Understanding Computation with Computation

Jukka Suomela
Aalto University, Finland

Joint work with...

Keijo Heljanko, Janne H Korhonen, Tuomo Lempiäinen
Patric RJ Östergård, Christopher Purcell, Siert Wieringa (Aalto)

Sebastian Brandt, Przemysław Uznański (ETH)

Matti Järvisalo, Joel Rybicki (Helsinki)

Juho Hirvonen (Paris Diderot)

Christoph Lenzen (MPI)

Stefan Schmid (Aalborg)

Danny Dolev (Jerusalem)

… and many others

Algorithm synthesis

- Computer science: what can be automated?
- Can we ***automate our own work***?
- Can we outsource algorithm design to computers?
  - **input:** problem specification
  - **output:** asymptotically optimal algorithm

Verification and synthesis

- **Verification**:
  - given problem *P* and algorithm *A*
  - does *A* solve *P* ?
- **Synthesis**:
  - given problem *P*
  - find an algorithm *A* that solves *P* ?

Verification and synthesis

- Algorithm **verification** often difficult
  - easy to run into e.g. halting problem
- Algorithm **synthesis** is entirely hopeless?
- Not necessarily!
  - verifying ***arbitrary*** algorithms in model M
  - synthesising only "***nice***" algorithms in model M

Setting

- Our focus: **distributed algorithms**
  - multiple nodes working in parallel
  - complicated interactions between nodes
  - possibly also faulty nodes, adversarial behaviour
- Computational techniques in algorithm design can outperform human beings

Setting

- We do **theory**, not practice
- Desired outputs:
  - ***algorithm design & analysis***
  - ***lower-bound proofs***
- We want **provably correct algorithms**, not something that “seems to work”

Success stories (1/4)

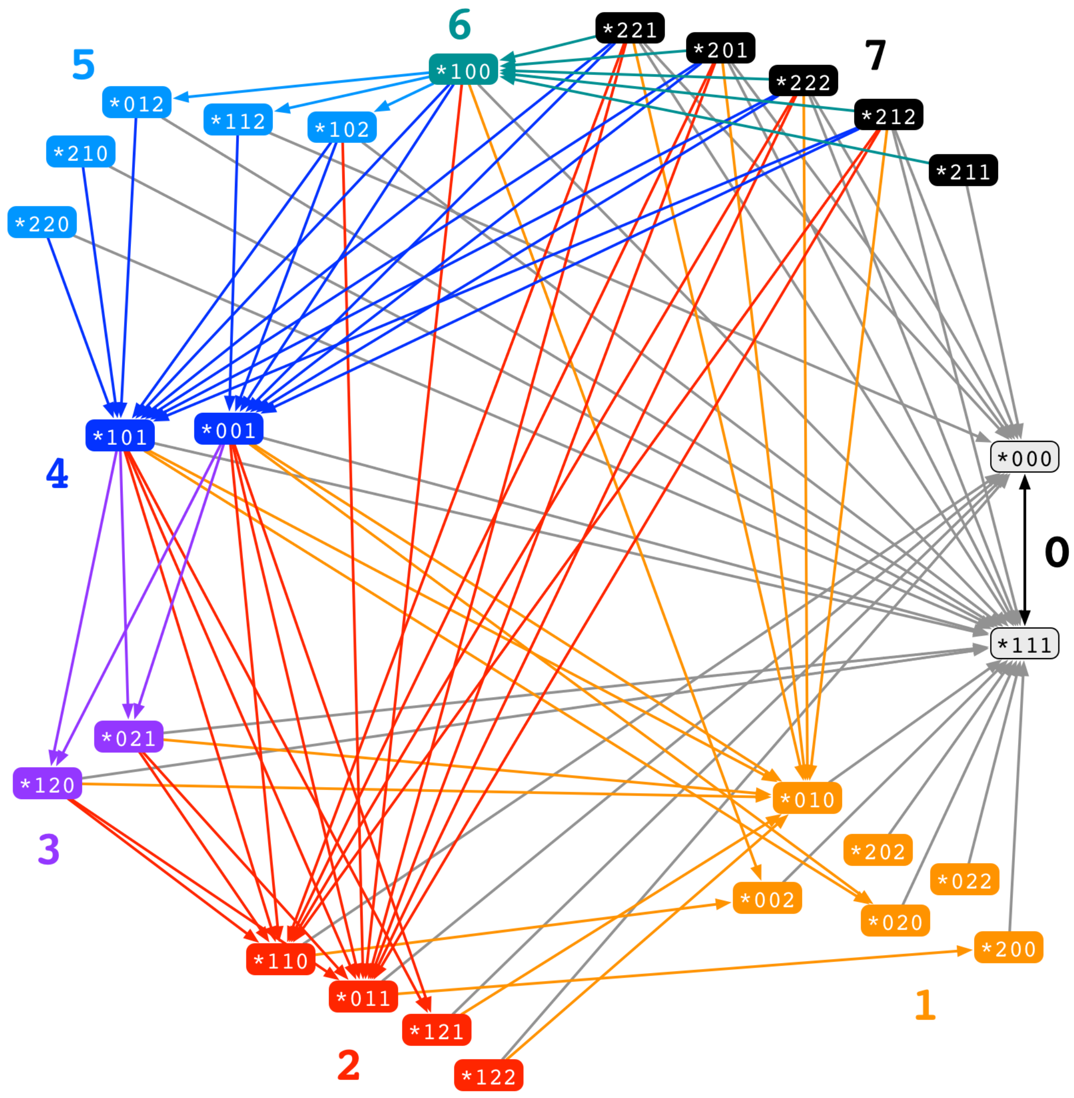
- **Fault-tolerant digital clock synchronisation**
  - nodes have to count clock pulses modulo c
  - ***self-stabilising algorithms***: reaches correct behaviour even if the starting state is arbitrary
  - ***Byzantine fault tolerance***: some nodes may be adversarial

