Polychromatic Colourings of Integers

Bhavik Mehta

Imperial College London

December 11, 2025

Table of Contents

1 Polychromatic colourings

2 Formalisation

3 Computation

Definitions

Let G be an arbitrary abelian group (it will typically be $\mathbb{Z}$ or $\mathbb{Z}/q\mathbb{Z}$)

Definitions

Let G be an arbitrary abelian group (it will typically be $\mathbb{Z}$ or $\mathbb{Z}/q\mathbb{Z}$)

Definition (Polychromatic colouring)

Given a finite subset $S \subseteq G$, a colouring χ of G is *polychromatic* for S if every translate of S meets every colour.

Definitions

Let G be an arbitrary abelian group (it will typically be $\mathbb{Z}$ or $\mathbb{Z}/q\mathbb{Z}$)

Definition (Polychromatic colouring)

Given a finite subset $S \subseteq G$, a colouring χ of G is *polychromatic* for S if every translate of S meets every colour.

Definition (Polychromatic number)

The polychromatic number $p(S)$ of $S \subseteq G$ is the largest number of colours possible in a polychromatic colouring for S.

Definitions

Let G be an arbitrary abelian group (it will typically be $\mathbb{Z}$ or $\mathbb{Z}/q\mathbb{Z}$)

Definition (Polychromatic colouring)

Given a finite subset $S \subseteq G$, a colouring χ of G is *polychromatic* for S if every translate of S meets every colour.

Definition (Polychromatic number)

The polychromatic number $p(S)$ of $S \subseteq G$ is the largest number of colours possible in a polychromatic colouring for S.

Must have $1 \leq p(S) \leq \#S$

Examples

If $S = \{0, 1\}$, then $p(S) = 2$. In fact if $\#S = 2$, then $p(S) = 2$.

Examples

If $S = \{0, 1\}$, then $p(S) = 2$. In fact if $\#S = 2$, then $p(S) = 2$.

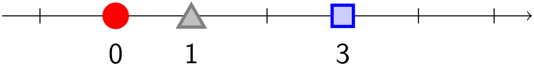
If $\#S = 3$, then $\#S$ may be 2 or 3.
For instance, if $S = \{0, 1, 2\}$, then we have an easy 3-colouring:

Examples

If $S = \{0, 1\}$, then $p(S) = 2$. In fact if $\#S = 2$, then $p(S) = 2$.

If $\#S = 3$, then $\#S$ may be 2 or 3.
For instance, if $S = \{0, 1, 2\}$, then we have an easy 3-colouring:

But if $S = \{0, 1, 3\}$, then no polychromatic 3-colouring can exist. Suppose we are given one:



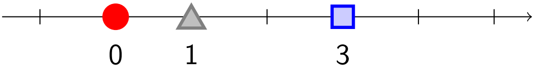
Any choice of colour for 2 would produce a contradiction, eg

Examples

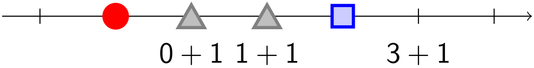
If $S = \{0, 1\}$, then $p(S) = 2$. In fact if $\#S = 2$, then $p(S) = 2$.

If $\#S = 3$, then $\#S$ may be 2 or 3.
For instance, if $S = \{0, 1, 2\}$, then we have an easy 3-colouring:

But if $S = \{0, 1, 3\}$, then no polychromatic 3-colouring can exist. Suppose we are given one:



Any choice of colour for 2 would produce a contradiction, eg

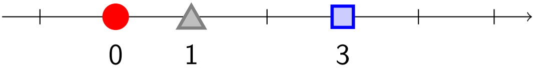


Examples

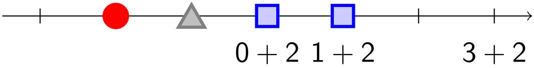
If $S = \{0, 1\}$, then $p(S) = 2$. In fact if $\#S = 2$, then $p(S) = 2$.

