The Power of Locality

Case Study: Graph Coloring
Case Study: Graph Coloring
Case Study: Graph Coloring

Assign colors to nodes.
Case Study: Graph Coloring

**Legal coloring:** neighbors have different colors!
Case Study: Graph Coloring

Optimal coloring: Minimal number of colors (aka chromatic number)
Applications

Country Maps

- Neighboring states should have different colors!
- Famous 4-color theorem: any map can be painted with four colors!

Medium Access

- Interference-free, efficient utilization of spectrum
- Neighboring cells should have different frequencies!
- Colors = frequencies, channels, etc.

Image Processing

- Chromatic scheduling for physical simulation
- Process nodes of same color in parallel without determinacy race
- No coordination, no mutual exclusion needed
Legal coloring? Chromatic number?
Legal coloring? Chromatic number?

Tree! 2 colors are enough...
What about this example?
What about this example?

3 colors needed and enough...
How to color a graph in a distributed manner?
How to color a graph in a distributed manner?
Send...

... in each round!

... receive...

... compute.
We will see in this course: there are techniques to execute an algorithm designed in the simple LOCAL model also in asynchronous networks!

The LOCAL Model: A Convenient Synchronous Model

We will see in this course: there are techniques to execute an algorithm designed in the simple LOCAL model also in asynchronous networks!

Send...

... receive...

... compute.

... in each round!
The LOCAL Model: A Convenient Synchronous Model

We will see in this course: there are techniques to execute an algorithm designed in the simple LOCAL model also in asynchronous networks!

Moreover, LOCAL algorithms can be made very robust (namely self-stabilizing), in an automatic manner!

Send...

... in each round!

... receive...

... compute.
The LOCAL Model: A Convenient Synchronous Model

We will see in this course: there are techniques to execute an algorithm designed in the simple LOCAL model also in asynchronous networks!

Moreover, LOCAL algorithms can be made very robust (namely self-stabilizing), in an automatic manner!

Unlike CONGEST model: message size and link capacity not bounded.
LOCAL Performance Metrics

- **Time Complexity:**
  Number of communication rounds

- **Message Complexity:**
  Number of messages sent

- **Local Computation:**
  Complexity of local computations
LOCAL Performance Metrics

- **Time Complexity:**
  Number of communication rounds

- **Message Complexity:**
  Number of messages sent

- **Local Computation:**
  Complexity of local computations

What else?
LOCAL Performance Metrics

- **Time Complexity:**
  - Number of communication rounds

- **Message Complexity:**
  - Number of messages sent

- **Local Computation:**
  - Complexity of local computations

- What else?
  - Quality of solution: Approximation ratio for example („price of locality“).
How to color a tree in a distributed manner?
How to color a *rooted tree* in a distributed manner?

Simplification:
- Assume unique node IDs
- Assume rooted
- Root ID 0
How to color a rooted tree in a distributed manner?

Simplification:
- Assume unique node IDs
- Assume rooted
- Root ID 0

Idea: interpret ID as color!
Communicate my color to children and take opposite color from my parent!
Slow Distributed Tree Coloring: Example

Round 1
Slow Distributed Tree Coloring: Example

Round 2
Slow Distributed Tree Coloring: Example

Round 3
Slow Distributed Tree Coloring: Example

Round 3
Slow Tree Algo

If root: color 0, send 0 to children
Otherwise: each node $v$:
  • Wait for message $x$ from parent
  • Choose color $y=1-x$
  • Send $y$ to children
Approximation quality:
Time complexity:
Message complexity:
Local complexity:
Slow Tree: Analysis

- **Approximation quality**: # colors?
- **Time complexity**: # rounds?
- **Message complexity**: # messages?
- **Local complexity**: local computations?
Slow Tree: Analysis

- **Approximation quality:** 2 colors suffice!
- **Time complexity:** $O(n)$, depth of the tree
- **Message complexity:** $O(n)$
- **Local complexity:** trivial, just flip!
Approximation quality: 2 colors suffice!

Time complexity: $O(n)$, depth of the tree

Message complexity: $O(n)$

Local complexity: trivial, just flip!

Can we do faster?
Yes we can!

3-coloring in $O(\log^* n)$ rounds
Yes we can!

3-coloring in $O(\log^* n)$ rounds

Idea: based on ID manipulations
  Again: interpret ID as color

Unique IDs $\rightarrow$ legal (but expensive) coloring!
How can we quickly reduce the ID space?
Intuition: $n$ vs $\log^* n$

$\log n$: How many times do I have to divide by 2 until $<2$?

