What Can Be Computed Locally?

Michael Kaufmann

Distributed Computing Seminar
ETH Zürich, 25.11.2003

Overview of this Presentation

- Part 1: Introduction
  - What are Local Algorithms?
  - General Results
- Part 2: Two local Algorithms
  - Weak Coloring
  - Formal Dining Philosophers problem

About the Paper

- "What Can Be Computed Locally?"
  - By Moni Naor & Larry Stockmeyer (1993)
  - Main topic of this presentation
- Follow-up paper:
  "Local Computations on Static and Dynamic Graphs"
  - By Alain Mayer, Moni Naor & Larry Stockmeyer (1995)
  - Simplifies some algorithms of the previous paper
    - More "high-level"
    - Dynamic network

Part 1:

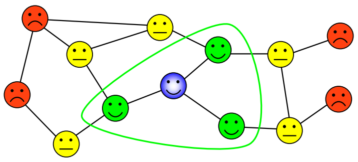
Introduction

&

General Results

Introduction

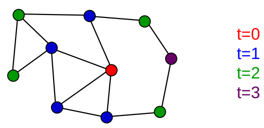
- Locality …



- … is important
  - Runtime is independent of the network size: Constant time t
    - Fast algorithms (parallel computation)
    - Very good scalability
  - Fault-tolerance
    - A computer crash only affects a small part of the network

The Model used (1)

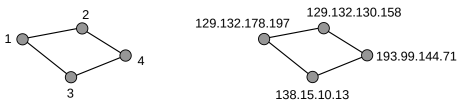
- Network model:
  - At each time unit, a processor may pass messages to each of its neighbors
  - Any computations carried out by individual processors take one time unit



  - Example:
    - If an algorithm takes constant time t=2, the red and the purple processor will never communicate
    - ⇒ In time *t*, every processor can only collect information that lies within radius *t*

The Model used (2)

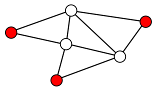
- Every processor has a unique ID
- This makes the processors distinguishable
  - Processors can tell other processors what neighbors they have
  - Examples for IDs:
    - IP address (32-bit-number)
    - MAC address (48-bit-number)
    - Processor serial number (Intel: 96-bit-number)



7

Locally Checkable Labelings (LCLs)

- Algorithms produce a labeling of the graph
  - In a LCL problem, every node is able to check if the labeling is locally correct
  - Examples of such labelings:
    - Vertex/Edge coloring
    - Maximal Independent Set
- Maximal Independent Set
  - This problem is locally checkable:
    - If node v is in the MIS, then no neighbor of v is in the MIS
    - If node v is not in the MIS, then at least one neighbor of v is in the MIS



8

Decidability / Undecidability

- Is it possible to decide if a given LCL problem L can be solved in constant time t?
  - Definition: Let d be the maximum degree of a node in the graph
- Yes, if $d \leq 2$.
- If $d \geq 3$:
  - Yes, if t is fixed
  - No, if t is not fixed

$\Rightarrow$ in practice (we don't know d), it's undecidable.

9

Randomized Algorithms

- Maybe Randomized Algorithms do a better job than deterministic algorithms on LCL problems?
- Simple answer: No.
- Don't use randomized algorithms on LCL problems. You can always find a deterministic algorithm.

10

Part 2:

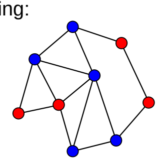
Weak Coloring

&

The Formal Dining Philosophers Problem

Weak Coloring

- Color the nodes of a graph, such that every node has at least one neighbor with a different color
- Weak 2-coloring:

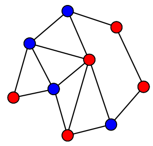


- Applications:
  - French Fries & Ketchup
  - Digital Camera & Printer

12

Proof: Every Graph has a Weak 2-Coloring

- Create an MST of the graph



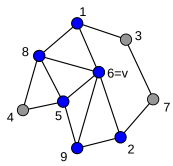
- Start at one node, walk through the MST in a breadth-first manner and color the nodes alternately

Weak Coloring as an LCL Problem

- A local algorithm for Weak 2-Coloring exists!
  - First (and only?) non-trivial locally solvable LCL problem
  - But: Algorithm only works if all nodes have odd degree
- Each node has to color itself with a local algorithm

