Chapter 1

Vertex Coloring

1.1 Problem & Model

Vertex coloring is an infamous graph theory problem. It is also a useful toy example to see the style of this course already in the first lecture. Vertex coloring does have quite a few practical applications, for example in the area of wireless networks where coloring is the foundation of so-called TDMA MAC protocols. Generally speaking, vertex coloring is used as a means to break symmetries, one of the main themes in distributed computing. In this chapter we will not really talk about vertex coloring applications, but treat the problem abstractly. At the end of the class you probably learned the fastest (but not constant!) algorithm ever! Let us start with some simple definitions and observations.

Problem 1.1 (Vertex Coloring)**.** *Given an undirected graph $G = (V, E)$, assign a color c_u to each vertex $u \in V$ such that the following holds: $e = (v, w) \in E \Rightarrow c_v \neq c_w$.*

Remarks:

- Throughout this course, we use the terms *vertex* and *node* interchangeably.
- The application often asks us to use few colors! In a TDMA MAC protocol, for example, less colors immediately imply higher throughput. However, in distributed computing we are often happy with a solution which is sub-optimal. There is a tradeoff between the optimality of a solution (efficacy), and the work/time needed to compute the solution (efficiency).

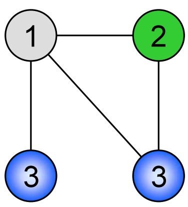

Figure 1.1: 3-colorable graph with a valid coloring.

Assumption 1.2 (Node Identifiers)**.** *Each node has a unique identifier, e.g., its IP address. We usually assume that each identifier consists of only* $\log n$ *bits if the system has* n *nodes.*

Remarks:

- Sometimes we might even assume that the nodes exactly have identifiers $1, \dots, n$.
- It is easy to see that node identifiers (as defined in Assumption 1.2) solve the coloring problem 1.1, but not very well (essentially requiring n colors). How many colors are needed is a well-studied problem.

Definition 1.3 (Chromatic Number)**.** *Given an undirected Graph* $G = (V, E)$*, the* chromatic number $\chi(G)$ *is the minimum number of colors to solve Problem 1.1.*

To get a better understanding of the vertex coloring problem, let us first look at a simple non-distributed ("centralized") vertex coloring algorithm:

Algorithm 1 Greedy Sequential

```
while ∃ uncolored vertex v do
    color v with the minimal color (number) that does not conflict with the
    already colored neighbors
end while
```

Definition 1.4 (Degree)**.** *The number of neighbors of a vertex* v*, denoted by* $\delta(v)$*, is called the* degree *of* v*. The maximum degree vertex in a graph* G *defines the graph degree* $\Delta(G) = \Delta$*.*

Theorem 1.5 (Analysis of Algorithm 1)**.** *The algorithm is correct and terminates in* n *"steps". The algorithm uses at most* $\Delta + 1$ *colors.*

Proof: Correctness and termination are straightforward. Since each node has at most Δ neighbors, there is always at least one color free in the range $\{1, \dots, \Delta + 1\}$.

Remarks:

- In Definition 1.7 we will see what is meant by "step".
- For many graphs coloring can be done with much less than $\Delta + 1$ colors.
- This algorithm is not distributed at all; only one processor is active at a time. Still, maybe we can use the simple idea of Algorithm 1 to define a distributed coloring subroutine that may come in handy later.

Now we are ready to study distributed algorithms for this problem. The following procedure can be executed by every vertex v in a distributed coloring algorithm. The goal of this subroutine is to improve a given initial coloring.

Procedure 2 First Free

Require: Node Coloring {e.g., node IDs as defined in Assumption 1.2}
Give v the smallest admissible color {i.e., the smallest node color not used by any neighbor}

Remarks:

- With this subroutine we have to make sure that two adjacent vertices are not colored at the same time. Otherwise, the neighbors may at the same time conclude that some small color c is still available in their neighborhood, and then at the same time decide to choose this color c.

