathcal{Y}\) as prescribed by the above definition. In particular, this holds for \(\pi_{Q,\ell}\).

**Lemma 24** For every \(\ell \in \{0, \ldots, k-1\}\) and nonempty set \(Q \subseteq B_\ell\), \(\pi_{Q,\ell}\) agrees with \(\tau_{Q,\ell}\).

**Proof** We design a stable partition \(\rho\) of \(V(G_k) = V(G'_k)\) that is stable on \(G'_k\), and agrees with \(\tau_{Q,\ell}\). So we start with \(\rho = \tau_{Q,\ell}\). For every cell \(\mathcal{X}_q^{\ell+1}\) in \(\tau_{Q,\ell}\), we add the cell \(\mathcal{X}_q^{\ell+1}\) to \(\rho\). For every cell \(\mathcal{Y}_m^{\ell}\) in \(\tau_{Q,\ell}\) (\(\ell \leq m \leq \ell + 1\)), we add the cell \(\mathcal{Y}_m^{\ell}\) to \(\rho\).

Then we add cells \(\{v_0\}, \{v_1\}\) and \(\{v_2\}\).

For every AND\(_p\)-gadget \(G\) of \(G_k\) (with in-terminals adjacent to \(Y\) and out-terminals adjacent to \(X\)), we define a partition \(\psi\) of the in-terminals \(B\) as follows: for \(u, v \in B\), \(u \approx_{\psi} v\) if and only if \(N(u) \cap Y\) is not distinguished from \(N(v) \cap Y\). Note that this yields a partition of \(B\) into binary blocks, and that this distinguishes an in-terminal pair \(b_{2q}, b_{2q+1}\) (which are adjacent to \(Y_{2q}^p\) and \(Y_{2q+1}^p\), respectively, with union \(Y_{q}^{p-1}\)) if and only if \(\ell \geq p\) holds, or both \(\ell = p - 1\) and \(q \notin Q\) hold.

Now we extend \(\rho\) by adding all cells of the coarsest stable partition of the AND\(_p\)-gadget \(G\) that refines \(\psi\). By Lemma 21, this partition distinguishes the out-terminals of \(G\) if and only if \(\ell \geq p\) (since \(Q\) is nonempty). Extending \(\rho\) this way for every AND-gadget yields the final partition \(\rho\) of \(V(G_k)\). By definition, \(\rho\) agrees with \(\tau_{Q,\ell}\). From the construction, the stability condition is easily verified for almost all cells of \(\rho\). Only cells \(\{a_0, a_1\} \in \rho\) consisting of out-terminals of AND\(_p\)-gadgets need to be
considered in more detail. As noted before, such cells only occur when \( p \geq \ell + 1 \).
Then we have for every integer \( q \) that \( X_{2q}^{p+1} \cup X_{2q+1}^{p+1} = X_q^p \subseteq X_{q+1}^{\ell+1} \in \rho \) (for some value \( q' \)). Since \( a_0 \) is adjacent to every \( X_{2q}^{p+1} \) and \( a_1 \) is adjacent to every \( X_{2q+1}^{p+1} \), it follows that \( N(a_0) \) and \( N(a_1) \) are not distinguished by \( \rho \). Therefore, \( \rho \) is stable for \( G_k' \).

Then the coarsest stable partition \( \pi_{Q,\ell} \) that refines \( \tau_{Q,\ell} \) also agrees with \( \tau_{Q,\ell} \). \( \square \)

Since \( \pi_{Q,\ell} \) is stable on \( G_k' \), any effective refining operation (with respect to \( G_k \)) should involve the edges between \( \mathcal{X} \) and \( \mathcal{Y} \). Since \( \pi_{Q,\ell} \) partitions \( \mathcal{X} \) and \( \mathcal{Y} \) as prescribed by \( \tau_{Q,\ell} \), we conclude that any effective elementary refining operation has the form described in the following corollary. Recall that a refining operation \((R, S)\) for a partition \( \pi \) is **elementary** if both \( R \) and \( S \) are classes of \( \pi \), and that by Proposition 19 it suffices to consider elementary refining operations.

