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

by C. Berkholz, P. Bonsma, M. Grohe

Simone Boscaratto

Supervisor: Alberto Policriti

University of Udine

5th December 2025

Outline

Aim

Goal

- To define a *canonical* colour refinement procedure.
- To optimise it as to get $O((m+n)\log n)$ time complexity.
- To show that no faster algorithm of this kind is possible.

Why does it matter?

- Colour refinement is able to distinguish *almost all* pairs of non-isomorphic graphs (Babai, Erdös, Selkov, 1980).
- The assumptions on the algorithms this bound applies are very humble and shared by all the main tools for practical graph isomorphism, e.g. `nauty` (McKay, 1981) and `traces` (McKay, Piperno, 2014).

Aim

Goal

- To define a *canonical* colour refinement procedure.
- To optimise it as to get $O((m+n)\log n)$ time complexity.
- To show that no faster algorithm of this kind is possible.

Why does it matter?

- Colour refinement is able to distinguish *almost all* pairs of non-isomorphic graphs (Babai, Erdös, Selkov, 1980).
- The assumptions on the algorithms this bound applies are very humble and shared by all the main tools for practical graph isomorphism, e.g. `nauty` (McKay, 1981) and `traces` (McKay, Piperno, 2014).

Aim

Goal

- To define a *canonical* colour refinement procedure.
- To optimise it as to get $O((m+n)\log n)$ time complexity.
- To show that no faster algorithm of this kind is possible.

Why does it matter?

- Colour refinement is able to distinguish *almost all* pairs of non-isomorphic graphs (Babai, Erdös, Selkov, 1980).
- The assumptions on the algorithms this bound applies are very humble and shared by all the main tools for practical graph isomorphism, e.g. `nauty` (McKay, 1981) and `traces` (McKay, Piperno, 2014).

Aim

Goal

- To define a *canonical* colour refinement procedure.
- To optimise it as to get $O((m+n)\log n)$ time complexity.
- To show that no faster algorithm of this kind is possible.

Why does it matter?

- Colour refinement is able to distinguish *almost all* pairs of non-isomorphic graphs (Babai, Erdös, Selkov, 1980).
- The assumptions on the algorithms this bound applies are very humble and shared by all the main tools for practical graph isomorphism, e.g. `nauty` (McKay, 1981) and `traces` (McKay, Piperno, 2014).

Aim

Goal

- To define a *canonical* colour refinement procedure.
- To optimise it as to get $O((m+n)\log n)$ time complexity.
- To show that no faster algorithm of this kind is possible.

Why does it matter?

- Colour refinement is able to distinguish *almost all* pairs of non-isomorphic graphs (Babai, Erdös, Selkov, 1980).
- The assumptions on the algorithms this bound applies are very humble and shared by all the main tools for practical graph isomorphism, e.g. `nauty` (McKay, 1981) and `traces` (McKay, Piperno, 2014).

Core idea

Hopcroft's idea

- This algorithm is based on idea of "process the smallest half" (Hopcroft, 1970), originally thought to tackle automata minimisation.
- (Cardon, Crochemore, 1982) and (Paige, Tarjan, 1987) already implemented algorithms with same running time, but **do not** provide a *canonical* stable partition.

Algorithm's core

1. Start with unit partition (all vertices have the same colour).
2. Smartly choose a *refining set* R and a union of partition cells S.
3. Use them to refine the previous partition until a stable colouring is reached.

Core idea

Hopcroft's idea

- This algorithm is based on idea of "process the smallest half" (Hopcroft, 1970), originally thought to tackle automata minimisation.
- (Cardon, Crochemore, 1982) and (Paige, Tarjan, 1987) already implemented algorithms with same running time, but **do not** provide a *canonical* stable partition.

Algorithm's core

1. Start with unit partition (all vertices have the same colour).
2. Smartly choose a *refining set* R and a union of partition cells S.
3. Use them to refine the previous partition until a stable colouring is reached.

Core idea

Hopcroft's idea

- This algorithm is based on idea of "process the smallest half" (Hopcroft, 1970), originally thought to tackle automata minimisation.
- (Cardon, Crochemore, 1982) and (Paige, Tarjan, 1987) already implemented algorithms with same running time, but **do not** provide a *canonical* stable partition.

Algorithm's core

1. Start with unit partition (all vertices have the same colour).
2. Smartly choose a *refining set* R and a union of partition cells S.
3. Use them to refine the previous partition until a stable colouring is reached.

Partition

Given a simple graph $G = (V, E)$ and $v \in V$, denote its *neighbourhood* and *degree* by $N(v) := \{u \in V \mid (u, v) \in E\}$ and $d(v) := |N(v)|$, respectively.

Directed case

One can adapt every definition to the directed case by considering out-/in-neighbourhood $N^+(v)/N^-(v)$ and out-/in-degree $d^+(v)/d^-(v)$.

Definition (Partition)

A *partition* π of V is a set of non-empty, pairwise-disjoint subsets of V whose union is V. Each $S \in \pi$ is called a *cell* of π.
The *order* of π is $|\pi|$: π is *discrete* if $|\pi| = |V|$, *unit* if $|\pi| = 1$.
Equivalence relation: $u \approx_\pi v \overset{\text{def}}{\iff} (\exists S \in \pi)(u, v \in S)$.

$V' \subseteq V$ is π-*closed* if it is the union of cells of π.

Partition

Given a simple graph $G = (V, E)$ and $v \in V$, denote its *neighbourhood* and *degree* by $N(v) := \{u \in V \mid (u, v) \in E\}$ and $d(v) := |N(v)|$, respectively.