If $\#S = 3$, then $\#S$ may be 2 or 3.
For instance, if $S = \{0, 1, 2\}$, then we have an easy 3-colouring:

But if $S = \{0, 1, 3\}$, then no polychromatic 3-colouring can exist. Suppose we are given one:



Any choice of colour for 2 would produce a contradiction, eg

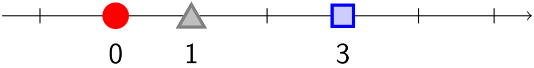


Examples

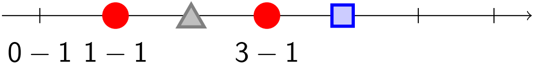
If $S = \{0, 1\}$, then $p(S) = 2$. In fact if $\#S = 2$, then $p(S) = 2$.

If $\#S = 3$, then $\#S$ may be 2 or 3.
For instance, if $S = \{0, 1, 2\}$, then we have an easy 3-colouring:

But if $S = \{0, 1, 3\}$, then no polychromatic 3-colouring can exist. Suppose we are given one:



Any choice of colour for 2 would produce a contradiction, eg



Bounding the size

- Monotonicity! Using fewer colours makes it easier, and bigger sets are easier.

Bounding the size

- Monotonicity! Using fewer colours makes it easier, and bigger sets are easier.
- If I fix the set, there's always some small number of colours which gives me a polychromatic colouring.

Bounding the size

- Monotonicity! Using fewer colours makes it easier, and bigger sets are easier.
- If I fix the set, there's always some small number of colours which gives me a polychromatic colouring.
- But what if I fix the number of colours?

Bounding the size

- Monotonicity! Using fewer colours makes it easier, and bigger sets are easier.
- If I fix the set, there's always some small number of colours which gives me a polychromatic colouring.
- But what if I fix the number of colours?
- With two colours, sets of size 2 can always have a polychromatic colouring, but not size 3.

Bounding the size

- Monotonicity! Using fewer colours makes it easier, and bigger sets are easier.
- If I fix the set, there's always some small number of colours which gives me a polychromatic colouring.
- But what if I fix the number of colours?
- With two colours, sets of size 2 can always have a polychromatic colouring, but not size 3.
- Strauss asked: If we fix the number of colours k, is there a large enough size m such that every S with $\#S \geq m$ admits a polychromatic k-colouring? In other words, such that every S with $\#S \geq m$ has $p(S) \geq k$?

First bound

Theorem (Erdős, Lovász 1975)

Yes, $m \leq (3 + o(1))k \log k$ works. In particular, some finite m works.

Equivalently, $p(S) \geq \frac{\#S}{(3+o(1)) \log \#S}$.

Small sets

In the case $\#S = 4$, Axenovich, Goldwasser, Lidický, Martin, Offner, Talbot and Young (2021) proved $p(S) \geq 3$, and hence that $m(3) = 4$.

Small sets

In the case $\#S = 4$, Axenovich, Goldwasser, Lidický, Martin, Offner, Talbot and Young (2021) proved $p(S) \geq 3$, and hence that $m(3) = 4$.
We will discuss the work-in-progress formalisation of these two results.

Table of Contents

1 Polychromatic colourings

2 Formalisation

3 Computation

Definitions

```
def IsPolychrom (S : Finset G) (χ : G → K) :
    Prop :=
  ∀ n : G, ∀ k : K, ∃ a ∈ S, χ (n + a) = k

lemma IsPolychrom.nonempty (h : IsPolychrom S χ) :
    S.Nonempty := by
  obtain ⟨i, hi, hi'⟩ := h 0 (χ 0)
  use i

lemma IsPolychrom.finite (hχ : IsPolychrom S χ) :
    Finite K := ...
```

More definitions


```
def HasPolychromColouring (K : Type*)
    (S : Finset G) : Prop :=
  ∃ χ : G → K, IsPolychrom S χ

lemma HasPolychromColouring.nonempty_set
    (h : HasPolychromColouring K S) :
    S.Nonempty := by
  obtain ⟨χ, hχ⟩ := h
  exact hχ.nonempty
```

Local lemma

- If you can't construct an object, make it randomly!