Rank of a Node: $r_w(v)$

- Let v be a node. $N^+(v)$ is the set of all neighbors of v, including v itself.
- $r_v(v)$ is the rank of v in the set of its neighbors $N^+(v)$

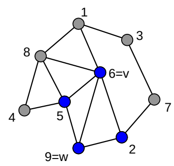


$N^+(v)$

- $N^+(v)=\{1,2,5,6,8,9\}$
- $r_v(v)=4$

Continued: Rank of a Node: $r_w(v)$

- $r_w(v)$ is the rank of v among the neighbors of node w ($=N^+(w)$)
  - Node v asks node w: "What is my rank in your perspective of view?"



$N^+(w)$

- $N^+(w)=\{2,5,6,9\}$
- $r_w(w)=4$
- $r_w(v)=3$

Local Algorithm for Weak 2-Coloring

- Works only if all nodes have odd degrees!
- Main idea: Calculate $r_w(v)$ for all neighbors $w \in N^+(v)$

- Phase 1: Generate a Weak Coloring with $d(d+1)^{d+2}$ colors
  - d is the maximum degree of a node in the graph
- Phase 2: Reduce the number of colors to 4
  - Algorithm needs time O(log*(d))
  - Works only if graph has bounded degree
- Phase 3: 4 colors to 2 colors
  - Algorithm needs time O(c), c = number of colors
  - Could also use this algorithm for phase 2

Algorithm that generates a Weak 2-Coloring (Phase 1)

- Every node v calculates its color vector C_v:
  - $C_v = (C_v[-1], C_v[0], C_v[1], \ldots, C_v[\deg(v)+1])$
    - The first component is in the range {1, ..., deg(v)}
    - The other components are in the range {1, ..., deg(v)+1}
  - Because of this, there are so many possible colors

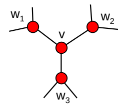
deg(v)	Largest possible color number
3	3072
5	1399680
8	$> 2^{32}$
11	$> 2^{48}$
14	$> 2^{64}$

Local Algorithm for Node v

- Preparation:
  - Create a list of all neighbors $w \in N^+(v)$
  - Sort it according to ID(w)
  - $C_v[-1] := deg(v)$
    - Nodes with different degrees are different and get different colors
  - $C_v[0] := r_v(v)$
    - Among the $r_w(v)$, $r_v(v)$ is special and needs to get stored at a fixed position
- For every neighbor w do: $C_v[r_v(w)] := r_w(v)$
- That's it! Algorithm is completely local: t=2
  - Every node asks only its neighbors

Proof: The Algorithm is correct (1)

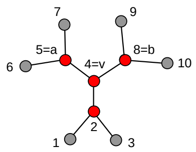
- Every node has a neighbor with a different color
- Proof by contradiction: Assume that v and all its neighbors have the same color.
  $C_v = C_w$ for all neighbors w of v



- We can conclude:
  - v and w have the same degree:
    $C_v[-1] = C_w[-1]$
  - v and w have the same rank among their neighbors:
    $C_v[0] = C_w[0] = r_v(v) = r_w(w)$

Proof (2): The contradiction is complete as soon as we can prove that...

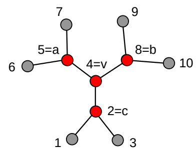
- ... v has two neighbors a and b that both have v at the same rank j among their neighborhood.
  - Formally: $r_a(v) = r_b(v) = j$



- Colors of a and b at array index j:
  - $\Rightarrow C_a[j] = C_a[r_a(v)] = r_v(a)$
  - $\Rightarrow C_b[j] = C_b[r_b(v)] = r_v(b)$
- $r_v(a)$ and $r_v(b)$ are not equal!
  - Thus the colors C_a and C_b are different!
  - This means that v has two neighbors with different colors
- To have the same colors:
  - $r_a(v) \neq r_b(v)$ (Rule 1)

Proof: The Algorithm is correct (3)

- Case 1: There are more neighbors with a higher ID than with a lower ID
  - Formally: $x := r_v(v) \leq \frac{deg(v)+1}{2}$
- For each such neighbor w:
  - $r_w(w) = r_v(v) \Rightarrow r_w(v) < r_v(v)$ (Rule 2)