Directed case

One can adapt every definition to the directed case by considering out-/in-neighbourhood $N^+(v)/N^-(v)$ and out-/in-degree $d^+(v)/d^-(v)$.

Definition (Partition)

A *partition* π of V is a set of non-empty, pairwise-disjoint subsets of V whose union is V. Each $S \in \pi$ is called a *cell* of π.
The *order* of π is $|\pi|$: π is *discrete* if $|\pi| = |V|$, *unit* if $|\pi| = 1$.
Equivalence relation: $u \approx_\pi v \overset{\text{def}}{\iff} (\exists S \in \pi)(u, v \in S)$.

$V' \subseteq V$ is π-*closed* if it is the union of cells of π.

Partition

Given a simple graph $G = (V, E)$ and $v \in V$, denote its *neighbourhood* and *degree* by $N(v) := \{u \in V \mid (u, v) \in E\}$ and $d(v) := |N(v)|$, respectively.

Directed case

One can adapt every definition to the directed case by considering out-/in-neighbourhood $N^+(v)/N^-(v)$ and out-/in-degree $d^+(v)/d^-(v)$.

Definition (Partition)

A *partition* π of V is a set of non-empty, pairwise-disjoint subsets of V whose union is V. Each $S \in \pi$ is called a *cell* of π.
The *order* of π is $|\pi|$: π is *discrete* if $|\pi| = |V|$, *unit* if $|\pi| = 1$.
Equivalence relation: $u \approx_\pi v \overset{\text{def}}{\iff} (\exists S \in \pi)(u, v \in S)$.

$V' \subseteq V$ is π-*closed* if it is the union of cells of π.

Partition

Given a simple graph $G = (V, E)$ and $v \in V$, denote its *neighbourhood* and *degree* by $N(v) := \{u \in V \mid (u, v) \in E\}$ and $d(v) := |N(v)|$, respectively.

Directed case

One can adapt every definition to the directed case by considering out-/in-neighbourhood $N^+(v)/N^-(v)$ and out-/in-degree $d^+(v)/d^-(v)$.

Definition (Partition)

A *partition* π of V is a set of non-empty, pairwise-disjoint subsets of V whose union is V. Each $S \in \pi$ is called a *cell* of π.
The *order* of π is $|\pi|$: π is *discrete* if $|\pi| = |V|$, *unit* if $|\pi| = 1$.
Equivalence relation: $u \approx_\pi v \overset{\text{def}}{\iff} (\exists S \in \pi)(u, v \in S)$.

$V' \subseteq V$ is π-*closed* if it is the union of cells of π.

Stability and refinement

Definition (Stability)

A partition π of V is *stable* for the undirected simple graph G iff

$$(\forall u, v \in V)(\forall R \in \pi)\Big(u \approx_\pi v \Rightarrow |N(u) \cap R| = |N(v) \cap R|\Big).$$

Definition (Refinement)

A partition ρ of V *refines* a partition π if $(\forall u, v \in V)(u \approx_\rho v \Rightarrow u \approx_\pi v)$: in this case, $\pi \preceq \rho$ (π is *coarser* than ρ; ρ is *finer* than π).

Stability and refinement

Definition (Stability)

A partition π of V is *stable* for the directed simple graph G iff

$$(\forall u, v \in V)(\forall R \in \pi)\Big(u \approx_\pi v \Rightarrow |N^+(u) \cap R| = |N^+(v) \cap R|\Big).$$

Definition (Refinement)

A partition ρ of V *refines* a partition π if $(\forall u, v \in V)(u \approx_\rho v \Rightarrow u \approx_\pi v)$: in this case, $\pi \preceq \rho$ (π is *coarser* than ρ; ρ is *finer* than π).

Stability and refinement

Definition (Stability)

A partition π of V is *stable* for the directed simple graph G iff

$$(\forall u, v \in V)(\forall R \in \pi)\Big(u \approx_\pi v \Rightarrow |N^+(u) \cap R| = |N^+(v) \cap R|\Big).$$

Definition (Refinement)

A partition ρ of V *refines* a partition π if $(\forall u, v \in V)(u \approx_\rho v \Rightarrow u \approx_\pi v)$: in this case, $\pi \preceq \rho$ (π is *coarser* than ρ; ρ is *finer* than π).

Refining operation (R, S)

Definition (Refining operation (R, S))

Given π, π' partitions of V, $R, S \subseteq V$ π-closed, π' is obtained from π by a *refining operation* (R, S) if

- $(\forall S' \in \pi)(S' \cap S = \emptyset \Rightarrow S' \in \pi')$, and
- $(\forall u, v \in S)\big(u \approx_{\pi'} v \Leftrightarrow u \approx_{\pi} v \wedge (\forall R' \in \pi)(R' \subseteq R \Rightarrow |N^+(u) \cap R'| = |N^+(v) \cap R'|)\big)$.

Meaning

- Every π-cell **not** included in S is not touched by the operation, i.e. just the π-cells of S are refined.
- The π'-cells that are **not** also π-cells are given by the nodes that have the same number of neighbours in each π-cell contained in R.

Two propositions about partitions and refinements

Proposition

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

Proposition

Let $G = (V, E)$ be a graph. For every partition π of V, there is a unique coarsest stable partition ρ that refines π.

Colours, colourings, colouring methods

Definition (Colouring)

A *colouring* of G is $\alpha\colon V(G) \to \mathbb{Z}$. It is a *$k$-colouring* if $\alpha(V(G)) \subseteq \{1, \dots, k\}$. The *colour class* i is $C_i^\alpha = \{v \in V(G) \mid \alpha(v) = i\}$; non-empty colour classes define a partition π_α of $V(G)$.