Definition 1.6 (Synchronous Distributed Algorithm)**.** *In a synchronous algorithm, nodes operate in synchronous rounds. In each round, each processor executes the following steps:*

1. *Do some local computation (of reasonable complexity).*
2. *Send messages to neighbors in graph (of reasonable size).*
3. *Receive messages (that were sent by neighbors in step 2 of the same round).*

Remarks:

- Any other step ordering is fine.
- What does "reasonable" mean in this context? We are somewhat flexible here, and different model variants exist. Generally, we will deal with algorithms that only do very simple computations (a comparison, an addition, etc.). Exponential-time computation is usually considered cheating in this context. Similarly, sending a message with a node ID, or a value is considered okay, whereas sending really long messages is considered cheating. We will have more exact definitions later, when we need them.

Algorithm 3 Reduce

Assume that initially all nodes have IDs (Assumption 1.2)
Each node v executes the following code
node v sends its ID to all neighbors
node v receives IDs of neighbors
while node v has an uncolored neighbor with higher ID **do**
  node v sends "undecided" to all neighbors
  node v receives new decisions from neighbors
end while
node v chooses a free color using subroutine **First Free** (Procedure 2)
node v informs all its neighbors about its choice

Definition 1.7 (Time Complexity)**.** *For synchronous algorithms (as defined in 1.6) the* time complexity *is the number of rounds until the algorithm terminates.*

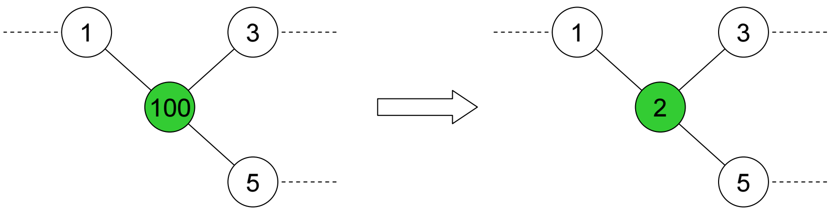

Figure 1.2: Vertex 100 receives the lowest possible color.

Remarks:

- The algorithm terminates when the last processor has decided to terminate.
- To guarantee correctness the procedure requires a legal input (i.e., pairwise different node IDs).

Theorem 1.8 (Analysis of Algorithm 3)**.** *Algorithm 3 is correct and has time complexity n. The algorithm uses at most $\Delta + 1$ colors.*

Remarks:

- Quite trivial, but also quite slow.
- However, it seems difficult to come up with a fast algorithm.
- Maybe it's better to first study a simple special case, a tree, and then go from there.

1.2 Coloring Trees

Lemma 1.9. $\chi(Tree) \leq 2$

Constructive Proof: If the distance of a node to the root is odd (even), color it 1 (0). An odd node has only even neighbors and vice versa. If we assume that each node knows its parent (root has no parent) and children in a tree, this constructive proof gives a very simple algorithm:

Algorithm 4 Slow Tree Coloring

Color the root 0, root sends 0 to its children
Each node v concurrently executes the following code:
if node v receives a message x (from parent) **then**
  node v chooses color $c_v = 1 - x$
  node v sends c_v to its children (all neighbors except parent)
end if

Remarks:

- With the proof of Lemma 1.9, Algorithm 4 is correct.
- How can we determine a root in a tree if it is not already given? We will figure that out later.
- The time complexity of the algorithm is the height of the tree.
- If the root was chosen unfortunately, and the tree has a degenerated topology, the time complexity may be up to n, the number of nodes.
- Also, this algorithm does not need to be synchronous . . . !

Definition 1.10 (Asynchronous Distributed Algorithm)**.** *In the asynchronous model, algorithms are event driven ("upon receiving message . . . , do . . . "). Processors cannot access a global clock. A message sent from one processor to another will arrive in finite but unbounded time.*

Remarks:

- The asynchronous model and the synchronous model (Definition 1.6) are the cornerstone models in distributed computing. As they do not necessarily reflect reality there are several models in between synchronous and asynchronous. However, from a theoretical point of view the synchronous and the asynchronous model are the most interesting ones (because every other model is in between these extremes).
- Note that in the asynchronous model, messages that take a longer path may arrive earlier.

Definition 1.11 (Time Complexity)**.** *For asynchronous algorithms (as defined in 1.6) the* time complexity *is the number of time units from the start of the execution to its completion in the worst case (every legal input, every execution scenario), assuming that each message has a delay of at most one time unit.*

Remarks:

- You cannot use the maximum delay in the algorithm design. In other words, the algorithm has to be correct even if there is no such delay upper bound.