Local lemma

- If you can't construct an object, make it randomly!
- Erdős and Lovász' proof introduced the (Lovász) local lemma

Local lemma

- If you can't construct an object, make it randomly!
- Erdős and Lovász' proof introduced the (Lovász) local lemma
- Given an independent collection A_i of 'bad' events, if each $\mathbb{P}(A_i) < 1$, then the probability of all of them failing is nonzero

Local lemma

- If you can't construct an object, make it randomly!
- Erdős and Lovász' proof introduced the (Lovász) local lemma
- Given an independent collection A_i of 'bad' events, if each $\mathbb{P}(A_i) < 1$, then the probability of all of them failing is nonzero
- But what if they're not independent?

Local lemma

- If you can't construct an object, make it randomly!
- Erdős and Lovász' proof introduced the (Lovász) local lemma
- Given an independent collection A_i of 'bad' events, if each $\mathbb{P}(A_i) < 1$, then the probability of all of them failing is nonzero
- But what if they're not independent?
- We can still show that the probability of all A_i failing is nonzero, if each individual one is unlikely enough and they are 'mostly' independent

Formal Local Lemma


```
theorem symmetricLocalLemma
    {h0 : MeasurableSpace Ω} {P : Measure Ω}
    [IsProbabilityMeasure P] [Fintype ι]
    (hA : ∀ i, MeasurableSet (A i))
    {p : ℝ} (hAp : ∀ i, P (A i) ≤ p)
    {N : ι → Finset ι}
    (h : lopsidedCondition P A N)
    {d : ℕ} (hd : d ≠ 0)
    (hN : ∀ i, #(N i) ≤ d)
    (hpd : exp 1 * p * (d + 1) ≤ 1) :
    0 < P (⋂ i, (A i)ᶜ) := by ...
```

Formal Local Lemma


```
theorem symmetricLocalLemma
    {h0 : MeasurableSpace Ω} {P : Measure Ω}
    [IsProbabilityMeasure P] [Fintype ι]
    (hA : ∀ i, MeasurableSet (A i))
    {p : ℝ} (hAp : ∀ i, P (A i) ≤ p)
    {N : ι → Finset ι}
    (h : lopsidedCondition P A N)
    {d : ℕ} (hd : d ≠ 0)
    (hN : ∀ i, #(N i) ≤ d)
    (hpd : exp 1 * p * (d + 1) ≤ 1) :
    0 < P (⋂ i, (A i)ᶜ) := by ...
```

Pick the right version to prove first!

Formal Local Lemma


```
theorem symmetricLocalLemma
    {h0 : MeasurableSpace Ω} {P : Measure Ω}
    [IsProbabilityMeasure P] [Fintype ι]
    (hA : ∀ i, MeasurableSet (A i))
    {p : ℝ} (hAp : ∀ i, P (A i) ≤ p)
    {N : ι → Finset ι}
    (h : lopsidedCondition P A N)
    {d : ℕ} (hd : d ≠ 0)
    (hN : ∀ i, #(N i) ≤ d)
    (hpd : exp 1 * p * (d + 1) ≤ 1) :
    0 < P (⋂ i, (A i)ᶜ) := by ...
```

Pick the right version to prove first!

Applying the Local Lemma

- The ‘bad’ event A_i will be the translation $S + i$ not receiving some colour

Applying the Local Lemma

- The ‘bad’ event A_i will be the translation $S + i$ not receiving some colour
- Events aren’t independent, but they don’t overlap too much

Applying the Local Lemma

- The 'bad' event A_i will be the translation $S + i$ not receiving some colour
- Events aren't independent, but they don't overlap too much
- Works reasonably well, we can estimate A_i easily, and bound d enough

Applying the Local Lemma

- The 'bad' event A_i will be the translation $S + i$ not receiving some colour
- Events aren't independent, but they don't overlap too much
- Works reasonably well, we can estimate A_i easily, and bound d enough
- But this only works for finitely many events!