Definition (Colouring method)

A *colouring method* produces a colouring β from a k-colouring α.

We are interested in colourings that are kept by isomorphisms.

Definition (Canonicity)

A *canonical colouring method* produces colourings β, β' that are preserved by a (α, α')-colour-preserving isomorphism between two isomorphic graphs G, G' with initial colouring α, α', resp.

Colours, colourings, colouring methods

Definition (Colouring)

A *colouring* of G is $\alpha\colon V(G) \to \mathbb{Z}$. It is a *$k$-colouring* if $\alpha(V(G)) \subseteq \{1, \dots, k\}$. The *colour class* i is $C_i^\alpha = \{v \in V(G) \mid \alpha(v) = i\}$; non-empty colour classes define a partition π_α of $V(G)$.

Definition (Colouring method)

A *colouring method* produces a colouring β from a k-colouring α.

We are interested in colourings that are kept by isomorphisms.

Definition (Canonicity)

A *canonical colouring method* produces colourings β, β' that are preserved by a (α, α')-colour-preserving isomorphism between two isomorphic graphs G, G' with initial colouring α, α', resp.

Colours, colourings, colouring methods

Definition (Colouring)

A *colouring* of G is $\alpha\colon V(G) \to \mathbb{Z}$. It is a k-*colouring* if $\alpha(V(G)) \subseteq \{1, \dots, k\}$. The *colour class* i is $C_i^\alpha = \{v \in V(G) \mid \alpha(v) = i\}$; non-empty colour classes define a partition π_α of $V(G)$.

Definition (Colouring method)

A *colouring method* produces a colouring β from a k-colouring α.

We are interested in colourings that are kept by isomorphisms.

Definition (Canonicity)

A *canonical colouring method* produces colourings β, β' that are preserved by a (α, α')-colour-preserving isomorphism between two isomorphic graphs G, G' with initial colouring α, α', resp.

Colours, colourings, colouring methods

Definition (Colouring)

A *colouring* of G is $\alpha\colon V(G) \to \mathbb{Z}$. It is a k-*colouring* if $\alpha(V(G)) \subseteq \{1, \dots, k\}$. The *colour class* i is $C_i^\alpha = \{v \in V(G) \mid \alpha(v) = i\}$; non-empty colour classes define a partition π_α of $V(G)$.

Definition (Colouring method)

A *colouring method* produces a colouring β from a k-colouring α.

We are interested in colourings that are kept by isomorphisms.

Definition (Canonicity)

A *canonical colouring method* produces colourings β, β' that are preserved by a (α, α')-colour-preserving isomorphism between two isomorphic graphs G, G' with initial colouring α, α', resp.

The algorithm: input & output

Input

- A n-vertices digraph G.
- A surjective (initial) ℓ-colouring α of $V(G)$.
- A *sufficient* refining colour set $S \subseteq \{1, \ldots, \ell\}$.

Sufficient refining colour set

$$(\forall C_i^\alpha)(\forall u, v \in C_i^\alpha)\Big((\exists C_j^\alpha)(|N^+(u) \cap C_j^\alpha| \neq |N^+(v) \cap C_j^\alpha|)$$
$$\Rightarrow (\exists j' \in S)(|N^+(u) \cap C_{j'}^\alpha| \neq |N^+(v) \cap C_{j'}^\alpha|)\Big)$$

This is necessary to get actual refinements through the algorithm.

Output

$\rightarrow$ A surjective canonical k-colouring β s.t. π_β is the coarsest stable partition of $V(G)$ refining π_α.

The algorithm: input & output

Input

- A n-vertices digraph G.
- A surjective (initial) ℓ-colouring α of $V(G)$.
- A *sufficient* refining colour set $S \subseteq \{1, \ldots, \ell\}$.

Sufficient refining colour set

$$(\forall C_i^\alpha)(\forall u, v \in C_i^\alpha)\Big((\exists C_j^\alpha)(|N^+(u) \cap C_j^\alpha| \neq |N^+(v) \cap C_j^\alpha|) \Rightarrow (\exists j' \in S)(|N^+(u) \cap C_{j'}^\alpha| \neq |N^+(v) \cap C_{j'}^\alpha|)\Big)$$

This is necessary to get actual refinements through the algorithm.

Output

$\rightarrow$ A surjective canonical k-colouring β s.t. π_β is the coarsest stable partition of $V(G)$ refining π_α.

The algorithm: input & output

Input

- A n-vertices digraph G.
- A surjective (initial) ℓ-colouring α of $V(G)$.
- A *sufficient* refining colour set $S \subseteq \{1, \dots, \ell\}$.

Sufficient refining colour set

$$(\forall C_i^\alpha)(\forall u, v \in C_i^\alpha)\Big((\exists C_j^\alpha)(|N^+(u) \cap C_j^\alpha| \neq |N^+(v) \cap C_j^\alpha|) \\ \Rightarrow (\exists j' \in S)(|N^+(u) \cap C_{j'}^\alpha| \neq |N^+(v) \cap C_{j'}^\alpha|)\Big)$$

This is necessary to get actual refinements through the algorithm.

Output

$\rightarrow$ A surjective canonical k-colouring β s.t. π_β is the coarsest stable partition of $V(G)$ refining π_α.