4 nodes

1 faulty node

3 states per node

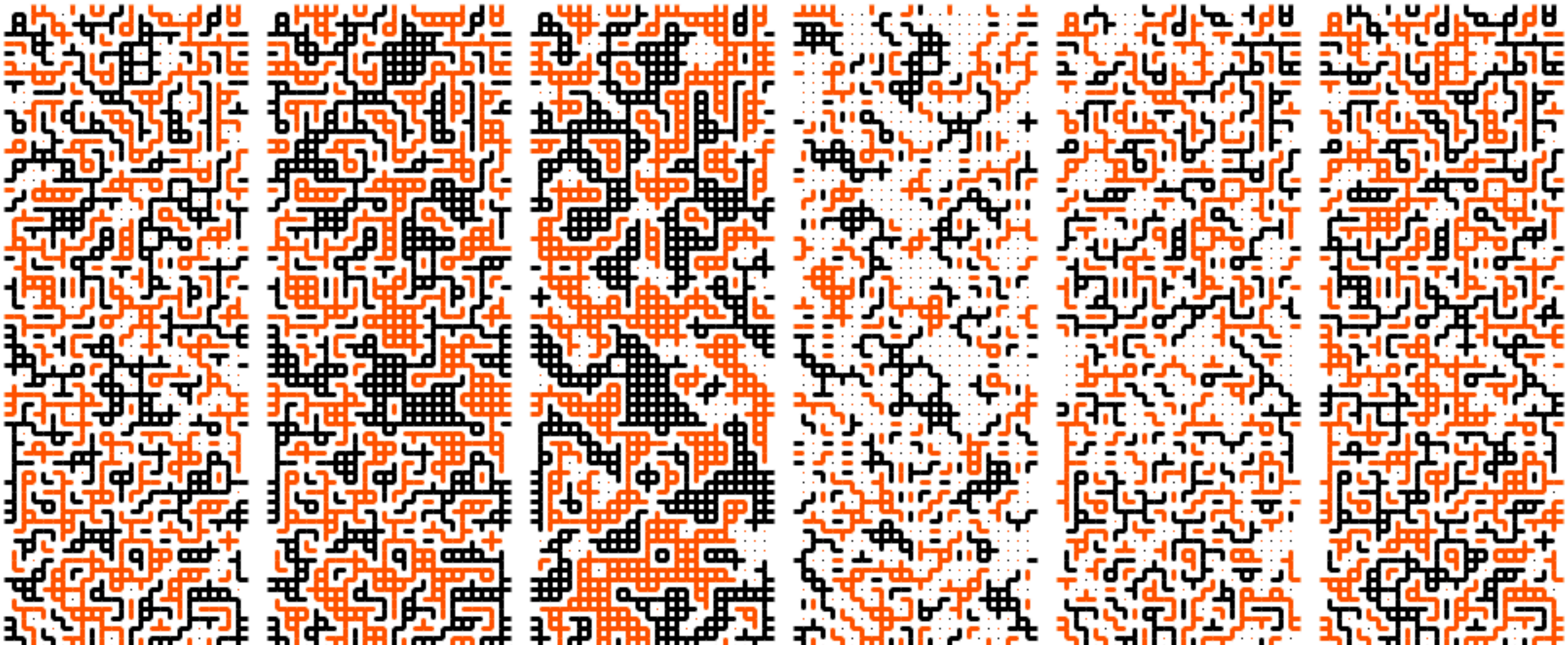
always stabilises
in at most 7 steps



Success stories (2/4)

- **Theorem:** any triangle-free *d*-regular graph has a cut of size $\left(\frac{1}{2} + \frac{\mathbf{0.281}}{\sqrt{d}}\right) m$
  - prior bound: $\left(\frac{1}{2} + \frac{\mathbf{0.177}}{\sqrt{d}}\right) m$ (Shearer 1992)
- **Proof:** we design a ***simple randomised distributed algorithm*** that finds such cuts (in expectation)

Pick a random cut, change sides if at least $\left\lceil \frac{d+\sqrt{d}}{2} \right\rceil$ neighbours on the same side