- Fill in this table:

i:	c	v	a	b
$r_v(i)$:	1	**2**	3	4
$r_i(v)$:	4	**2**	1	1

1. $r_a(v) \neq r_b(v)$
2. $r_w(v) < r_v(v)$

Proof: The Algorithm is correct (4)

- Case 1, in general (case 2 is similar):

i:	Neighbors of v				v	Neighbors of v					
$r_v(i)$:	1	2	...	x-1	**x**	x+1	x+2	...	2x-1	2x	...
$r_i(v)$:					**x**	1	2	...	x-1	y	...

x-1 Nodes

$\geq$ x Nodes, x-1 smaller ranks than x

1. $r_a(v) \neq r_b(v)$
2. $r_w(v) < r_v(v)$

- $y \neq x \Rightarrow$ two neighbors of v have different colors
- $y = x \Rightarrow$ node has a different color

Algorithm that generates a Weak 2-Coloring (Phase 2)

- Phase 2: Reduce the number from c colors to 4 colors
- Algorithm for node v:
  - Choose the smallest c' with $\binom{c'}{\lfloor c'/2 \rfloor} \geq c$
  - Associate a different subset $S_i \subset \{1, \ldots, c'\}$ of size $\lfloor c'/2 \rfloor$ to every $i \in \{1, \ldots, c\}$
  - v has at least one neighbor w with a different color
  - v recolors itself to a color that is in $S_{color(v)}$, but not in $S_{color(w)}$.
  - Such a color exists, because the subsets have the same size and are not equal.
- But does node v know c?
  - c could be calculated if d is bounded

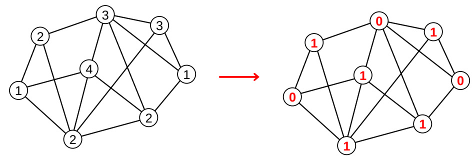
$\Rightarrow$ This is no local algorithm if d is unbounded!

Algorithm that generates a Weak 2-Coloring (Phase 3)

- Phase 3: 4 colors to 2 colors
  - Original coloring: c colors {1,2,...,c}
  - Recoloring in c rounds/steps
    - If c is not fixed, this is no constant-time algorithm (but it is local)
- Each node v waits until it has the smallest color number among its original-colored neighbors in $N^+(v)$.
- Then, v recolors itself according to the following rules:
  1. If v has only original-colored neighbors: Recolor to 0
  2. If v has recolored neighbors:
     If all the recolored neighbors have color 1: Recolor to 0
     There are recolored neighbors with color 0: Recolor to 1
- After the recoloring, node v announces its new color to its neighbors.

Phase 3: Correctness and Example

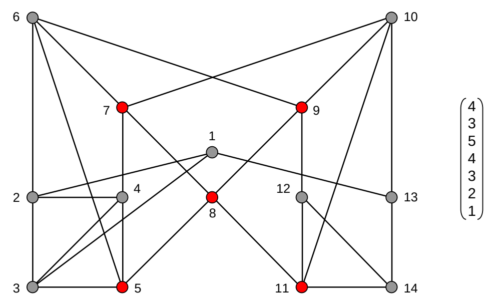
- Correctness at node v:
  - If v used rule 2, the node has a different-colored neighbor
  - If v used rule 1, it must have a neighbor w with a bigger original color than v.
    - w will recolor itself after v and use rule 2.
    - Because v has color 0, w will recolor itself to 1.



Nodes with even Degrees

- Now we see why this algorithm doesn't work with even degreed nodes
  - Every pigeon can find a hole if $r_v(v) = \frac{\deg(v)}{2} + 1$
- How difficult is it to find an example where the algorithm fails?
  - Node v is not properly colored if...
    - The degree d of v is even
    - Its rank in its neighborhood is $\frac{d}{2} + 1$
    - Every neighbor w of v has degree d and rank $r_w(w) = \frac{d}{2} + 1$ as well

Nodes with even Degrees: Example where the Algorithm fails



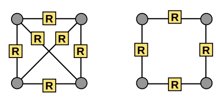
The Formal Dining Philosophers Problem: Introduction

- Variant of the Dining Philosophers Problem
- Formal dining:

A philosopher must wear two cuff links (*Manschetten-knöpfe*) while eating!