$n, n/2, n/4, n/8, \ldots, 8, 4, 2, 1$

$\log n$
Intuition: \( n \) vs \( \log^* n \)

**log \( n \):** How many times do I have to :2 until <2?

\( n, n/2, n/4, n/8, \ldots, 8, 4, 2, 1 \)

\[ \log n \]

**loglog \( n \):** How many times do I have to \( \sqrt{x} \) until <2?

\( n, \sqrt{n}, \sqrt[3]{n}, \sqrt[4]{n}, \ldots, <2 \)

\[ \log\log n \]
Intuition: $n$ vs $\log^* n$

$\log n$: How many times do I have to divide by 2 until <2?

$n, \frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \ldots, 8, 4, 2, 1$

$\log \log n$: How many times do I have to take the square root until <2?

$n, \sqrt{n}, \sqrt[2]{n}, \sqrt[3]{n}, \sqrt[4]{n}, \ldots, <2$

$\log^* n$: How many times do I have to take the logarithm until <2?

$n, \log n, \log \log n, \log \log \log n, \ldots, <2$
n = atoms in universe $\approx 10^{80}$
$\log^*(\text{atoms in universe}) \approx 5$
**Slow Algo**

No parallelism!

**Time:** \( n \)

---

**Fast Algo**

Efficient parallel manipulations!

**Time:** \( \log^* n \)
Log*-Time Coloring with Label Manipulation

Initially ID = label of node $v = \text{color } c_v$
Log*-Time Coloring with Label Manipulation

Initially ID = label of node \( v \) = color \( c_v \)

Log \( n \) bits to represent \( n \) unique IDs

Initially ID = label of node \( v \) = color \( c_v \)

Unique IDs \( \rightarrow \) legal (but expensive) coloring!
Log*-Time Coloring with Label Manipulation

Algorithm: in round $i$, node $v$:
1. Send my $c_v$ to children (in parallel!)
2. Receive parent ID/color $c_p$
Log*-Time Coloring with Label Manipulation

Algorithm: in round i, node v:
1. Send my $c_v$ to children (in parallel!)
2. Receive parent ID/color $c_p$
3. Let $i$ be the smallest index where $c_v$ and $c_p$ differ (from right, binary)
4. My new $c_v = i \parallel c_v(i)$

ID = color for next round: the position!
Log*-Time Coloring with Label Manipulation

Example:

Differ at position 5 = (0101)_2

Differ at position 8 = (1000)_2

Round 1

Algorithm: in round \(i\), node \(v\):
1. Send my \(c_v\) to children (in parallel!)
2. Receive parent ID/color \(c_p\)
3. Let \(i\) be the smallest index where \(c_v\) and \(c_p\) differ (from right, binary)
4. My new \(c_v = i \| c_v(i)\)
Log*-Time Coloring with Label Manipulation

**Algorithm:** in round $i$, node $v$:
1. Send my $c_v$ to children (in parallel!)
2. Receive parent ID/color $c_p$
3. Let $i$ be the smallest index where $c_v$ and $c_p$ differ (from right, binary)
4. My new $c_v = i \parallel c_v(i)$
Log*-Time Coloring with Label Manipulation

Round 1

How long are the new IDs?

Describing position in x-bit string takes log x bits, so: loglog n bits

3. Let \( i \) be the smallest index where \( c_v \) and \( c_p \) differ (from right, binary)
4. My new \( c_v = i \parallel c_v(i) \)
Log*-Time Coloring with Label Manipulation

Describing position in x-bit string takes log x bits, so: loglog n bits

3. Let i be the smallest index where \( c_v \) and \( c_p \) differ (from right, binary)
4. My new \( c_v = i \parallel c_v(i) \) +1 bit

Round 1

How long are the new IDs?
Algorithm: in round i, node v:
1. Send my \( c_v \) to children (in parallel!)
2. Receive parent ID/color \( c_p \)
3. Let \( i \) be the smallest index where \( c_v \) and \( c_p \) differ (from right, binary)
4. My new \( c_v = i || c_v(i) \)
Log*-Time Coloring with Label Manipulation

**Algorithm:** in round $i$, node $v$:
1. Send my $c_v$ to children (in parallel!)
2. Receive parent ID/color $c_p$
3. Let $i$ be the smallest index where $c_v$ and $c_p$ differ (from right, binary)
4. My new $c_v = i \parallel c_v(i)$
Log*-Time Coloring with Label Manipulation

How long are the new IDs?

Differ at position 3 = (11)_2

Round 2

Algorithm: in round i, node v:
1. Send my c_v to children (in parallel!)
2. Receive parent ID/color c_p
3. Let i be the smallest index where c_v and c_p differ (from right, binary)
4. My new c_v = i \| c_v(i)
Log*-Time Coloring with Label Manipulation

How long are the new IDs?

