Logical Characterizations in Distributed Computing

Jukka Suomela
Aalto University
Recently in Paris...

I was one of the examiners in Fabian Reiter’s PhD defense at Paris Diderot

Fabian’s talk started with *Fagin’s theorem* and then proceeded to introduce the "*Helsinki–Tampere theorem*"

What is this about?
Back to February 2010

I gave a talk in the Finite Model Theory Seminar on an unusual topic: models of distributed computing

Led to a collaboration that initiated the study of distributed graph algorithms from the perspective of descriptive complexity
Helsinki–Tampere team

Lauri Hella
Matti Järvisalo
Antti Kuusisto
Juhana Laurinharju

Tuomo Lempiäinen
Kerkko Luosto
J.S.
Jonni Virtema
What is distributed computing?
Centralized vs. distributed

• Theory of *centralized* computing:
  • “what can be computed efficiently with my laptop?”
  • input & output: encoded as a *string*
  • model of computing: *Turing machines*

• Theory of *distributed* computing:
  • ???
Distributed computing

• Can mean *lots of different things*
  • causes lots of confusion

• I’ll explain *two commonly used interpretations*
  • these are just “two extremes”
  • there is a whole spectrum of variants between them
Big data perspective

“*Too large* for my laptop to solve, I’ll have to resort to Amazon cloud”

Network algorithms

“How to coordinate data transmissions in a large network *without centralized control*?”
Big data perspective

Network algorithms
Big data perspective

• Focus: *computation*

• Distributed perspective helps us

Network algorithms

• Focus: *communication*

• Distributed perspective additional challenge
**Big data perspective**

- Fully centralized control
- *Global* perspective
- Input & output in one place

**Network algorithms**

- No centralized control
- *Local* perspective
- Input & output distributed
Big data perspective

• I know *everything* about input

• I need to know *everything* about solution

Network algorithms

• Each node knows its *own part* of input
  • e.g. local constraints

• Each node needs its *own part* of solution
  • e.g. when to switch on?
Big data perspective

• Explicit input
  • encoded as a string, stored on my laptop

• Well-known network structure
  • tightly connected cluster computer

Network algorithms

• Implicit input
  • input graph = network structure

• Unknown network structure
  • e.g. entire global Internet right now
**Big data perspective**

Can we divide problem in **small independent tasks** that can be solved **in parallel**?

**Network algorithms**

If each node is only aware of its **local neighborhood**, can we nevertheless find a **globally consistent solution**?
Big data perspective

Network algorithms
LOCAL model

• Initial knowledge:
  • local input, number of neighbors

• Communication round:
  • send message to each neighbor
  • receive message from each neighbor
  • update state
  • possibly: announce local output and stop
LOCAL model

• Each node labeled with a “unique identifier”
  • constant $k$ such that if we have a graph with $n$ nodes, unique identifiers are distinct values from $\{1, 2, \ldots, n^k\}$
LOCAL model

Equivalent:
- “running time”
- number of synchronous communication rounds
- how far do we need to look in the graph

Fast algorithm ↔ highly “localized” solution
LOCAL model

• The usual computer science perspective:
  • what is the worst-case running time?
  • asymptotically, as a function of $n$

• Two-player game:
  • player A chooses the algorithm
  • player B then chooses the graph, local inputs, unique identifiers
LOCAL model

• **Everything is computable in $O(n)$ rounds!**
  • assuming a connected graph
  • gather everything, solve locally by brute force
  • exploits: large messages, unlimited local computation

• Interesting question: what can be done in $o(n)$ rounds?
LOCAL model: examples

- Example: *graph coloring* with \( k \) colors
  - local input: nothing
  - local output: what is my own color
  - constraint: adjacent nodes have different colors
LOCAL model: examples

• Example: *graph coloring* with $k$ colors

• Graph family: *path* with $n$ nodes
  • $k = 2$: $\Theta(n)$ rounds
  • $k = 3$: $\Theta(\log^* n)$ rounds
  • $k = 100$: $\Theta(\log^* n)$ rounds
LOCAL model: examples

• Example: *graph coloring* with $k$ colors

• Graph family: 2D *grid* with $n \times n$ nodes
  • $k = 2$: $\Theta(n)$ rounds
  • $k = 3$: $\Theta(n)$ rounds
  • $k = 4$: $\Theta(\log^* n)$ rounds
  • $k = 100$: $\Theta(\log^* n)$ rounds
LOCAL model: examples

• Example: weak 2-coloring
  • label nodes with \{0, 1\}
  • each node has a neighbor with a different label

• Graph family: regular graphs
  • 4-regular graphs: $\Theta(\log^* n)$ rounds
  • 5-regular graphs: $\Theta(1)$ rounds
LOCAL model

• Why do we keep seeing “Θ(log* n)”?