Pipeline

1. Take the partition $C_1^\alpha, \ldots, C_\ell^\alpha$.
2. Define $S_{\text{refine}} := \text{stack}(S)$ (highest colour $\rightarrow$ top of stack).
3. At each step, given $(C_1, \ldots, C_k)$ until $S_{\texttt{refine}} = \emptyset$:
   1. Let $r := \text{pop}(S_{\text{refine}})$ (*refining colour*);
   2. Apply the refining operation (C_r, V): every colour class $C_s \in \{C_1, \ldots, C_k\}$ is split according to its nodes' *colour degree* $d_r^+(v) := |N^+(v) \cap C_r|$ w.r.t. colour r;
   3. New colours generated by splitting each C_s will be s and $k+1, \ldots, k+d-1$ (d: number of different colour degrees in C_s);
   4. New colours are put onto S_{refine} **unless** C_s has already been used as refining set: in this case, a colour b which class C_b has maximum size is **not** put onto S_{refine} (Hopcroft's trick).

Correctness

To prove the algorithm's correctness, the following are needed:

- At the end of each iteration:
  - $C_1, \ldots, C_k$ is an actual partition of $V(G)$; (trivial)
  - S_{refine} is a sufficient refining colour set for the corresponding k-colouring. (tedious)
- At the end of the algorithm:
  - The resulting partition π_β is the coarsest stable partition of $V(G)$ refining π_α; (trivial)
  - The partition π_β is canonical. (by induction)

Correctness

To prove the algorithm's correctness, the following are needed:

- At the end of each iteration:
  - $C_1, \ldots, C_k$ is an actual partition of $V(G)$; (trivial)
  - S_{refine} is a sufficient refining colour set for the corresponding k-colouring. (tedious)
- At the end of the algorithm:
  - The resulting partition π_β is the coarsest stable partition of $V(G)$ refining π_α; (trivial)
  - The partition π_β is canonical. (by induction)

Correctness

To prove the algorithm's correctness, the following are needed:

- At the end of each iteration:
  - $C_1, \ldots, C_k$ is an actual partition of $V(G)$; (trivial)
  - S_{refine} is a sufficient refining colour set for the corresponding k-colouring. (tedious)
- At the end of the algorithm:
  - The resulting partition π_β is the coarsest stable partition of $V(G)$ refining π_α; (trivial)
  - The partition π_β is canonical. (by induction)

Correctness

To prove the algorithm's correctness, the following are needed:

- At the end of each iteration:
  - $C_1, \ldots, C_k$ is an actual partition of $V(G)$; (trivial)
  - S_{refine} is a sufficient refining colour set for the corresponding k-colouring. (tedious)
- At the end of the algorithm:
  - The resulting partition π_β is the coarsest stable partition of $V(G)$ refining π_α; (trivial)
  - The partition π_β is canonical. (by induction)

Correctness

To prove the algorithm's correctness, the following are needed:

- At the end of each iteration:
  - $C_1, \ldots, C_k$ is an actual partition of $V(G)$; (trivial)
  - S_{refine} is a sufficient refining colour set for the corresponding k-colouring. (tedious)
- At the end of the algorithm:
  - The resulting partition π_β is the coarsest stable partition of $V(G)$ refining π_α; (trivial)
  - The partition π_β is canonical. (by induction)

How can we do better?

What is missing

The described algorithm has nested for-loops (bad time complexity) and no description of the data structures used (but for the stack S_{refine}).

Improvements – data structures

- **Colour classes.** Doubly-linked lists $C[i]$, with $i \in \{1, \ldots, n\}$.
- **Colour degrees.** Array $\text{cdeg}[v]$, with $v \in \{1, \ldots, n\}$.
  - Maximum colour degree for each colour: array maxcdeg.
  - Colours having a vertex with $\text{cdeg}[w] \geq 1$: list $\text{Color}_{\text{adj}}$.
  - Vertices with $\text{cdeg}[w] \geq 1$ and colour i: list $A[i]$.
- **Colour classes split.** New list $\text{Color}_{\text{split}} \subseteq \text{Color}_{\text{adj}}$ containing actually split-up colours.

How can we do better?

What is missing

The described algorithm has nested for-loops (bad time complexity) and no description of the data structures used (but for the stack S_{refine}).

Improvements – data structures

- **Colour classes.** Doubly-linked lists $C[i]$, with $i \in \{1, \ldots, n\}$.
- **Colour degrees.** Array $\text{cdeg}[v]$, with $v \in \{1, \ldots, n\}$.
  - Maximum colour degree for each colour: array maxcdeg.
  - Colours having a vertex with $\text{cdeg}[w] \geq 1$: list $\text{Color}_{\text{adj}}$.
  - Vertices with $\text{cdeg}[w] \geq 1$ and colour i: list $A[i]$.
- **Colour classes split.** New list $\text{Color}_{\text{split}} \subseteq \text{Color}_{\text{adj}}$ containing actually split-up colours.

Cost analysis

Lemma

The algorithm can be implemented in such a way that the refining operation (R, S) takes time $O(|R| + D^-(R) + k \log k)$, where $D^-(R) = \sum_{v \in R} d^-(v)$, k number of newly introduced colours; initialisation step takes time $O(n)$.

Lemma

The algorithm has an implementation with complexity $O((m+n)\log n)$.

Proof.

$$\sum_R |R| + D^-(R) + \sum_i k_i \log k_i = (n+m)\log n + n \log n$$

$$\to T(n,m) \in O(n) + O((n+m)\log n) + O(n \log n) = O((m+n)\log n).$$

□

Cost analysis

Lemma

The algorithm can be implemented in such a way that the refining operation (R, S) takes time $O(|R| + D^-(R) + k \log k)$, where $D^-(R) = \sum_{v \in R} d^-(v)$, k number of newly introduced colours; initialisation step takes time $O(n)$.

Lemma

The algorithm has an implementation with complexity $O((m+n) \log n)$.