Differ at position 3 = (11)_2

Round 2

Describing position in x-bit string takes log x bits, so: loglogloglog n bits

1. Send my c_v to children (in parallel!)
2. Receive parent ID/color c_p
3. Let i be the smallest index where c_v and c_p differ (from right, binary)
4. My new c_v = i || c_v(i) + 1 bit

Algorithm:
in round i,
node v:

1. Send my c_v to children (in parallel!)
2. Receive parent ID/color c_p
3. Let i be the smallest index where c_v and c_p differ (from right, binary)
4. My new c_v = i || c_v(i) + 1 bit
Algorithm: in round $i$, node $v$:
1. Send my $c_v$ to children (in parallel!)
2. Receive parent ID/color $c_p$
3. Let $i$ be the smallest index where $c_v$ and $c_p$ differ (from right, binary)
4. My new $c_v = i \parallel c_v(i)$
Algorithm: in round $i$, node $v$:

1. Send my $c_v$ to children (in parallel!)
2. Receive parent ID/color $c_p$
3. Let $i$ be the smallest index where $c_v$ and $c_p$ differ (from right, binary)
4. My new $c_v = i \parallel c_v(i)$
Analysis

- How long does it take until $O(1)$ colors?

- Why is coloring always legal?
Analysis

- How long does it take until $O(1)$ colors?
  - # bits/colors reduced by a log-factor in each round
  - The definition of log*

  $\log^* n$: How many times do I have to $\log x$ until $<2$?

- Why is coloring always legal?

  **Algorithm:** My new $c_v = i || c_v(i)$
Analysis

- How long does it take until $O(1)$ colors?
  - # bits/colors reduced by a log-factor in each round
  - The definition of $\log^*$!

  $\log^* n$: How many times do I have to $\log x$ until <2?

- Why is coloring always legal?

  **Algorithm:** My new $c_v = i || c_v(i)$

  By contradiction: To get the same ID as my father, I need to differ at same position from father as father from grandfather. But then last bit must be different: there I took my own bit (and father will do the same with his different bit)!
6-Colors

Assume: legal initial coloring, root with label $c_v=0$
Each other node $v$ does (in parallel):
  Send $c_v$ to kids
Repeat (until $c_w \in \{0,\ldots,5\}$ for all $w$):
  1. Receive $c_p$ from parent
  2. Interpret $c_v/c_p$ as little-endian bitstrings $c(k)\ldots c(1)c(0)$
  3. Let $i$ be smallest index where $c_v$ and $c_p$ differ
  4. New label is: $i||c_v(i)$
  5. Send $c_v$ to kids
6-Colors

Assume: legal initial coloring, root with label c_0
Each other node v does (in parallel):
  Send c_v to kids
Repeat (until c_w ∈ {0,...,5} for all w):
  1. Receive c_p from parent
  2. Interpret c_v/c_p as little-endian bitstrings c(k)...c(1)c(0)
  3. Let i be smallest index where c_v and c_p differ
  4. New label is: i∥c_v(i)
  5. Send c_v to kids

Note: we stop if color in {0,...,5}: why?
Could I go down to 2-bit colors, i.e., \{0, ..., 3\}: No, requires 2 bits to address index where they differ, plus adding the „difference-bit“ gives more than two bits.

Note: we stop if color in \{0, ..., 5\}: why?
Summary of Algorithm

6-Colors

Assume: legal initial coloring, meet with label e.g. 0.
Each other node v does (in parallel):
Send $c_v$ to kids
Repeat (until $c_w \in \{0, \ldots, 5\}$ for all w):
1. Receive $c_p$ from parent
2. Interpret $c_v/c_p$ as little-endian bitstrings $c(k)\ldots c(1)c(0)$
   - position index (0,1,2) requires two bits, plus one "difference-bit" gives three again

Could I go down to 2-bit colors, i.e., $\{0, \ldots, 3\}$: No, requires 2 bits to address index where they differ, plus adding the "difference-bit" gives more than two bits.

For 3-bit colors $\{0, \ldots, 7\}$ this still works: e.g., $7=(111)_2$ can be described with 3 bits, and position index (0,1,2) requires two bits, plus one "difference-bit" gives three again

Note: we stop if color in $\{0, \ldots, 5\}$: why?
Summary of Algorithm

6-Colors

Assume: legal initial coloring, root with labels p?
Each other node v does (in parallel)
Send c_v to kids
Repeat (until c_w ∈ {0,...,5} for all w):
1. Receive c_p from parent
2. Interpret c_v/c_p as little-endian bitstrings c(k)...c(1)c(0)
3. Interpolate, e.g., if index where c_v and c_p differ

Could I go down to 2-bit colors, i.e., {0,...,3}? No, requires 2 bits to address
index where they differ, plus adding the “difference-bit” gives more than two bits.

But actually colors 110 (for color “6“) and 111 (for color “7“) are not
needed, as we can do another round! IDs of three bits can only differ
at positions 00 (for “0“), 01 (for “1“), 10 (for “2“).