Definition 1.12 (Message Complexity)**.** *The* message complexity *of a synchronous or asynchronous algorithm is determined by the number of messages exchanged (again every legal input, every execution scenario).*

Theorem 1.13 (Analysis of Algorithm 4)**.** *Algorithm 4 is correct. If each node knows its parent and its children, the (asynchronous) time complexity is the tree height which is bounded by the diameter of the tree; the message complexity is* $n - 1$ *in a tree with* n *nodes.*

Remarks:

- In this case the asynchronous time complexity is the same as the synchronous time complexity.
- Nice trees, e.g., balanced binary trees, have logarithmic height, that is we have a logarithmic time complexity.
- This algorithm is not very exciting. Can we do better than logarithmic?

The following algorithm terminates in $\log^* n$ time. Log-Star?! That's the number of logarithms (to the base 2) you need to take to get down to at least 2, starting with n:

Definition 1.14 (Log-Star)**.**
$\forall x \leq 2: \ \log^* x := 1 \qquad \forall x > 2: \ \log^* x := 1 + \log^*(\log x)$

Remarks:

- Log-star is an amazingly slowly growing function. Log-star of all the atoms in the observable universe (estimated to be 10^{80}) is 5. There are functions which grow even more slowly, such as the inverse Ackermann function, however, the inverse Ackermann function of all the atoms is 4. So log-star increases indeed very slowly!

Here is the idea of the algorithm: We start with color labels that have $\log n$ bits. In each synchronous round we compute a new label with exponentially smaller size than the previous label, still guaranteeing to have a valid vertex coloring! But how are we going to do that?

Algorithm 5 "6-Color"

Assume that initially the vertices are legally colored. Using Assumption 1.2 each label only has $\log n$ bits
The root assigns itself the label 0.
Each other **node** v executes the following code (synchronously in parallel)
send c_v to all children
repeat
  receive c_p from parent
  interpret c_v and c_p as little-endian bit-strings: $c(k), \ldots, c(1), c(0)$
  let i be the smallest index where c_v and c_p differ
  the new label is i (as bitstring) followed by the bit $c_v(i)$ itself
  send c_v to all children
until $c_w \in \{0, \ldots, 5\}$ for all nodes w

Example:

Algorithm 5 executed on the following part of a tree:

Grand-parent	0010110000	$\rightarrow$	10010	$\rightarrow$	$\ldots$
Parent	1010010000	$\rightarrow$	01010	$\rightarrow$	111
Child	0110010000	$\rightarrow$	10001	$\rightarrow$	001

Theorem 1.15. *Algorithm 5 terminates in* $\log^* n$ *time.*

Remarks:

- Colors 11$*$ (in binary notation, i.e., 6 or 7 in decimal notation) will not be chosen, because the node will then do another round. This gives a total of 6 colors (i.e., colors 0,. . . , 5).
- Can one reduce the number of colors in only constant steps? Note that Algorithm 3 does not work (since the degree of a node can be much higher than 6)! For fewer colors we need to have siblings monochromatic!
- Before we explore this problem we should probably have a second look at the end game of the algorithm, the UNTIL statement. Is this algorithm truly local?! Let's discuss!

Algorithm 6 Shift Down

Root chooses a new (different) color from $\{0, 1, 2\}$
Each other **node** v concurrently executes the following code:
Recolor v with the color of parent

Lemma 1.16 (Analysis of Algorithm 6)**.** *Algorithm 6 preserves coloring legality; also siblings are monochromatic.*

Now Algorithm 3 (Reduce) can be used to reduce the number of used colors from six to three.