Proof.

$$\sum_R |R| + D^-(R) + \sum_i k_i \log k_i = (n+m) \log n + n \log n$$

$$\rightarrow T(n,m) \in O(n) + O((n+m) \log n) + O(n \log n) = O((m+n) \log n).$$

Main result

Theorem

For any digraph G with n nodes, m edges and a surjective ℓ-colouring α, a canonical surjective k-colouring β such that π_β is the coarsest stable partition refining π_α can be computed in time $O((m+n)\log n)$

Variants

- **Queue vs. stack.** $O((m+n)\log n)$ time complexity using either.
- **Iterative refinement.** *Individualisation* of a single node gives $O((m+n)\log n)$ time complexity; individualising each node on a different branch leads to worst-case non-polynomial time complexity.
- **Undirected case.** $O((m+n)\log n)$ time complexity.
- **Edge coloured digraph.** $O((m+n)\log(m+n))$ time complexity.
- **Bi-stable colouring of a digraph.** $O((m+n)\log n)$ time complexity.

Variants

- **Queue vs. stack.** $O((m+n)\log n)$ time complexity using either.
- **Iterative refinement.** *Individualisation* of a single node gives $O((m+n)\log n)$ time complexity; individualising each node on a different branch leads to worst-case non-polynomial time complexity.
- **Undirected case.** $O((m+n)\log n)$ time complexity.
- **Edge coloured digraph.** $O((m+n)\log(m+n))$ time complexity.
- **Bi-stable colouring of a digraph.** $O((m+n)\log n)$ time complexity.

Variants

- **Queue vs. stack.** $O((m+n)\log n)$ time complexity using either.
- **Iterative refinement.** *Individualisation* of a single node gives $O((m+n)\log n)$ time complexity; individualising each node on a different branch leads to worst-case non-polynomial time complexity.
- **Undirected case.** $O((m+n)\log n)$ time complexity.
- **Edge coloured digraph.** $O((m+n)\log(m+n))$ time complexity.
- **Bi-stable colouring of a digraph.** $O((m+n)\log n)$ time complexity.

Variants

- **Queue vs. stack.** $O((m+n)\log n)$ time complexity using either.
- **Iterative refinement.** *Individualisation* of a single node gives $O((m+n)\log n)$ time complexity; individualising each node on a different branch leads to worst-case non-polynomial time complexity.
- **Undirected case.** $O((m+n)\log n)$ time complexity.
- **Edge coloured digraph.** $O((m+n)\log(m+n))$ time complexity.
- **Bi-stable colouring of a digraph.** $O((m+n)\log n)$ time complexity.

Variants

- **Queue vs. stack.** $O((m+n)\log n)$ time complexity using either.
- **Iterative refinement.** *Individualisation* of a single node gives $O((m+n)\log n)$ time complexity; individualising each node on a different branch leads to worst-case non-polynomial time complexity.
- **Undirected case.** $O((m+n)\log n)$ time complexity.
- **Edge coloured digraph.** $O((m+n)\log(m+n))$ time complexity.
- **Bi-stable colouring of a digraph.** $O((m+n)\log n)$ time complexity.

Variants

- **Queue vs. stack.** $O((m+n)\log n)$ time complexity using either.
- **Iterative refinement.** *Individualisation* of a single node gives $O((m+n)\log n)$ time complexity; individualising each node on a different branch leads to worst-case non-polynomial time complexity.
- **Undirected case.** $O((m+n)\log n)$ time complexity.
- **Edge coloured digraph.** $O((m+n)\log(m+n))$ time complexity.
- **Bi-stable colouring of a digraph.** $O((m+n)\log n)$ time complexity.

Cost of operations and partition

To determine the complexity lower bound, we need to define the cost of the operations used in the algorithm.

Definition (Cost of the operation (R, S))

$$\mathrm{cost}(R, S) := |\{(u, v) \mid u \in R, v \in S, uv \in E(G)\}|$$

Definition (Cost of a partition)

$$\begin{cases} \mathrm{cost}(\pi) := 0 & \text{if } \pi \text{ is stable} \\ \mathrm{cost}(\pi) := \min_{R,S} \mathrm{cost}(\pi(R, S)) + \mathrm{cost}(R, S) & \text{otherwise} \end{cases}$$

where the minimum is taken over all effective refining operations (R, S) that can be applied to π.

Proposition (Monotonicity)

Let π and ρ be such that $\pi \preceq \rho \preceq \pi_\infty$. Then $\mathrm{cost}(\pi) \geq \mathrm{cost}(\rho)$.

Cost of operations and partition

To determine the complexity lower bound, we need to define the cost of the operations used in the algorithm.

Definition (Cost of the operation (R, S))

$$\mathrm{cost}(R, S) := |\{(u, v) \mid u \in R, v \in S, uv \in E(G)\}|$$

Definition (Cost of a partition)

$$\begin{cases} \mathrm{cost}(\pi) := 0 & \text{if } \pi \text{ is stable} \\ \mathrm{cost}(\pi) := \min_{R,S} \mathrm{cost}(\pi(R, S)) + \mathrm{cost}(R, S) & \text{otherwise} \end{cases}$$

where the minimum is taken over all effective refining operations (R, S) that can be applied to π.

Proposition (Monotonicity)

Let π and ρ be such that $\pi \preceq \rho \preceq \pi_\infty$. Then $\mathrm{cost}(\pi) \geq \mathrm{cost}(\rho)$.

Cost of operations and partition

To determine the complexity lower bound, we need to define the cost of the operations used in the algorithm.

Definition (Cost of the operation (R, S))

$$\mathrm{cost}(R, S) := |\{(u, v) \mid u \in R, v \in S, uv \in E(G)\}|$$

Definition (Cost of a partition)

$$\begin{cases} \mathrm{cost}(\pi) := 0 & \text{if } \pi \text{ is stable} \\ \mathrm{cost}(\pi) := \min_{R,S} \mathrm{cost}(\pi(R, S)) + \mathrm{cost}(R, S) & \text{otherwise} \end{cases}$$

where the minimum is taken over all effective refining operations (R, S) that can be applied to π.