**Corollary 25** Let \((R, S)\) be an effective elementary refining operation on \( \pi_{Q,\ell} \). Then for some \( q \in Q \), \( R = X_{2q}^{\ell+1} \) or \( R = X_{2q+1}^{\ell+1} \), and \( S = Y_q^\ell \). The cost of this operation is \( k^2 2^{k-(\ell+1)} \).

This motivates the following definition: for \( q \in Q \), by \( r_q(\pi_{Q,\ell}) \) we denote the partition of \( V(G_k) \) that results from the above refining operation. (Both choices of \( R \) yield the same result.)

**Lemma 26** For every \( \ell \in \{0, \ldots, k-1\} \), nonempty \( Q \subseteq B_{\ell} \) and \( q \in Q \):

\[- r_q(\pi_{Q,\ell}) \leq \pi_{B_{\ell+1},\ell+1}, \text{and} \]
\[- \text{if } Q' = Q \setminus \{q\} \text{ is nonempty, then } r_q(\pi_{Q,\ell}) \leq \pi_{Q',\ell}. \]

**Proof** Choose \( Q' \) and \( \ell' \) satisfying one of the conditions (i.e. \( Q' = B_{\ell+1} \) and \( \ell' = \ell + 1 \), or \( Q' = Q \setminus \{q\} \) and \( \ell' = \ell \)). Then \( \tau_{Q,\ell} \leq \tau_{Q',\ell'} \), so also \( \pi_{Q,\ell} \leq \pi_{Q',\ell'} \) (since \( \pi_{Q',\ell'} \) is also a stable partition that refines \( \tau_{Q,\ell} \)). If we now obtain a partition \( \rho \) from \( \pi_{Q,\ell} \) by splitting up one cell such that the only vertex pairs \( u, v \) with \( u \approx_{\pi_{Q,\ell}} v \) but \( u \not\approx_{\rho} v \) are vertex pairs with \( u \not\approx_{\pi_{Q',\ell'}} v \), then clearly still \( \rho \leq \pi_{Q',\ell'} \) holds. This is exactly how \( r_q(\pi_{Q,\ell}) \) is obtained. \( \square \)

**Lemma 27** Let \( \omega \) be the coarsest stable partition for \( G_k \). For all \( \ell \in \{0, \ldots, k-1\} \) and nonempty \( Q \subseteq B_{\ell} \): \( \pi_{Q,\ell} \leq \omega \).

**Proof** First, we note that by considering the various vertex degrees and using Proposition 17, one can verify that \( \omega \) refines \( \{X, \mathcal{X}, \mathcal{Y}, Y, \{v_0\}, \{v_1\}, \{v_2\}, V_G\} \), where \( V_G \) denotes all vertices in AND-gadgets. In particular, \( V_G \) is \( \omega \)-closed, so \( \omega \) induces a stable partition on \( G(V_G) \) (Proposition 16), and therefore it does so on every AND-gadget of \( G_k \) (which are components of \( G(V_G) \)). Note that for any two different AND\(_\ell\)-gadgets \( H_1 \) and \( H_2 \) of \( G_k \), there exists an integer \( d \) such that \( H_1 \) contains a vertex at distance exactly \( d \) from the \( \omega \)-closed set \( X \cup Y \), but \( H_2 \) does not. This observation can be combined with Proposition 17 to show that if \( u \) and \( v \) are part of different AND-gadgets, then \( u \not\approx_{\omega} v \). Subsequently this yields that for any
AND-gadget of $G_k$ with output terminals $a_0, a_1$, the set $\{a_0, a_1\}$ is $\omega$-closed, and the set of input terminals $B$ of this gadget is $\omega$-closed.

