Chapter 1

Vertex Coloring

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 algorithm ever! Let us start with some simple definitions and observations.

1.1 Problem & Model

Problem 1.1 (Vertex Coloring)**.** *Given an undirected graph $G = (V, E)$, assign a color c_v to each vertex $v \in V$ 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 suboptimal. There is a tradeoff between the optimality of a solution (efficacy), and the work/time needed to compute the solution (efficiency).

Assumption 1.3 (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.*

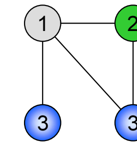

Figure 1.2: 3-colorable graph with a valid coloring.

Remarks:

- Sometimes we might even assume that the nodes exactly have identifiers $1, \ldots, n$.
- It is easy to see that node identifiers (as defined in Assumption 1.3) solve the coloring problem 1.1, but using n colors is not exciting. How many colors are needed is a well-studied problem:

Definition 1.4 (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 1.5 Greedy Sequential

```
1: while there is an uncolored vertex v do
2:    color v with the minimal color (number) that does not conflict with the
      already colored neighbors
3: end while
```

Definition 1.6 (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 1.7. *Algorithm 1.5 is correct and terminates in n "steps". The algorithm uses at most $\Delta + 1$ colors.*

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

Remarks:

- In Definition 1.11 we will see what is meant by "step".
- Sometimes $\chi(G) \ll \Delta + 1$.

Definition 1.8 (Synchronous Distributed Algorithm)**.** *In a synchronous distributed algorithm, nodes operate in synchronous rounds. In each round, each node executes the following steps:*

1. *Send messages to neighbors in graph (of reasonable size).*

2. *Receive messages (that were sent by neighbors in step 1 of the same round).*

3. *Do some local computation (of reasonable complexity).*

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 fishy. We will have more exact definitions later, when we need them.

- We can build a distributed version of Algorithm 1.5:

Algorithm 1.9 Reduce

```
1: Assume that initially all nodes have IDs
2: Each node v executes the following code:
3: node v sends its ID to all neighbors
4: node v receives IDs of neighbors
5: while node v has an uncolored neighbor with higher ID do
6:    node v sends "undecided" to all neighbors
7:    node v receives new decisions from neighbors
8: end while
9: node v chooses the smallest admissible free color
10: node v informs all its neighbors about its choice
```

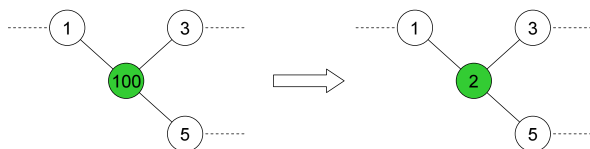

Figure 1.10: Vertex 100 receives the lowest possible color.

Definition 1.11 (Time Complexity)**.** *For synchronous algorithms (as defined in 1.8) the* time complexity *is the number of rounds until the algorithm terminates. The algorithm terminates when the last node terminates.*

Theorem 1.12. *Algorithm 1.9 is correct and has time complexity n. The algorithm uses at most $\Delta + 1$ colors.*

Proof. Nodes choose colors that are different from their neighbors, and no two neighbors choose concurrently. In each round at least one node chooses a color, so we are done after at most n rounds. □

Remarks:

- In the worst case, this algorithm is still not better than sequential.

- Moreover, 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 1.13. $\chi(Tree) \leq 2$

Proof. Call some node the root of the tree. 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. □

Remarks:

- 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 1.14 Slow Tree Coloring

```
1: Color the root 0, root sends 0 to its children
2: Each node v concurrently executes the following code:
3: if node v receives a message c_p (from parent) then
4:    node v chooses color c_v = 1 - c_p
5:    node v sends c_v to its children (all neighbors except parent)
6: end if
```

Theorem 1.15. *Algorithm 1.14 is correct. If each node knows its parent and its children, the time complexity is the tree height which is bounded by the diameter of the tree.*

Remarks:

- 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.

- Nice trees, e.g., balanced binary trees, have logarithmic height, that is we have a logarithmic time complexity.

- However, if the tree has a degenerated topology, the time complexity may again be up to n, the number of nodes.

- This algorithm is not very exciting. Can we do better than logarithmic?

Here is the idea of the algorithm: We start with color labels that have $\log n$ bits. In each round we compute a new label with exponentially smaller size than the previous label, still guaranteeing to have a valid vertex coloring! The 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 2. Formally:

Definition 1.16 (Log-Star)**.**
$\forall x \leq 2: \ \log^* x := 1 \quad \forall x > 2: \ \log^* x := 1 + \log^*(\log x)$

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. So log-star increases indeed very slowly! There are functions which grow even more slowly, such as the inverse Ackermann function, however, the inverse Ackermann function of all the atoms is already 4.