Applying the Local Lemma

- The 'bad' event A_i will be the translation $S + i$ not receiving some colour
- Events aren't independent, but they don't overlap too much
- Works reasonably well, we can estimate A_i easily, and bound d enough
- But this only works for finitely many events!
- So the theorem only works if we only care about finitely many translations...

Applying the Local Lemma

- The 'bad' event A_i will be the translation $S + i$ not receiving some colour
- Events aren't independent, but they don't overlap too much
- Works reasonably well, we can estimate A_i easily, and bound d enough
- But this only works for finitely many events!
- So the theorem only works if we only care about finitely many translations...
- It works for finite groups, which is nice, but not good enough

Selection Lemma

Instead, uncover Rado's Selection Lemma

Selection Lemma

Instead, uncover Rado's Selection Lemma

```
theorem Finset.rado_selection {α β : Type}
    [Finite β] (g : Finset α → α → β) :
    ∃ χ : α → β, ∀ s : Finset α,
      ∃ t : Finset α, s ⊆ t ∧
        ∀ x ∈ s, χ x = g t x := by ...
```

Selection Lemma

Instead, uncover Rado's Selection Lemma

```
theorem Finset.rado_selection {α β : Type}
    [Finite β] (g : Finset α → α → β) :
    ∃ χ : α → β, ∀ s : Finset α,
      ∃ t : Finset α, s ⊆ t ∧
        ∀ x ∈ s, χ x = g t x := by ...
```

Proof uses Tychonoff, and that the intersection of closed sets in a compact space having the finite intersection property is nonempty.

Selection Lemma

Instead, uncover Rado's Selection Lemma

```
theorem Finset.rado_selection {α β : Type}
    [Finite β] (g : Finset α → α → β) :
    ∃ χ : α → β, ∀ s : Finset α,
      ∃ t : Finset α, s ⊆ t ∧
        ∀ x ∈ s, χ x = g t x := by ...
```

Proof uses Tychonoff, and that the intersection of closed sets in a compact space having the finite intersection property is nonempty.
This 'compactness' principle appears often throughout combinatorics, so isolating it into Rado's Selection Lemma helps formal re-use.

Analysis

Putting those pieces together, we get

```
lemma exists_of_le {k m : ℕ}
    {S : Finset G} (hm : #S = m)
    (hm₂ : 2 ≤ m) (hk : k ≠ 0)
    (hkm : Real.exp 1 * (m * (m - 1) + 1) *
                k * (1 - (k : ℝ)⁻¹) ^ m ≤ 1) :
    HasPolychromColouring (Fin k) S := by
  ...
```

Analysis

Putting those pieces together, we get

```
lemma exists_of_le {k m : ℕ}
    {S : Finset G} (hm : #S = m)
    (hm₂ : 2 ≤ m) (hk : k ≠ 0)
    (hkm : Real.exp 1 * (m * (m - 1) + 1) *
          k * (1 - (k : ℝ)⁻¹) ^ m ≤ 1) :
    HasPolychromColouring (Fin k) S := by
  ...
```

The bound doesn't depend on the group!

Analysis

Putting those pieces together, we get

```
lemma exists_of_le {k m : ℕ}
    {S : Finset G} (hm : #S = m)
    (hm₂ : 2 ≤ m) (hk : k ≠ 0)
    (hkm : Real.exp 1 * (m * (m - 1) + 1) *
              k * (1 - (k : ℝ)⁻¹) ^ m ≤ 1) :
    HasPolychromColouring (Fin k) S := by
  ...
```

The bound doesn't depend on the group!
It turns out to be satisfied if $m \geq 3k^2$, or with a bit of calculus, $m \geq k(3 \log k + 2 \log \log k + 5.2) = (3 + o(1)) k \log k$.