We now prove that $\omega[X]$ is discrete. Suppose that there is an AND-gadget in $G_k$ for which the out-terminals are not distinguished by $\omega$. Then let $\ell$ be the minimum value such that this holds for the AND-$\ell$-gadget $G$ of $G_k$. As observed above, we may apply Corollary 22 to $G$, which shows that there is at least one pair of in-terminals $b_{2q}$ and $b_{2q+1}$ that is not distinguished by $\omega$. By stability, and since $Y$ is $\omega$-closed, this shows that there are vertices $y_i \in Y^{\ell}_{2q}$ and $y_j \in Y^{\ell}_{2q+1}$ in the adjacent binary blocks such that $y_i \approx \omega y_j$. Then, considering the $\omega$-closed subgraph $G_k[X \cup X' \cup Y' \cup Y]$, it easily follows that $x_i \approx \omega x_j$. If $\ell \geq 2$, then $x_i \in X^{\ell}_{2q}$ is adjacent to the out-terminal $a_0$ of the AND-$\ell-1$ gadget of $G_k$, whereas $x_j \in X^{\ell}_{2q+1}$ is adjacent to the other out-terminal $a_1$ of this gadget. By choice of $\ell$, $a_0 \not\approx_\omega a_1$, so since $\{a_0, a_1\}$ is $\omega$-closed, this gives a contradiction with stability. If $\ell = 1$, then we consider the starting gadget: $x_i \in X^1_{0}$ is adjacent to $v_0$, and $x_j \in X^1_{1}$ is adjacent to $v_1$, but $\{v_0\}$ and $\{v_1\}$ are distinct cells of $\omega$, a contradiction with stability.

We conclude that for every AND-gadget, $\omega$ distinguishes the out-terminals. Since every vertex $x_i \in X$ is adjacent to a unique set of such out-terminals, it follows that $\omega[X]$ is discrete. Therefore, for every $q$, $X^k_q$ and $Y^k_q$ are $\omega$-closed. Hence $\omega$ refines $\tau_{Q,\ell}$ for every $Q$ and $\ell$, and thus it refines $\pi_{Q,\ell}$ for every $Q$ and $\ell$.

Proof of Theorem 15 Let $G_k$ be the graph described in Section 4.2, and $\pi_{Q,\ell}$ be the partitions of $V(G_k)$ from Definition 23. By Lemma 27, the coarsest stable partition $\omega$ of $G$ refines all partitions $\pi_{Q,\ell}$. For ease of notation, we define $\pi_{0,\ell} := \pi_{B_{\ell+1},\ell+1}$ for all $\ell < k - 1$. By Corollary 25, any effective elementary refining operation on a partition $\pi_{Q,\ell}$ has cost $2^{k-(\ell+1)k^2}$, and results in $r_q(\pi_{Q,\ell})$ for some $q \in Q$. Denote $Q' = Q \setminus \{q\}$. By Lemma 26, $r_q(\pi_{Q,\ell}) \preceq \pi_{Q',\ell}$. By Proposition 19, to compute the cost($\pi_{Q,\ell}$), it suffices to consider only partitions that can be obtained by elementary refining operations. So we may now apply Proposition 18 to conclude that

$$
\text{cost}(\pi_{Q,\ell}) \geq 2^{k-(\ell+1)k^2} + \min_{q \in Q} \text{cost}(\pi_{Q \setminus \{q\},\ell}).
$$

By induction on $|Q|$ it then follows that $\text{cost}(\pi_{B_{\ell},\ell}) \geq 2^{k-1}k^2 + \text{cost}(\pi_{B_{\ell+1},\ell+1})$ for all $0 \leq \ell \leq k - 1$. Hence, by induction on $\ell$, $\text{cost}(\pi_{B_{\ell,0}}) \geq 2^{k-1}k^3$, which lower bounds cost($\alpha$). By Proposition 20, $n \in O(2^k k)$ and $m \in O(2^k k^2)$, so log $n \in O(k)$. This shows that $\text{cost}(\alpha) \in \Omega((m + n) \log n)$.

4.4 Related lower bounds

In this section, we sketch how our construction also yields lower bounds for two other partitioning problems.

Bisimilarity Bisimilarity is a key concept in concurrency theory and automated verification. A bisimulation is a binary relation defined on the states of a transition system (or between two transition systems). Intuitively, two states are bisimilar if the processes starting in these states look the same. Formally, a transition system is a vertex-labelled directed graph. Let $S = (V, E, \lambda)$, where $(V, E)$ is a directed graph.
and \( \lambda \), a function that assigns a set of properties to each state \( v \in V \). A bisimulation on \( S \) is a relation \( \sim \) on \( V \) satisfying the following three properties for all \( v, w \in V \) such that \( v \sim w \):

(i) \( \lambda(v) = \lambda(w) \);
(ii) for all \( v' \in N^+(v) \) there is a \( w' \in N^+(w) \) such that \( v' \sim w' \);
(iii) for all \( w' \in N^+(w) \) there is a \( v' \in N^+(v) \) such that \( v' \sim w' \).