Algorithm 7 Six-2-Three

Each node v concurrently executes the following code:
Run Algorithm 5 for $\log^* n$ rounds.
for $x = 5, 4, 3$ **do**
  Perform subroutine Shift down (Algorithm 6)
  if $c_v = x$ **then**
    choose new color $c_v \in \{0, 1, 2\}$ using subroutine **First Free** (Algorithm 2)
  end if
end for

Theorem 1.17 (Analysis of Algorithm 7)**.** *Algorithm 7 colors a tree with three colors in time $\mathcal{O}(\log^* n)$.*

Remarks:

- The term $\mathcal{O}()$ used in Theorem 1.15 is called "big O" and is often used in distributed computing. Roughly speaking, $\mathcal{O}(f)$ means "in the order of f, ignoring constant factors and smaller additive terms." More formally, for two functions f and g, it holds that $f \in O(g)$ if there are constants x_0 and c so that $|f(x)| \le c|g(x)|$ for all $x \ge x_0$. For an elaborate discussion on the big O notation we refer to other introductory math or computer science classes.

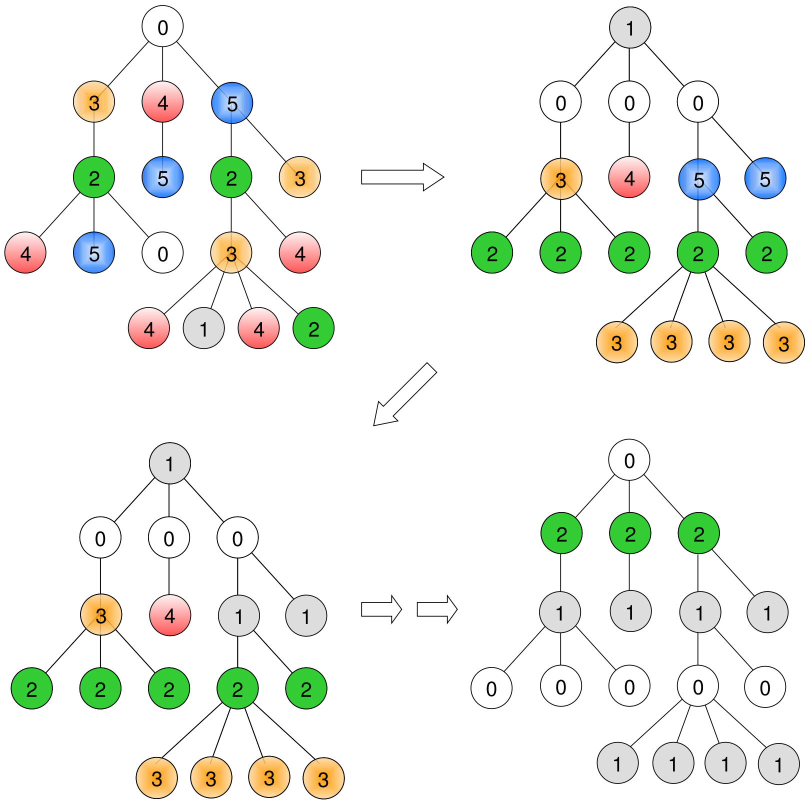

Figure 1.3: Possible execution of Algorithm 7.

- As one can easily prove, a fast tree-coloring with only 2 colors is more than exponentially more expensive than coloring with 3 colors. In a tree degenerated to a list, nodes far away need to figure out whether they are an even or odd number of hops away from each other in order to get a 2-coloring. To do that one has to send a message to these nodes. This costs time linear in the number of nodes.

- The idea of this algorithm can be generalized, e.g., to a ring topology. Also a general graph with constant degree Δ can be colored with $\Delta + 1$ colors in $\mathcal{O}(\log^* n)$ time. The idea is as follows: In each step, a node compares its label to each of its neighbors, constructing a logarithmic difference-tag as in 6-color (Algorithm 5). Then the new label is the concatenation of all the difference-tags. For constant degree Δ, this gives a 3Δ-label in $\mathcal{O}(\log^* n)$ steps. Algorithm 3 then reduces the number of colors to $\Delta + 1$ in $2^{3\Delta}$ (this is still a constant for constant Δ!) steps.

- Unfortunately, coloring a general graph is not yet possible with this technique. We will see another technique for that in Chapter 7. With this technique it is possible to color a general graph with $\Delta + 1$ colors in $\mathcal{O}(\log n)$ time.

- A lower bound shows that many of these log-star algorithms are asymptotically (up to constant factors) optimal. We will also see that later.