Proposition (Monotonicity)

Let π and ρ be such that $\pi \preceq \rho \preceq \pi_\infty$. Then $\mathrm{cost}(\pi) \geq \mathrm{cost}(\rho)$.

Cost of operations and partition

To determine the complexity lower bound, we need to define the cost of the operations used in the algorithm.

Definition (Cost of the operation (R, S))

$$\mathrm{cost}(R, S) := |\{(u, v) \mid u \in R, v \in S, uv \in E(G)\}|$$

Definition (Cost of a partition)

$$\begin{cases} \mathrm{cost}(\pi) := 0 & \text{if } \pi \text{ is stable} \\ \mathrm{cost}(\pi) := \min_{R,S} \mathrm{cost}(\pi(R, S)) + \mathrm{cost}(R, S) & \text{otherwise} \end{cases}$$

where the minimum is taken over all effective refining operations (R, S) that can be applied to π.

Proposition (Monotonicity)

Let π and ρ be such that $\pi \preceq \rho \preceq \pi_\infty$. Then $\mathrm{cost}(\pi) \geq \mathrm{cost}(\rho)$.

Cost of operations and partition

To determine the complexity lower bound, we need to define the cost of the operations used in the algorithm.

Definition (Cost of the operation (R, S))

$$\mathrm{cost}(R, S) := |\{(u, v) \mid u \in R, v \in S, uv \in E(G)\}|$$

Definition (Cost of a partition)

$$\begin{cases} \mathrm{cost}(\pi) := 0 & \text{if } \pi \text{ is stable} \\ \mathrm{cost}(\pi) := \min_{R,S} \mathrm{cost}(\pi(R, S)) + \mathrm{cost}(R, S) & \text{otherwise} \end{cases}$$

where the minimum is taken over all effective elementary refining operations (R, S) that can be applied to π.

Proposition (Monotonicity)

Let π and ρ be such that $\pi \preceq \rho \preceq \pi_\infty$. Then $\mathrm{cost}(\pi) \geq \mathrm{cost}(\rho)$.

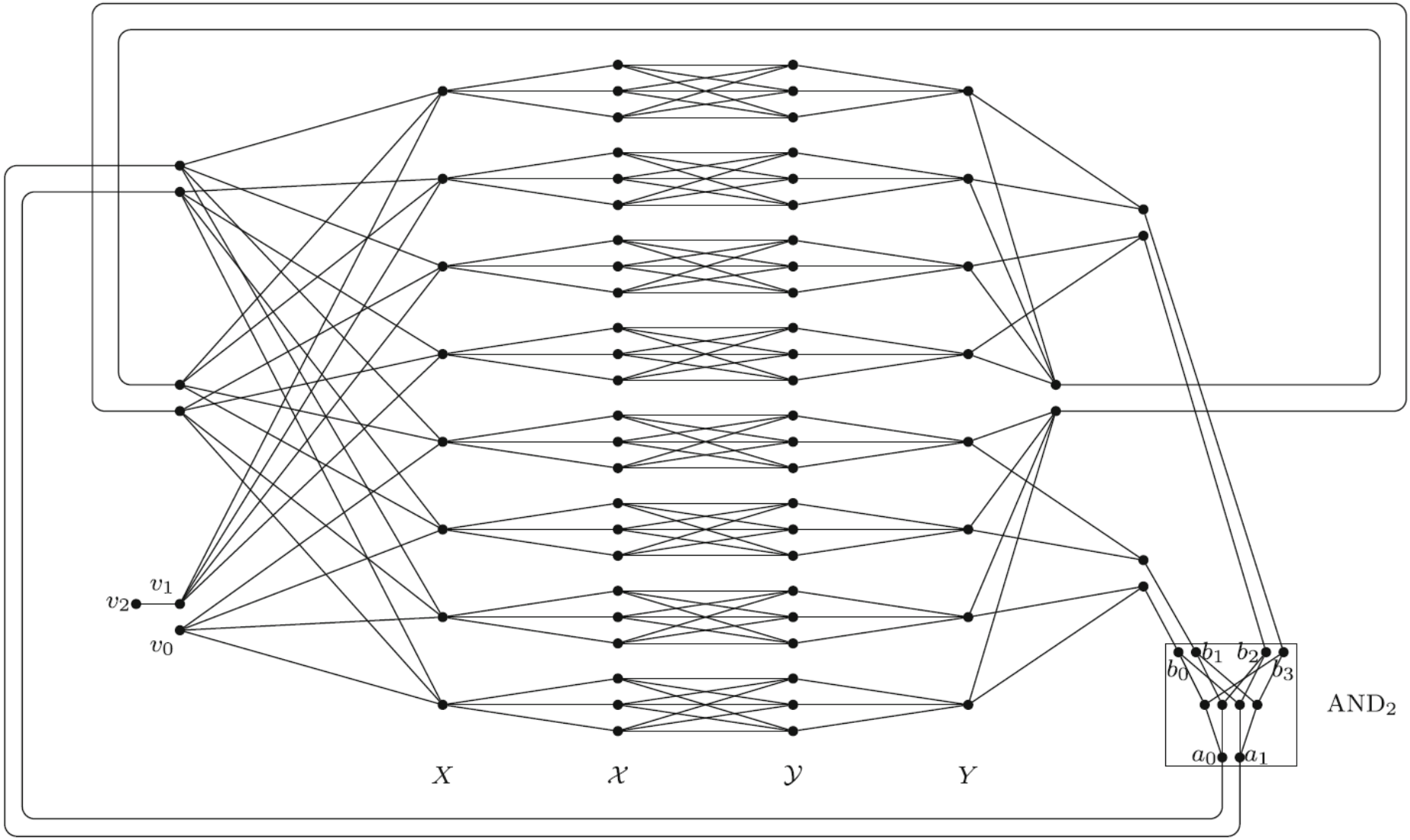
... is this the actual best we can do?