Not every bisimulation is an equivalence relation, but the reflexive symmetric transitive closure of a bisimulation is still a bisimulation. For convenience, in the following we assume that all bisimulations are equivalence relations. This is justified by the fact that the partition refinement algorithms (see below) that are commonly used to compute bisimulations, and that we study here, represent the relations using partitions of the vertex set and hence implicitly assume that the relations they represent are equivalence relations.

It is not hard to see that on each transition system \( S \) there is a unique coarsest bisimulation, which we call the bisimilarity relation on \( S \). The bisimilarity relation can be defined by letting \( v \) be bisimilar to \( w \) if there is a bisimulation \( \sim \) such that \( v \sim w \); it is then straightforward to verify that bisimilarity is a bisimulation and that all other bisimulations refine it. We remark that the bisimilarity relation on a transition system is precisely what Paige and Tarjan [24] call the coarsest relational partition of the initial partition given by the labelling. Thus the problem of computing the bisimilarity relation of a given transition system is equivalent to the problem of computing the coarsest relational partition considered in [24].

Note the similarity between a bisimulation and a stable colouring of a vertex-coloured digraph, which we may view as a transition system with a labelling \( \lambda \) that maps each vertex to its colour. Condition (i) just says that a bisimulation refines the original colouring, as a stable colouring is supposed to do as well. Conditions (ii) and (iii), which are equivalent under the assumption that a bisimulation be an equivalence relation and hence symmetric, says that if two vertices \( v, w \) are in the same class \( C \) then for every other class \( D \), either both \( v \) and \( w \) have an out-neighbour in \( D \) or neither of them has. Thus instead of refining by the degree in \( D \), we just refine by the Boolean value “degree at least 1”. This immediately implies that the coarsest stable colouring of \( S \) refines the coarsest bisimulation, that is, the bisimilarity relation, on \( S \).

It should be clear from these considerations that the bisimilarity relation on a transition system \( S \) with \( n \) vertices and \( m \) edges can be computed in time \( O((n+m) \log n) \) by a slight modification of the partitioning algorithm for computing the coarsest stable colouring (assuming, of course, that the labels can be computed and compared in constant time) [24].

As for the coarsest stable colouring, we may ask if the bisimilarity relation can be computed in linear time. It turns out that our lower bound for colour refinement implies a lower bound for bisimilarity. Again, we consider the class of partition refinement algorithms. As the partition refinement algorithms for colour refinement, partition refinement algorithms for bisimilarity maintain a partition of the set of vertices of the given transition system, and they iteratively refine it using refining operations until a bisimulation is reached. In each refining operation, such an algorithm chooses a union of current partition cells as refining set \( R \), and chooses another
(possibly overlapping) union of partition cells $S$. Cells in $S$ are split up according to the out-neighbourhoods of the vertices in the cells in $R$. That is, two vertices $v, w$ currently in the same cell in $S$ remain in the same cell after the refinement step if and only if for all cells $C$ of the partition, with $C \subseteq R$, it holds that

$$N^+(v) \cap C \neq \emptyset \iff N^+(v') \cap C \neq \emptyset.$$ 

Recall that $N^+(v)$ denotes the set of out-neighbours of a vertex $v$ in a directed graph (or transition system). The cost $\text{bcost}(R, S)$ of such a refinement relation $(R, S)$ is the number of edges from $S$ to $R$. Again, the sum of the costs of all refinement operations is a reasonable lower bound for the running time of a partition refinement algorithm. The cost $\text{bcost}(\alpha)$ of a partition $\alpha$ of the vertex set is then defined as the minimum cost of a sequence of refinement operations that transforms $\alpha$ to the coarsest bisimulation refining it (see Definition 14).

**Theorem 28** For every integer $k \geq 2$, there is a transition system $S_k$ with $n \in O(2^kk)$ vertices and $m \in O(2^kk^2)$ edges and constant labelling function, such that $\text{bcost}(\alpha) \in \Omega((m + n) \log n)$, where $\alpha$ is the unit partition of $V(S_k)$.

