Principles of Distributed Computing
Roger Wattenhofer

Chapter 1

Symmetry Breaking 1: Vertex Coloring

Section 1.1: Introduction

(Useful toy problem to see style of lecture)
(Vertex coloring infamous as tough graph theory problem)

Problem 1.1 [Vertex Coloring]: Given undirected Graph $G = (V,E)$. Assign a color c_u to each vertex $u \in V$ such that if $e = (v,w) \in E$, then $c_v \neq c_w$. Use few colors!

Assumption 1.2 [Node Identifiers]: Each node has a unique identifier (the IP-Address, for example). Sometimes we assume that each identifier is only $O(\log n)$ bits if the system has n nodes. (Sometimes we might even assume that the nodes exactly have identifiers 1, …, n.)

Assumption 1.2 solves Problem 1.1, but badly.

Definition 1.3 [Chromatic Number]: Given an undirected Graph. The chromatic number $\chi(G)$ is the minimum number of colors to solve Problem 1.1.

Algorithm 1.4 [Greedy Sequential]: As long as there are uncolored vertices: Take an arbitrary uncolored vertex v, and color it with the minimal color (number) that does not conflict with the already colored neighbors.

Definition 1.5 [Degree]: The number of neighbors of a vertex v is called the degree of v $\delta(v)$. The maximum degree vertex in a Graph G defines the Graph degree $\Delta(G) = \Delta$.

Theorem 1.6 [Analysis of Algorithm 1.4]: The algorithm is correct and terminates in $O(n)$ steps. The algorithm uses $\Delta+1$ colors.

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

Remarks:
- “Steps”
- 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. But: Use idea of Algorithm 1.4 to define “local” coloring subroutine 1.7

Algorithm 1.7 [First Free]: Color a vertex with the smallest available (not used by neighbors) color.

Remark:
- With this subroutine one cannot color two adjacent vertices at the same time.

Definition 1.8 [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. ("reasonable" complexity)
2. Send messages to neighbors in graph. ("reasonable" size)
3. Receive messages (that were sent by neighbors in step 2 of the same round)

Remark:

- Any other step ordering is fine.

Algorithm 1.9 [Reduce]: Assume that initially the vertices are legally colored with colors in the range {1, …, m} (Assumption 1.2). Then each vertex v executes the following code (in parallel)

```
1. for x from $\Delta$+2 to m do
2.     if $c_v$ = x then
3.         c chooses a free color with Algorithm 1.7 in {1, …, $\Delta$+1}.
4.         c informs neighbors about its choice
```

Definition 1.10 [Time Complexity]: For synchronous algorithms (Definition 1.8) the time complexity is the number of rounds until the algorithm terminates.

Remarks:

- The algorithm terminates when the last processor has decided to terminate.
- With every legal input.

Theorem 1.11 [Analysis of Algorithm 1.9]: Algorithm 1.9 is correct and has time complexity $m-\Delta+1$, that is $O(n)$ with $m = O(n)$. The algorithm uses $\Delta+1$ colors.

Remarks:

- Quite trivial, huh.
- And also damn slow.

Section 1.2: Coloring Trees

Lemma 1.12: $\chi(\text{Tree}) \leq 2$.

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 1.13 [Slow tree coloring]:

- Color the root 0, root sends "0" to children.
- When receiving a message "x" (from parent), a node u chooses color $c_u = 1-x$, and sends "c_u" to its children (all neighbors except parent).

Remarks:

- With the proof of Lemma 1.12, the algorithm 1.13 is correct.
- How can we determine a root in a tree? (Later!)
- The time complexity of the algorithm is the height of the tree.
- When the root was chosen unfortunately, this can be up to the diameter of the tree.
- Wait a second… this algorithm does not need to be synchronous…!

Definition 1.14 [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:

- Probably as unrealistic as the synchronous model (Definition 1.8). There are several models in between synchronous and asynchronous. However, from a theory standpoint the synchronous and the asynchronous model are the most interesting (because every other model is "in-between" these extremes).
- Messages that take a longer path may arrive earlier.

Definition 1.15 [Time Complexity]: For asynchronous algorithms (Definition 1.14) 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 occurs a delay of at most one time unit.

Remark:

- 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 1.16 [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 1.17 [Analysis of Algorithm 1.13]: Algorithm 1.13 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:

- Note that asynchronous time complexity is the same as synchronous time complexity.
- "Nice" trees have logarithmic height, that is O(log n) 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 you have to take to get down to at least 2, starting with n. But we need a bit more colors.)

(Idea: start with color labels of size O(log n). In each synchronous round: make label of logarithmic size by getting essence out of label by reading it as a bitstring!...?)

Algorithm 1.18 [“6-Color”]: Assume that initially the vertices are legally colored with colors in the range {0, …, O(n)} (O(log n) bits; Assumption 1.2). The root assigns itself the label 0 immediately. Each other vertex v executes the following code (in parallel)

```
1. send $c_v$ to all children
2. repeat
3.         receive $c_p$ from parent
4.         interpret $c_v$ and $c_p$ as little-endian bit-strings: c(k), …, c(1), c(0)
5.         let i be the smallest index where $c_v$ and $c_p$ differ
6.         the new label is i (as bitstring) followed by the bit $c_v(i)$ itself.
7.         send $c_v$ to all children
8. until new label has as many bits as previous label
```

Example (only part of a tree):

Grand-parent	1010110000	→	10010	→	…
Parent	1010010000	→	1010	→	111
Child	0110010000	→	10001	→	1

Theorem 1.19 [Analysis of Algorithm 1.18]: Algorithm 1.18 terminates in $\log^* n + O(1)$ time.

Proof: [Peleg 7.3]

Remarks:

- Some nodes might terminate earlier than others…
- Colors 11x (binary, decimal 6 or 7) will not be chosen, because the node will then do another round (since it has reduced the number of bits from 4 to 3). Gives a total of 6 colors.
- Can one reduce the number of colors in only constant steps? 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 1.20 [Shift down]: Concurrently at all vertices: Recolor with the color of parent. Root (as an exception) simply chooses any new color.

Lemma 1.21: Algorithm 1.20 preserves coloring legality; also siblings are monochromatic.

Algorithm 1.22 [Six-2-Three]: For all nodes

```
1. for x from 4 to 6 do
2.         Perform subroutine 1.20 (shift down)
3.         A vertex v colored x chooses as new color using subroutine 1.7 (first free)
```

Theorem 1.23: Algorithm 1.22 colors a tree with three colors in time $O(\log^* n)$.

Corollary 1.24: A general graph with constant degree Δ can be colored with Δ+1 colors in $O(\log^* n)$ time.

Algorithm idea: In each step, a vertex compares its label to each of its neighbors, constructing a logarithmic difference-tag as in 6-color. Then the new label is the concatenation of all the difference-tags. For constant degree Δ, this gives a 2Δ label in $O(\log^* n)$ steps. Algorithm 1.9 (Reduce) then reduces the number of colors to $\Delta+1$ in $2^{2\Delta}$ (still a constant!) steps.

Remark:

- One can color a general graph (without constant degree) with a recursive algorithm with $\Delta+1$ colors in $O(\Delta \log n)$ time.

Theorem 1.25: Coloring a ring with three colors costs time $\Omega(\log^* n)$.

Proof: [Peleg 7.5]