At this point we shall construct a graph representing the worst-case scenario to prove the following.

Theorem

For every integer $k \geq 2$, there is a graph G_k with $n \in O(2^k k)$ vertices and $m \in O(2^k k^2)$ edges, such that $\mathrm{cost}(\alpha) \in \Omega((m+n)\log n)$, where α is the unit partition of $V(G_k)$.

An 'easy' example



Graph construction

For each $k \geq 2$, in G_k there are:

- Layers X, Y: sets with 2^k-many nodes each;
- Layers $\mathcal{X}, \mathcal{Y}$: sets with $2^k k$-many nodes each;
- Gadgets AND_ℓ for $\ell \in \{1, \ldots, k-1\}$:
  - $\ell = 1$: $V = \{a_0, a_1, b_0, b_1\}$, $E = \{a_0 b_0, a_1 b_1\}$;
  - $\ell = 2$: it is X_3 from (Cai, Fürer, Immermann, 1992);
  - $\ell \geq 3$: one copy of AND_2 connected to two copies of $\mathrm{AND}_{\ell-1}$.

Its nodes are connected in such a way that the cost of each refining operation is maximised.

Purpose

Partition refinement of G_k behaves as follows.

1. X, Y are cells.
2. X splits into X_0^1, X_1^1 of equal size:
   - $\rightarrow$ $\mathcal{X}$ is split into $\mathcal{X}_0^1, \mathcal{X}_1^1$ (*binary blocks* of level $\ell = 0$),
   - $\rightarrow$ $\mathcal{Y}$ is split accordingly (expensive operation),
   - $\rightarrow$ Y splits into Y_0^1, Y_1^1.
3. AND_1 causes X_0^1, X_1^1 to split again: ...

$\vdots$

F. Discrete colouring of X (and Y).

Cost lower bound

At each step of the refining process, one has only two effective choices for R, S, and either way the operation costs $2^{k-(\ell+1)}k^2$. This leads to a total cost of $2^{k-1}k^3 = \Omega(m \log n)$ to partition each binary block of level $\ell + 1$ into binary blocks of level $\ell + 2$.

What's next?

- Consider partitions that are stable w.r.t. $G'_k = G_k - [\mathcal{X}, \mathcal{Y}]$ and such that $\mathcal{X}, \mathcal{Y}$ are partitioned into binary blocks.
- Show that they can be refined only using refining operations (R, S) where R is a binary block of $\mathcal{X}$ and S is a binary block of $\mathcal{Y}$.

As $n = |V(G_k)| \in O(2^k k)$ and $m = |E(G_k)| \in O(2^k k^2)$, we get

$$\text{cost}(\alpha) \in \Omega((m+n) \log n).$$

Cost lower bound

At each step of the refining process, one has only two effective choices for R, S, and either way the operation costs $2^{k-(\ell+1)}k^2$. This leads to a total cost of $2^{k-1}k^3 = \Omega(m \log n)$ to partition each binary block of level $\ell + 1$ into binary blocks of level $\ell + 2$.

What's next?

- Consider partitions that are stable w.r.t. $G'_k = G_k - [\mathcal{X}, \mathcal{Y}]$ and such that $\mathcal{X}, \mathcal{Y}$ are partitioned into binary blocks.
- Show that they can be refined only using refining operations (R, S) where R is a binary block of $\mathcal{X}$ and S is a binary block of $\mathcal{Y}$.

As $n = |V(G_k)| \in O(2^k k)$ and $m = |E(G_k)| \in O(2^k k^2)$, we get

$$\text{cost}(\alpha) \in \Omega((m+n)\log n).$$

Cost lower bound

At each step of the refining process, one has only two effective choices for R, S, and either way the operation costs $2^{k-(\ell+1)}k^2$. This leads to a total cost of $2^{k-1}k^3 = \Omega(m \log n)$ to partition each binary block of level $\ell + 1$ into binary blocks of level $\ell + 2$.

What's next?

- Consider partitions that are stable w.r.t. $G'_k = G_k - [\mathcal{X}, \mathcal{Y}]$ and such that $\mathcal{X}, \mathcal{Y}$ are partitioned into binary blocks.
- Show that they can be refined only using refining operations (R, S) where R is a binary block of $\mathcal{X}$ and S is a binary block of $\mathcal{Y}$.

As $n = |V(G_k)| \in O(2^k k)$ and $m = |E(G_k)| \in O(2^k k^2)$, we get

$$\text{cost}(\alpha) \in \Omega((m + n) \log n).$$

Bisimilarity

A directed coloured G represents a transition system.

Definition (Bisimilarity)

Coarsest bisimulation $\sim$ on V, i.e. such that $v \sim w$ implies

- $\lambda(v) = \lambda(w)$;
- for all $v' \in N^+(v)$ there exists a $w' \in N^+(w)$ s.t. $v' \sim w'$;
- for all $w' \in N^+(w)$ there exists a $v' \in N^+(v)$ s.t. $v' \sim w'$.

Instead of refining a class by the degree towards another, we refine by the Boolean value 'degree at least 1'.

Complexity lower bound

Lower bound for colour refinement implies a lower bound for bisimilarity.