**sketch** The proof is essentially the same as the proof of Theorem 15. The transition system $S_k$ is a directed version of the the graph $G_k$. Fig. 4 illustrates the direction of the edges. All vertices get the same label.

It is not hard to show that the bisimilarity classes of $S_k$ are exactly the same as the colour classes of $G_k$ in the coarsest stable colouring and that essentially the refinement steps do the same on $G_k$ and $S_k$. Thus the lower-bound proof carries over.

![Fig. 4](image-url) The transitions system $S_3$ corresponding to the graph $G_3$ of Fig. 1
Equivalence in 2-Variable Logic  It is a well-known fact (due to Immerman and Lander [17]) that colour refinement assigns the same colour to two vertices of a graph if and only if the vertices satisfy the same formulas of the logic $\mathcal{C}^2$, two-variable first-order logic with counting.

Two variable first-order logic $L^2$ is the fragment of first order logic consisting of all formulas built with just two variables. For example, the following $L^2$-formula $\phi(x)$ in the language of directed graphs says that from vertex $x$ one can reach a sink (a vertex of out-degree 0) in four steps:

$$\phi(x) := \exists y(E(x,y) \land \exists x(E(y,x) \land \exists y(E(x,y) \land \forall y \neg E(x,y))))).$$

Two variable first-order logic with counting $\mathcal{C}^2$ is the extension of $L^2$ by counting quantifiers of the form $\exists^{\geq i}x$, for all $i \geq 1$. For example, the following $\mathcal{C}^2$-formula $\psi(x)$ in the language of directed graphs says that from vertex $x$ one can reach a vertex of out-degree at least 10 in four steps:

$$\psi(x) := \exists y(E(x,y) \land \exists x(E(y,x) \land \exists y(E(x,y) \land \exists^{\geq 10} x E(x,y))))).$$

This formula is not equivalent to any formula of $L^2$. Two-variable logics, and more generally finite variable logics, have been studied extensively in finite model theory (see, for example, [12, 14, 16, 19]).

We call two vertices of a graph $L^2$-equivalent ($\mathcal{C}^2$-equivalent) if they satisfy the same formulas of the logic $L^2$ ($\mathcal{C}^2$, respectively). Now Immerman and Lander’s theorem states that for all graphs $G$ (possible coloured and/or directed) and all vertices $v, w \in V(G)$, the vertices $v$ and $w$ have the same colour in the coarsest bi-stable colouring of $G$ if and only if they are $\mathcal{C}^2$-equivalent. (Recall that bi-stable was defined in Section 3.4.) In particular, this implies that the $\mathcal{C}^2$-equivalence classes of a graph can be computed in time $O((n + m) \log n)$, but not better (by a partition-refinement algorithm).

On plain undirected graphs, the logic $L^2$ is extremely weak. However, on coloured and/or directed graphs, the logic is quite interesting. The $L^2$-equivalence relation refines the bisimilarity relation. It is well known that the $L^2$-equivalence relation can be computed in time $O((n + m) \log n)$ by a variant of the colour refinement algorithm. Our lower bounds can be extended to show that it cannot be computed faster by a partition-refinement algorithm.

An Open Problem  The key idea of the $O((n + m) \log n)$ partitioning algorithms is Hopcroft’s idea of processing the smaller half. Hopcroft originally proposed this idea for the minimisation of deterministic finite automata. The algorithm proceeds by identifying equivalent states and then collapsing each equivalence class to a single new state. The partitioning problem (computing classes of equivalent states) is actually just the bisimilarity problem for finite automata, which may be viewed as edge-labelled transition systems.

However, for DFA-minimisation we only need to compute the bisimilarity relation for deterministic finite automata, that is, transition systems where each state has
exactly one outgoing edge of each edge label. The systems in our lower bound proof are highly nondeterministic. Thus our lower bounds do not apply.

It remains a very interesting open problem whether similar lower bounds can be proved for DFA-minimisation, or whether DFA-minimisation is possible in linear time. Paige, Tarjan, and Bonic [25] proved that this is possible for DFAs with a single-letter alphabets. To the best of our knowledge, the only known result in this direction is a family of examples due to Berstel and Carton [7] (also see [6, 10]) showing that the $O(n \log n)$ bound for Hopcroft’s original algorithm is tight.