Algorithm 1.17 "6-Color"

1: Assume that initially the nodes have IDs of size $\log n$ bits
2: The root assigns itself the label 0
3: **Each** other **node** v executes the following code
4: send own color c_v to all children
5: **repeat**
6:    receive color c_p from parent
7:    interpret c_v and c_p as bit-strings
8:    let i be the index of the smallest bit where c_v and c_p differ
9:    the new label is i (as bitstring) followed by the ith bit of c_v
10:    send c_v to all children
11: **until** $c_w \in \{0, \dots, 5\}$ for all nodes w

Example:

Algorithm 1.17 executed on the following part of a tree:

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

Theorem 1.18. *Algorithm 1.17 terminates in $\log^* n + k$ time, where k is a constant independent of n.*

Proof. We need to show that parent p and child c always have different colors. Initially, this is true, since all nodes start out with their unique ID. In a round, let i be the smallest index where child c has a different bit from parent p. If parent p differs in a different index bit $j \neq i$ from its own parent, parent and child will compute different colors in that round. On the other hand, if $j = i$, the symmetry is broken by p having a different bit at index i.

Regarding runtime, note that the size of the largest color shrinks dramatically in each round, apart from the symmetry-breaking bit, exactly as a logarithmic function. With some (tedious and boring) machinery, one can show that indeed every node will have a color in the range $\{0, \dots, 5\}$ in $\log^* n + k$ rounds. □

Remarks:

- Let us have a closer look at the end game of the algorithm. 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).
- What about that last line of the loop? How do the nodes know that all nodes now have a color in the range $\{0, \dots, 5\}$? The answer to this question is surprisingly complex. One may hardwire the number of rounds into the until statement, such that all nodes execute the loop for exactly the same number of rounds. However, in order to do so, all nodes need to know n, the number of nodes, which is ugly. There are (non-trivial) solutions where nodes do not need to know n, see exercises.
- Can one reduce the number of colors? Note that Algorithm 1.9 does not work (since the degree of a node can be much higher than 6)! For fewer colors we need to have siblings monochromatic!

Algorithm 1.19 Shift Down

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

Lemma 1.20. *Algorithm 1.19 preserves coloring legality; also siblings are monochromatic.*

Now Algorithm 1.9 can be used to reduce the number of used colors from 6 to 3.

Algorithm 1.21 Six-2-Three

1: **Each node** v concurrently executes the following code:
2: **for** $x = 5, 4, 3$ **do**
3:    Perform subroutine Shift down (Algorithm 1.19)
4:    **if** $c_v = x$ **then**
5:       choose the smallest admissible new color $c_v \in \{0, 1, 2\}$
6:    **end if**
7: **end for**

Theorem 1.23. *Algorithms 1.17 and 1.21 color a tree with three colors in time $\mathcal{O}(\log^* n)$.*

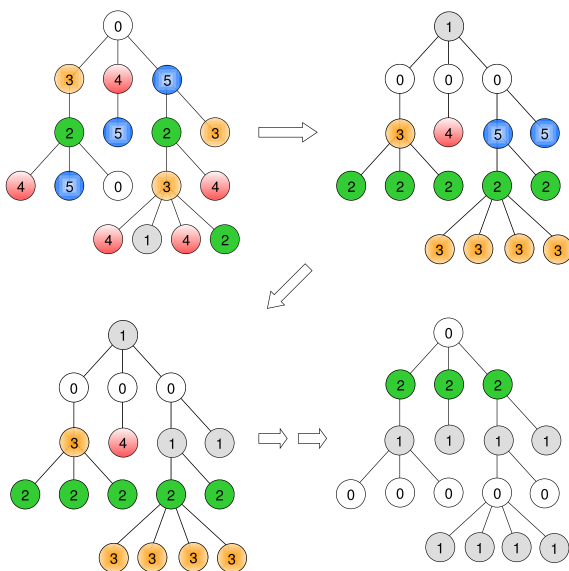

Figure 1.22: Possible execution of Algorithm 1.21.

Remarks:

- The term $\mathcal{O}()$ used in Theorem 1.18 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 $|f(x)| \leq c|g(x)|$ for all $x \geq x_0$. For an elaborate discussion on the big O notation we refer to other introductory math or computer science classes, or Wikipedia.
- 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 Algorithm 1.17. 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 1.9 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 see that later.

Chapter Notes

The basic technique of the log-star algorithm is by Cole and Vishkin [CV86]. A tight bound of $\frac{1}{2}\log^* n$ was proven recently [RS15]. 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, BEPS12, PS13, CPS14, BEK14].

Some parts of this chapter are also discussed in Chapter 7 of [Pel00], e.g., the proof of Theorem 1.18.

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.

[BEK14] Leonid Barenboim, Michael Elkin, and Fabian Kuhn. Distributed (delta+1)-coloring in linear (in delta) time. *SIAM J. Comput.*, 43(1):72–95, 2014.

[BEPS12] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 321–330, 2012.

[CPS14] Kai-Min Chung, Seth Pettie, and Hsin-Hao Su. Distributed algorithms for the lovász local lemma and graph coloring. In *ACM Symposium on Principles of Distributed Computing*, pages 134–143, 2014.

[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.

[PS13] Seth Pettie and Hsin-Hao Su. Fast distributed coloring algorithms for triangle-free graphs. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP*, pages 681–693, 2013.

[RS15] Joel Rybicki and Jukka Suomela. Exact bounds for distributed graph colouring. In *Structural Information and Communication Complexity - 22nd International Colloquium, SIROCCO 2015, Montserrat, Spain, July 14-16, 2015, Post-Proceedings*, pages 46–60, 2015.

[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.