Chapter Notes

The basic technique of the log-star algorithm is by Cole and Vishkin [CV86]. The technique can be generalized and extended, e.g., to a ring topology or to graphs with constant degree [GP87, GPS88, KMW05]. Using it as a subroutine, one can solve many problems in log-star time. For instance, one can color so-called growth bounded graphs (a model which includes many natural graph classes, for instance unit disk graphs) asymptotically optimally in $\mathcal{O}(\log^* n)$ time [SW08]. Actually, Schneider et al. show that many classic combinatorial problems beyond coloring can be solved in log-star time in growth bounded and other restricted graphs.

In a later chapter we learn a $\Omega(\log^* n)$ lower bound for coloring and related problems [Lin92]. Linial's paper also contains a number of other results on coloring, e.g., that any algorithm for coloring d-regular trees of radius r that run in time at most $2r/3$ require at least $\Omega(\sqrt{d})$ colors.

For general graphs, later we will learn fast coloring algorithms that use a maximal independent sets as a base. Since coloring exhibits a trade-off between efficacy and efficiency, many different results for general graphs exist, e.g., [PS96, KSOS06, BE09, Kuh09, SW10, BE11b, KP11, BE11a].

Some parts of this chapter are also discussed in Chapter 7 of [Pel00], e.g., the proof of Theorem 1.15.

Bibliography

[BE09] Leonid Barenboim and Michael Elkin. Distributed (delta+1)-coloring in linear (in delta) time. In *41st ACM Symposium On Theory of Computing (STOC)*, 2009.

[BE11a] Leonid Barenboim and Michael Elkin. Combinatorial Algorithms for Distributed Graph Coloring. In *25th International Symposium on DIStributed Computing*, 2011.

[BE11b] Leonid Barenboim and Michael Elkin. Deterministic Distributed Vertex Coloring in Polylogarithmic Time. *J. ACM*, 58(5):23, 2011.

[CV86] R. Cole and U. Vishkin. Deterministic coin tossing and accelerating cascades: micro and macro techniques for designing parallel algorithms. In *18th annual ACM Symposium on Theory of Computing (STOC)*, 1986.

[GP87] Andrew V. Goldberg and Serge A. Plotkin. Parallel (Δ+1)-coloring of constant-degree graphs. *Inf. Process. Lett.*, 25(4):241–245, June 1987.

[GPS88] Andrew V. Goldberg, Serge A. Plotkin, and Gregory E. Shannon. Parallel Symmetry-Breaking in Sparse Graphs. *SIAM J. Discrete Math.*, 1(4):434–446, 1988.

[KMW05] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. On the Locality of Bounded Growth. In *24th ACM Symposium on the Principles of Distributed Computing (PODC), Las Vegas, Nevada, USA*, July 2005.

[KP11] Kishore Kothapalli and Sriram V. Pemmaraju. Distributed graph coloring in a few rounds. In *30th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, 2011.

[KSOS06] Kishore Kothapalli, Christian Scheideler, Melih Onus, and Christian Schindelhauer. Distributed coloring in $O(\sqrt{\log n})$ Bit Rounds. In *20th international conference on Parallel and Distributed Processing (IPDPS)*, 2006.

[Kuh09] Fabian Kuhn. Weak graph colorings: distributed algorithms and applications. In *21st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2009.

[Lin92] N. Linial. Locality in Distributed Graph Algorithms. *SIAM Journal on Computing*, 21(1)(1):193–201, February 1992.

[Pel00] David Peleg. *Distributed computing: a locality-sensitive approach.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[PS96] Alessandro Panconesi and Aravind Srinivasan. On the Complexity of Distributed Network Decomposition. *J. Algorithms*, 20(2):356–374, 1996.

[SW08] Johannes Schneider and Roger Wattenhofer. A Log-Star Distributed Maximal Independent Set Algorithm for Growth-Bounded Graphs. In *27th ACM Symposium on Principles of Distributed Computing (PODC), Toronto, Canada*, August 2008.

[SW10] Johannes Schneider and Roger Wattenhofer. A New Technique For Distributed Symmetry Breaking. In *29th Symposium on Principles of Distributed Computing (PODC), Zurich, Switzerland*, July 2010.