• All of these are algorithms that exploit *numerical values of unique identifiers*
  • more precisely, it is Θ(log* s), where s = size of the identifier space
  • we just assumed that s = poly(n)
LOCAL model

• What if we don’t have unique identifiers?
Weak models of distributed computing
“Weak models”

- Initial knowledge:
  - local input, number of neighbors

- Communication round:
  - send message to each neighbor
  - receive message from each neighbor
  - update state
  - possibly: announce local output and stop

All of this identical to the LOCAL model!
“Weak models”

• Key difference: nodes are identical
  • no unique identifiers
  • “anonymous networks”
“Weak models”

• How to refer to your neighbors?

• *Port-numbering model:*
  
  • node of degree $d$ can refer to its neighbors with numbers $1, 2, \ldots, d$
  
  • “this is the message that I got from neighbor $x$”
  
  • “I want to send this message to neighbor $x$”
“Weak models”

• How to refer to your neighbors?

• **Set–broadcast model:**
  • no way to refer to specific neighbors
  • “this is the set of messages that I got from my neighbors in this round”
  • “I want to broadcast this message to all neighbors”
Weak models: computability

• Many problems cannot be solved at all

• Key challenges:
  • breaking symmetry
  • detecting cycles
Breaking symmetry

• Example: graph coloring

• Input graph:  

• Impossible to solve!
Breaking symmetry

• Input graph: \[ \text{○} \rightarrow \text{○} \]

• **Proof:**
  • same state before round \( t \)
  • same outgoing messages
  • same incoming messages
  • same state after round \( t \)
Detecting cycles

- Not possible to tell the difference between these graphs
Detecting cycles

• Not possible to tell the difference between these graphs
  • *Proof:* covering maps preserve everything
Weak models

• Lots of different models of distributed computing
  • “VV”, “MV”, “SV”, “VB”, “MB”, “SB” ...

• Key questions about each model:
  • which problems can be solved at all?
  • which problems can be solved in constant time?
Logical characterizations
Weak models & modal logic

• Natural 1:1 correspondence between:
  • constant-time distributed algorithms
    set–broadcast model
  • formulas in basic modal logic

• Both equally expressive: can “solve”
  the same set of graph problems
Modal logic & computing

• Textbook approach:
  • possible world ≈ possible state of the system
  • accessibility relation ≈ state transition

• Our perspective:
  • possible world ≈ computer
  • accessibility relation ≈ communication link
<table>
<thead>
<tr>
<th>Modal logic</th>
<th>Distributed algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kripke model $K = (W, (R_\alpha)_{\alpha \in I}, \tau)$</td>
<td></td>
</tr>
<tr>
<td>states $W$</td>
<td>input graph $G = (V, E)$</td>
</tr>
<tr>
<td>relations $R_\alpha$, $\alpha \in I$</td>
<td>port numbering $p$</td>
</tr>
<tr>
<td>valuation $\tau$</td>
<td>nodes $V$</td>
</tr>
<tr>
<td>proposition symbols $q_1, q_2, \ldots$</td>
<td>edges $E$ and port numbering $p$</td>
</tr>
<tr>
<td>formula $\varphi$</td>
<td>node degrees (initial state)</td>
</tr>
<tr>
<td>formula $\varphi$ is true in state $v$</td>
<td></td>
</tr>
<tr>
<td>modal depth of $\varphi$</td>
<td>algorithm $\mathcal{A}$</td>
</tr>
<tr>
<td></td>
<td>algorithm $\mathcal{A}$ outputs 1 in node $v$</td>
</tr>
<tr>
<td></td>
<td>running time of $\mathcal{A}$</td>
</tr>
</tbody>
</table>
Technology transfer

Using tools from logic to prove results on distributed computing

e.g. bisimulation

Figure 5: Classes of graph problems. (a) Trivial subset relations between the classes.
(b) The linear order identified in this work.