For 3-bit colors {0,...,7} this still works: e.g., 7=(111)_2 can be described with 3 bits, and
position index (0,1,2) requires two bits, plus one “difference-bit” gives three again.
With 6-COLORS algorithm we can get down to 6 colors.

What about improving it to 2 colors?
With 6-COLORS algorithm we can get down to 6 colors.

What about improving it to 2 colors?

Impossible: takes linear time.
What about 3 colors?
Let us note a simple trick: shift colors down by one level makes siblings „independent“.
And preserves legal coloring…
Let us note a simple trick: shift colors down by one level makes siblings „independent“. And preserves legal coloring…

Result: all my descendants have same color! At most 2 colors are occupied: father and descendants! 3rd color free!
Observation: Shift Down

Let us note a simple trick: shift colors down by one level makes siblings „independent“.
And preserves legal coloring…

Shift Down

Each node $v$ concurrently does:
recolor $v$ with color of parent

Result: all my descendants have same color! At most 2 colors are occupied: father and descendants! 3rd color free!
Each other node $v$ does (in parallel):
1. Run "6-Colors" for $\log^*(n)$ rounds
2. For $x=5,4,3$:
   1. Perform **Shift Down**
   2. If $(c_v=x)$ choose new color $c_v \in \{0,1,2\}$ according "first free" principle
6-to-3

Each other node $v$ does (in parallel):
1. Run "6-Colors" for $\log^*(n)$ rounds
2. For $x=5,4,3$:
   1. Perform **Shift Down**
   2. If $(c_v=x)$ choose new color $c_v \in \{0,1,2\}$ according "first free" principle

Why still $\log^* n$ time?
Each other node \( v \) does (in parallel):
1. Run "6-Colors" for \( \log^*(n) \) rounds
2. For \( x = 5,4,3 \):
   1. Perform **Shift Down**
   2. If \( (c_v = x) \) choose new color \( c_v \epsilon \{0,1,2\} \) according "**first free**" principle

**Why still \( \log^* n \) time?**

**Just 3 more rounds!**
6-to-3

Each node $v$ does (in parallel):

1. Run "6-Colors" for $\log^*(n)$ rounds
2. For $x=5,4,3$:
   1. Perform **Shift Down**
   2. If $(c_v = x)$ choose new color $c_v \in \{0,1,2\}$ according to the "first free" principle

Why not do in same step?
Each node \( v \) does (in parallel):

1. Run "6-Colors" for \( \log^*(n) \) rounds
2. For \( x=5,4,3 \):
   1. Perform \textbf{Shift Down}
   2. If \( (c_v=x) \) choose new color \( c_v \in \{0,1,2\} \) according to "first free" principle

Why not do in same step?

- Could be harmful: same 3rd color!
- Need to do it for independent sets.

E.g. 5 \( \rightarrow \) 1

E.g. 4 \( \rightarrow \) 1
Example: Shift Down + Drop Color 4

Siblings no longer have same color: must do shift down again first!
Example: 6-to-3

new color for 5: first free

Careful: cannot recolor 4 at same time!
Remark: Optimality

One can show that no local algorithm can 3-color a graph faster than in $O(\log^* n)$. 
Remark: Optimality

One can show that no local algorithm can 3-color a graph faster than in $O(\log^* n)$.

In fact:
in 0 rounds: $\geq n$ colors
in 1 round: $\geq \log n$ colors
in 2 rounds: $\geq \log\log n$ colors
etc.!
Remark: Optimality

One can show that no local algorithm can 3-color a graph faster than in $O(\log^* n)$.

**Proof idea:** Recall the elephant!

In fact:
in 0 rounds: $\geq n$ colors
in 1 round: $\geq \log n$ colors
in 2 rounds: $\geq \log\log n$ colors etc.!

A local coloring algorithm can be seen as a function:

$$f: \text{neighborhood} \rightarrow \text{color}$$

A deterministic algorithm needs to decide in the same way given same neighborhood: risk illegal coloring. Only with communication neighborhoods start look different and require less colors.
Lower Bound

Set of neighborhoods → Local coloring algo → Vertex coloring

ALG 4 → ALG 4

4

7

ALG 7 → ALG 7
Lower Bound

Can reduce problem of finding lower bound to determine chromatic number of special neighborhood dependency graphs.
Concluding Remarks

Can we reduce to 2 colors?

Not without increasing runtime significantly!
(Linear time, more than exponentially worse!)

Simple on purpose: results more general!

log* runtime is also possible on more general graphs
Many results: see ACM PODC conference!
Where can I learn more?

- Distributed Computing book by David Peleg
- Lecture notes by Roger Wattenhofer ETH Zurich
- ACM Survey by Jukka Suomela
- Research: ACM PODC Conference