Bisimilarity

A directed coloured G represents a transition system.

Definition (Bisimilarity)

Coarsest bisimulation $\sim$ on V, i.e. such that $v \sim w$ implies

- $\lambda(v) = \lambda(w)$;
- for all $v' \in N^+(v)$ there exists a $w' \in N^+(w)$ s.t. $v' \sim w'$;
- for all $w' \in N^+(w)$ there exists a $v' \in N^+(v)$ s.t. $v' \sim w'$.

Instead of refining a class by the degree towards another, we refine by the Boolean value 'degree at least 1'.

Complexity lower bound

Lower bound for colour refinement implies a lower bound for bisimilarity.

Bisimilarity

A directed coloured G represents a transition system.

Definition (Bisimilarity)

Coarsest bisimulation $\sim$ on V, i.e. such that $v \sim w$ implies

- $\lambda(v) = \lambda(w)$;
- for all $v' \in N^+(v)$ there exists a $w' \in N^+(w)$ s.t. $v' \sim w'$;
- for all $w' \in N^+(w)$ there exists a $v' \in N^+(v)$ s.t. $v' \sim w'$.

Instead of refining a class by the degree towards another, we refine by the Boolean value 'degree at least 1'.

Complexity lower bound

Lower bound for colour refinement implies a lower bound for bisimilarity.

Bisimilarity

A directed coloured G represents a transition system.

Definition (Bisimilarity)

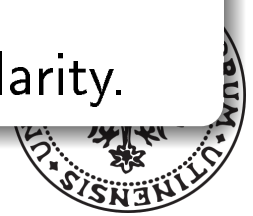
Coarsest bisimulation $\sim$ on V, i.e. such that $v \sim w$ implies

- $\lambda(v) = \lambda(w)$;
- for all $v' \in N^+(v)$ there exists a $w' \in N^+(w)$ s.t. $v' \sim w'$;
- for all $w' \in N^+(w)$ there exists a $v' \in N^+(v)$ s.t. $v' \sim w'$.

Instead of refining a class by the degree towards another, we refine by the Boolean value 'degree at least 1'.

Complexity lower bound

Lower bound for colour refinement implies a lower bound for bisimilarity.



Two-variable first-order logic with counting

Theorem (Immermann, Ladner, 1990)

For all v, w in a graph G, v and w have the same colour in the coarsest bi-stable colouring of G iff they are $\mathcal{C}^2$-equivalent.

Complexity lower bound

$\mathcal{C}^2$-equivalence classes can be computed in time no less than $O((m+n)\log n)$.

Two-variable first-order logic with counting

Theorem (Immermann, Ladner, 1990)

For all v, w in a graph G, v and w have the same colour in the coarsest bi-stable colouring of G iff they are $\mathcal{C}^2$-equivalent.

Complexity lower bound

$\mathcal{C}^2$-equivalence classes can be computed in time no less than $O((m+n)\log n)$.

Two-variable first-order logic with counting

Theorem (Immermann, Ladner, 1990)

For all v, w in a graph G, v and w have the same colour in the coarsest bi-stable colouring of G iff they are $\mathcal{C}^2$-equivalent.

Complexity lower bound

$\mathcal{C}^2$-equivalence classes can be computed in time no less than $O((m+n)\log n)$.

DFA minimisation

The graph G_k turns into a highly non-deterministic transition system. Hence the complexity lower bound **do not** apply to *Deterministic Finite state Automata* DFAs.

Open problem

Is DFA-minimisation possible in linear time?

Known special case

(Paige, Tarjan, Bonic, 1985) showed that a DFA with a single-letter function can be minimised in linear time.

DFA minimisation

The graph G_k turns into a highly non-deterministic transition system. Hence the complexity lower bound **do not** apply to *Deterministic Finite state Automata* DFAs.

Open problem

Is DFA-minimisation possible in linear time?

Known special case

(Paige, Tarjan, Bonic, 1985) showed that a DFA with a single-letter function can be minimised in linear time.

DFA minimisation

The graph G_k turns into a highly non-deterministic transition system. Hence the complexity lower bound **do not** apply to *Deterministic Finite state Automata* DFAs.

Open problem

Is DFA-minimisation possible in linear time?

Known special case

(Paige, Tarjan, Bonic, 1985) showed that a DFA with a single-letter function can be minimised in linear time.

Further developments

Stronger lower bounds

(Groote, Martens, de Vink, 2023) shows that a tighter lower bound of $\Omega(n)$ can be applied to bisimulation refinement by means of parallel algorithms.

Coalgebraic refinement

(Wissmann, Dorsch, Milius, Schröder, 2020) presents a coalgebraic partition refinement algorithm, extending the complexity lower bound $O((m+n)\log n)$ to e.g. Markov chains and Segala systems.

Bibliography

Berkholz, Bonsma, Grohe. *Tight Lower and Upper Bound for the Complexity of Canonical Colour Refinement.* (2017)

- Babai, Erdös, Selkov. *Random Graph Isomorphism.* (1980)
- Cai, Fürer, Immerman. *An optimal lower bound on the number of variables for graph identifications.* (1992)
- Cardon, Crochemore. *Partitioning a graph in* $O(|A| \log_2 |V|)$. (1982)
- McKay. *Practical Graph Isomorphism.* (1981)
- McKay, Piperno. *Practical Graph Isomorphism II.* (2014)
- Immerman, Lander. *Describing Graphs: a First-Order Approach to Graph Canonization.* (1990)
- Paige, Tarjan. *Three partition refinement algorithms.* (1987)
- Paige, Tarjan, Bonic. *A linear time solution to the single function coarsest partition problem.* (1985)