Remark 2.

In each problem class, we consider algorithms in which each node knows its own degree. While this is natural in all other cases, it may seem odd in the case of class $SB$. In principle, we could define yet another class of problems $SB_o$, defined in terms of degree-oblivious algorithms in $Set \setminus Broadcast$, i.e., algorithms with a constant initialisation function $z_0$. However, it is easy to see that $SB_o$ is entirely trivial—in essence, one can only solve the problem of distinguishing non-isolated nodes from isolated nodes—while there are many non-trivial problems that we can solve in class $SB$. Hence we will not consider class $SB_o$ in this work. However, class $SB_o$ is more interesting if one considers labelled graphs; see Section 3.4.

Contributions

This work is a systematic study of the complexity classes $VV_c$, $VV$, $MV$, $SV$, $VB$, $MB$, and $SB$, as well as their constant-time counterparts. Our main contributions are two-fold. First, we present a complete characterisation of the containment relations between these classes. The definitions of the classes imply the partial order depicted in Figure 5a. For example, classes $VB$ and $SV$ are seemingly orthogonal, and it would be natural to assume that neither $VB \not\subset SV$ nor $SV \not\subset VB$ holds. However, we show that this is not the case. Unexpectedly, the classes form a linear order (see Figure 5b):

$$SB \subset MB \subset VB \subset SV \subset MV \subset VV \subset VV_c.$$ (1)

In summary, instead of seven classes that are possibly distinct, we have precisely four distinct classes. These four distinct classes of problems can be concisely characterised as follows, from the strongest to the weakest:

1. consistent port numbering (class $VV_c$),
2. no incoming port numbers (class $SV$ and equivalent),
3. no outgoing port numbers (class $VB$ and equivalent),
4. neither (class $SB$).

6
What has happened since 2010?
Beyond constant time

• Easy: running time \approx \text{operator depth}

• Much more challenging to capture: non-constant running time
Beyond constant time

- Promising approach: *fixed-point logic*
  - e.g. *modal μ-calculus*
  - Antti Kuusisto (CSL 2013)
  - Fabian Reiter (ICALP 2017)
Nondeterminism & alternation

• **Stronger** models of distributed computing
  • cf. nondeterministic & alternating Turing machines
  • cf. class NP & polynomial hierarchy

• Logical characterizations:
  • “alternating local distributed automata”
    ≈ monadic second-order logic
  • Fabian Reiter (LICS 2015)
Nondeterminism & alternation

• Active research topic: *distributed decision*
  • yes-instance: *all nodes say “yes”*
  • no-instance: *at least one node says “no”*

• E.g.: $O(\log n)$ *bits per node* per quantifier
  • Göös, S. (PODC 2011)
  • Feuilloley, Fraigniaud, Hirvonen (ICALP 2016)
What is happening right now?
Structural complexity theory

- **Centralized computing:**
  - time hierarchy theorem
  - more time → can solve more problems

- **Distributed computing:**
  - gap results
  - \( o(\log n) \) rounds ≈ as good as \( O(\log^* n) \) rounds
Structural complexity theory

- Key idea that has enabled lots of progress: *identify the right family of problems*
  - do not try to prove something about “all graph problems”
  - focus on “LCL problems” (locally checkable labeling)
  - distributed analogue of class NP: solutions are easy to verify, but may be hard to find
Structural complexity theory

• *Lots of progress:*
  • Brandt et al. (STOC 2016)
  • Chang et al. (FOCS 2016)
  • Ghaffari & Su (SODA 2017)
  • Brandt et al. (PODC 2017)
  • Chang & Pettie (FOCS 2017)
  • Balliu et al. (STOC 2018) …
Structural complexity theory

• One of the current obstacles: we seem to be still lacking the right definitions
  • example: LCLs work well for graphs of maximum degree $O(1)$, but how to generalize beyond that?

• Could we try to replace the current algorithmic or graph-theoretic definitions with logical characterizations?
Thanks!
Happy Birthday!