Asymptotic answer

```
theorem exists_colouring_asymptotic
    {ε : ℝ} (hε : 0 < ε) :
    ∀ᶠ k : ℕ in atTop, ∀ S : Finset G,
      (3 + ε) * k * log k ≤ #S →
        HasPolychromColouring (Fin k) S := by
```

Asymptotic answer


```
theorem exists_colouring_asymptotic
    {ε : ℝ} (hε : 0 < ε) :
    ∀ᶠ k : ℕ in atTop, ∀ S : Finset G,
      (3 + ε) * k * log k ≤ #S →
        HasPolychromColouring (Fin k) S := by
```

Filters are great!

So far...

- Probability, topology, calculus

So far...

- Probability, topology, calculus
- Formalisation involves understanding, restructuring and sometimes just translating an informal proof

So far...

- Probability, topology, calculus
- Formalisation involves understanding, restructuring and sometimes just translating an informal proof
- Often end up with more steps in the formal than informal versions, but the arguments are human-sized anyway

So far...

- Probability, topology, calculus
- Formalisation involves understanding, restructuring and sometimes just translating an informal proof
- Often end up with more steps in the formal than informal versions, but the arguments are human-sized anyway
- Mathlib's global coherence allowed me to apply probability to my question

So far...

- Probability, topology, calculus
- Formalisation involves understanding, restructuring and sometimes just translating an informal proof
- Often end up with more steps in the formal than informal versions, but the arguments are human-sized anyway
- Mathlib's global coherence allowed me to apply probability to my question
- Looking for general results encouraged me to find Rado's Selection Lemma

So far...

- Probability, topology, calculus
- Formalisation involves understanding, restructuring and sometimes just translating an informal proof
- Often end up with more steps in the formal than informal versions, but the arguments are human-sized anyway
- Mathlib's global coherence allowed me to apply probability to my question
- Looking for general results encouraged me to find Rado's Selection Lemma
- Automation helped prove the results in calculus (but this one could be better!)

Table of Contents

1 Polychromatic colourings

2 Formalisation

3 Computation

The proof

I began reading the proof that if $\#S = 4$ is a set of integers, $p(S) \geq 3$...

The proof

I began reading the proof that if $\#S = 4$ is a set of integers, $p(S) \geq 3$...

> *It is possible, though tedious, to prove the entire theorem by hand.*

The proof

I began reading the proof that if $\#S = 4$ is a set of integers, $p(S) \geq 3$...

> *It is possible, though tedious, to prove the entire theorem by hand. Thus in the interest of simplifying the exposition, we verified*

The proof

I began reading the proof that if $\#S = 4$ is a set of integers, $p(S) \geq 3$...

> *It is possible, though tedious, to prove the entire theorem by hand. Thus in the interest of simplifying the exposition, we verified using a computer search that for every S with diameter at most 288 there exists an S-polychromatic 3-coloring...*

The proof

I began reading the proof that if $\#S = 4$ is a set of integers, $p(S) \geq 3$...

> *It is possible, though tedious, to prove the entire theorem by hand. Thus in the interest of simplifying the exposition, we verified using a computer search that for every S with diameter at most 288 there exists an S-polychromatic 3-coloring...*

How many such S are there? $\binom{288}{3} \approx 4$ million.

The computer search

The code for this search has been included as an ancillary file with the preprint of this paper

The computer search

The code for this search has been included as an ancillary file with the preprint of this paper

- I tried running the C++ code; it didn't terminate after 3 days.

The computer search

The code for this search has been included as an ancillary file with the preprint of this paper

- I tried running the C++ code; it didn't terminate after 3 days.
- The 'manual' proof seems lost

The computer search

The code for this search has been included as an ancillary file with the preprint of this paper

- I tried running the C++ code; it didn't terminate after 3 days.
- The 'manual' proof seems lost
  There is apparently a handwritten version in John G.'s notes without any computer search - but this was never shared [...]. (He has since retired and these notes are in one of some non-trivial number of boxes...)