Acknowledgments We thank Christof Löding for discussions and references regarding the lower bounds for bisimilarity and DFA-minimisation.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

1. Babai, L.: Moderately Exponential Bound for Graph Isomorphism. In: Proc. FCT’81, pp. 34–50. Springer (1981)
2. Babai, L., Erdös, P., Selkow, S.: Random graph isomorphism. SIAM J. Comput. 9, 628–635 (1980)
3. Babai, L., Luks, E.: Canonical Labeling of Graphs. In: Proc. STOC’83, pp. 171–183 (1983)
4. Berkholz, C.: Lower bounds for heuristic algorithms. Ph.D. thesis, RWTH Aachen University. http://publications.rwth-aachen.de/record/462250 (2014)
5. Berkholz, C., Nordström, J.: Near-Optimal Lower Bounds on Quantifier Depth and Weisfeiler-Leman Refinement Steps. In: Proceedings of the 31St Annual ACM/IEEE Symposium on Logic in Computer Science (LICS’16) (2016). To appear
6. Berstel, J., Boasson, L., Carton, O.: Continuant polynomials and worst-case behavior of Hopcroft’s minimization algorithm. Theor. Comput. Sci. 410(30-32), 2811–2822 (2009)
7. Berstel, J., Carton, O.: On the Complexity of Hopcroft’s State Minimization Algorithm. In: Proc. CIAA'04, pp. 35–44 (2005)
8. Cai, J., Fürer, M., Immerman, N.: An optimal lower bound on the number of variables for graph identifications. Combinatorica 12(4), 389–410 (1992)
9. Cardon, A., Crochemore, M.: Partitioning a graph in $O(|A| \log_2 |V|)$. Theor. Comput. Sci. 19(1), 85–98 (1982)
10. Castiglione, G., Restivo, A., Sciortino, M.: Circular sturmian words and Hopcroft’s algorithm. Theor. Comput. Sci. 410(43), 4372–4381 (2009)
11. Darga, P., Lifftiton, M., Sakallah, K., Markov, I.: Exploiting Structure in Symmetry Detection for CNF. In: Proc. DAG’04, pp. 530–534 (2004)
12. Ebbinghaus, H.D., Flum, J. Finite Model Theory, 2nd edn. Springer, Verlag (1999)
13. Fredman, M., Tarjan, R.: Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM 34, 596–615 (1987)
14. Grädel, E., Kolaitis, P., Libkin, L., Marx, M., Spencer, J., Vardi, M., Venema, Y., Weinstein, S.: Finite model theory and its applications. springer, verlag (2007)
15. Hopcroft, J.: An N Log N Algorithm for Minimizing States in a Finite Automaton. In: Z. Kohavi, A. Paz (eds.) Theory of Machines and Computations, pp. 189–196. Academic Press (1971)
16. Immerman, N.: Descriptive complexity springer verlag (1999)
17. Immerman, N., Lander, E.: Describing Graphs: a First-Order Approach to Graph Canonization. In: A. Selman Complexity theory retrospective pp. 59-81 Springer-Verlag (1990)
18. Junttila, T., Kaski, P.: Engineering an Efficient Canonical Labeling Tool for Large and Sparse Graphs. In: Proc. ALENEX’07, pp. 135–149 (2007)
19. Libkin, L.: Elements of finite model theory. Springer Verlag (2004)
20. McKay, B.: Practical graph isomorphism. Congressus Numerantium 30, 45–87 (1981)
21. McKay, B.: Nauty User’s Guide (Version 2.4). Computer Science Dept., Australian National University (2007)
22. McKay, B., Piperno, A.: Practical graph isomorphism, II. J. Symb. Comput. 60, 94–112 (2014)
23. Miyazaki, T.: The complexity of McKay’s canonical labeling algorithm. Groups and Computation II 28, 239–256 (1997)
24. Paige, R., Tarjan, R.: Three partition refinement algorithms. SIAM J. Comput. 16(6), 973–989 (1987)
25. Paige, R., Tarjan, R., Bonic, R.: A linear time solution to the single function coarsest partition problem. Theor. Comput. Sci. 40, 67–84 (1985)