The Formal Dining Philosophers Problem: Definition

- Each node represents a processor and each edge a resource (or "cuff link").
  - A processor needs any two cuff links to eat
  - Two processors share one resource and are therefore in a conflict
  - Example:
    - Storage server farm
- Find a local algorithm
  - Safety?
  - Liveness?



Formal DPP *Dijkstra's DPP*

Finding an Algorithm for the Formal Dining Philosophers Problem

- Generate a Weak 2-Coloring
  - Colors: {0, 1, ∗}
  - We assume that the minimum degree of a node is 3.
  - All nodes where the algorithm fails recolor itself to color ∗.
- Assign two cuff links permanently to nodes colored ∗.
  - Are there enough cuff links left for the other nodes?
- Nodes colored {0,1} run a dynamic algorithm to get two cuff links
  - Length of the "waiting chain"?

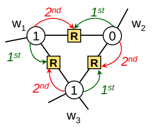
Permanent Assignment of Cuff Links to Nodes colored ∗

- The algorithm fails at node v only if...
  - v has even degree
  - half of its neighbors have lower and half have higher ranks
- A node colored ∗ grabs the two cuff links that lie on the edges to two nodes with lower IDs
- Are there enough cuff links left?
  - If w is a neighbor of v (v is colored ∗), then...
    - w has the same degree as v (at least 4)
    - The rank of w among its neighbors is half the degree plus 1

⇒ In the "worst case", only half of the adjacent edges are grabbed permanently

Nodes colored {0,1}

- Nodes colored {0,1} must run this algorithm to get a cuff link:
  1. Request cuff link from the first neighbor
  2. Request cuff link from the second neighbor
  3. Eat
  4. Release cuff links
  - "Request" means: Grab the cuff link, or wait until it's ready
- First and second neighbor need to be defined carefully to prevent deadlocks
  - Bad choice of 1st and 2nd neighbors:

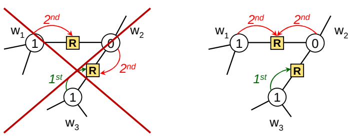


How to choose the Second and First Neighbor

- Trick: Choose the second neighbor first
  - Deadlock only occurs if a node can't grab its second resource
- If v is colored 1:
  - Choose any neighbor colored 0 as second neighbor
  - Announce this to all neighbors
- If v is colored 0:
  - Wait if v has been chosen as a second neighbor by neighbor w
    - If yes: Choose w as second neighbor to match the choice of w
    - If no: Choose any neighbor colored 1 as second neighbor
- Then choose an arbitrary first neighbor (other than the second neighbor)
  - Never choose a neighbor colored ∗ as first neighbor

Deadlock? – Proof about the Length of the Waiting Chain

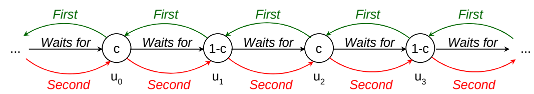
- Given any assignment of first and second neighbors, the maximum length of a waiting chain is at most 4
  - Can this happen?



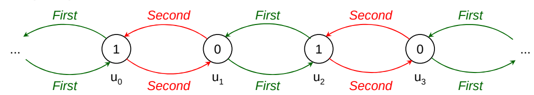
No, because w_2 would choose w_1 as its second neighbor to match the choice of w_1

Proof: Maximum Length of the Waiting Chain (1)

- Try to build a very long waiting chain:



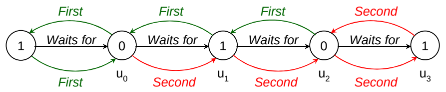
- The rules were violated. If we obey the rules, we get this (c=1):



⇒ It's impossible to build a waiting chain of arbitrary length!

Proof: Maximum Length of the Waiting Chain (2)

- The longest possible waiting chain has length 4



Summary of this Presentation

- Local algorithms & LCL problems
- It's undecidable if a local algorithm for a given LCL problem exists
- Randomized local algorithms: Don't use them
- Weak 2-Coloring:
  - Local algorithm that works if all nodes have odd degree
  - Color Generation & Color Reduction
  - Fails only in very rare cases
- Formal Dining Philosophers Problem:
  - Efficient algorithm based on Weak Coloring
  - Static "cuff link" allocation for nodes where Weak Coloring fails