The computer search

The code for this search has been included as an ancillary file with the preprint of this paper

- I tried running the C++ code; it didn't terminate after 3 days.
- The 'manual' proof seems lost
  There is apparently a handwritten version in John G.'s notes without any computer search - but this was never shared [...]. (He has since retired and these notes are in one of some non-trivial number of boxes...)
- Can we salvage the proof?

Computers to the rescue

- The C++ code is only slow when trying to find a few of the colourings

Computers to the rescue

- The C++ code is only slow when trying to find a few of the colourings
- For most of them, it finds me the colouring fast which I can dump to file

Computers to the rescue

- The C++ code is only slow when trying to find a few of the colourings
- For most of them, it finds me the colouring fast which I can dump to file
- For the few that don't finish quickly, instead stop trying, and...

Computers to the rescue

- The C++ code is only slow when trying to find a few of the colourings
- For most of them, it finds me the colouring fast which I can dump to file
- For the few that don't finish quickly, instead stop trying, and...
- Z3 is a theorem prover from 2012, particularly good at software verification tasks and solving combinatorial optimisation problems

Computers to the rescue

- The C++ code is only slow when trying to find a few of the colourings
- For most of them, it finds me the colouring fast which I can dump to file
- For the few that don't finish quickly, instead stop trying, and...
- Z3 is a theorem prover from 2012, particularly good at software verification tasks and solving combinatorial optimisation problems
- So, use Z3 to fill in the gaps

Computers to the rescue

- The C++ code is only slow when trying to find a few of the colourings
- For most of them, it finds me the colouring fast which I can dump to file
- For the few that don't finish quickly, instead stop trying, and...
- Z3 is a theorem prover from 2012, particularly good at software verification tasks and solving combinatorial optimisation problems
- So, use Z3 to fill in the gaps
- I got Z3 (with Python) to help the search get over the line, and print out the colourings generated

Sending it to Lean

- For each $0 < a < b < c < 289$ we *should* now have a valid colouring

Sending it to Lean

- For each $0 < a < b < c < 289$ we *should* now have a valid colouring
- Read them all into Lean, and have Lean check that each one indeed works

Sending it to Lean

- For each $0 < a < b < c < 289$ we *should* now have a valid colouring
- Read them all into Lean, and have Lean check that each one indeed works
- 'Certificate'-based verification: I don't need to verify the Z3/C++ code, just verify that the output it gave does what I need

Sending it to Lean

- For each $0 < a < b < c < 289$ we *should* now have a valid colouring
- Read them all into Lean, and have Lean check that each one indeed works
- 'Certificate'-based verification: I don't need to verify the Z3/C++ code, just verify that the output it gave does what I need
- Lean's metaprogramming is written in Lean, so this is clean to do

Sending it to Lean

- For each $0 < a < b < c < 289$ we *should* now have a valid colouring
- Read them all into Lean, and have Lean check that each one indeed works
- 'Certificate'-based verification: I don't need to verify the Z3/C++ code, just verify that the output it gave does what I need
- Lean's metaprogramming is written in Lean, so this is clean to do
- To check an individual one, write kernel-optimised bitwise operations

```
example :
    ∀ (a b c : ℤ),
      0 < a → a < b → b < c → c < 289 →
    HasPolychromColouring (Fin 3) {0, a, b, c}
```

Summary

- Lean enables formalisation of a huge range of mathematics

Summary

- Lean enables formalisation of a huge range of mathematics
- Mathlib enables doing this increasingly quickly, as it grows

Summary

- Lean enables formalisation of a huge range of mathematics
- Mathlib enables doing this increasingly quickly, as it grows
- Nothing is impossible!

Acknowledgments

Joachim Breitner, Mario Carneiro, David Thrane Christiansen, Rémy Degenne, Marc Huisinga, Kenny Lau, Jason Reed, John Talbot, Sebastian Ullrich, Eric Wieser, Andrew Yang

Thank you for your attention!