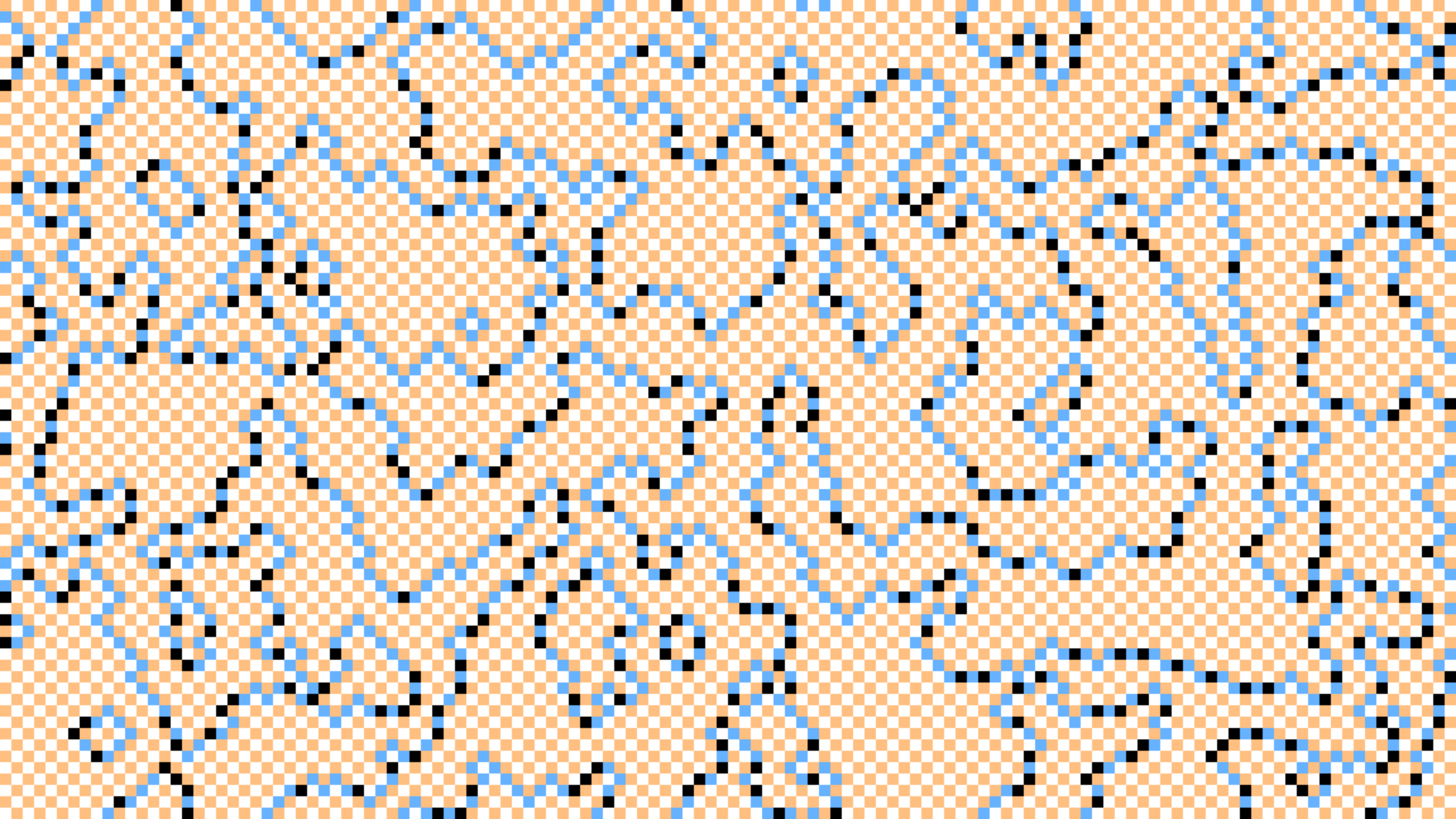


Success stories (3/4)

- Classical symmetry-breaking primitive:
  - input: directed path coloured with **n colours**
  - output: directed path coloured with **3 colours**
- Prior work: **½ $\log^*(n) \pm O(1)$** rounds
- New result: exactly **½ $\log^*(n)$** rounds for infinitely many n

Success stories (4/4)

- Any **locally checkable labelling problem**
  - maximal independent set, colouring …
- Setting: cycles, 2-dimensional grids, …
- Complexity is $O(1)$, $\Theta(\log^* n)$, or $\Theta(n)$
- **Synthesis possible for class $\Theta(\log^* n)$**



Key challenges

- A combinatorial search problem
  - find an object *A* that satisfies these constraints…
- How to make the problem **finite**?
  - so that the problem is ***solvable at least in principle***
- How to solve it in **practice**?
  - how to avoid ***combinatorial explosion***

Key challenges

- Much easier to make the problem finite if we **fix some parameters**:
  - algorithm for $n = 10$ nodes?
  - algorithm for any n, but maximum degree $\Delta = 10$?
- How to **generalise**?

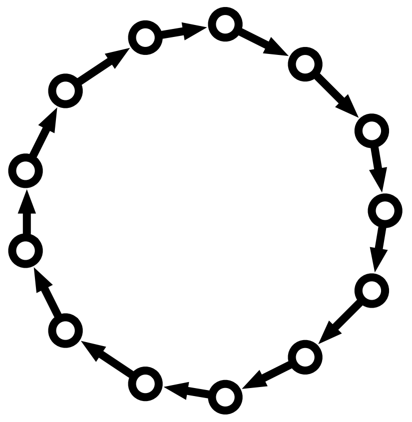
How to generalise

1. **Computer-inspired algorithms**
   - computer solves ***small cases***, generalise the idea
2. **Generalise by induction**
   - computer solves the ***base case***, prove inductive step
3. **Direct synthesis for the general case**
   - sit down and relax

How to generalise

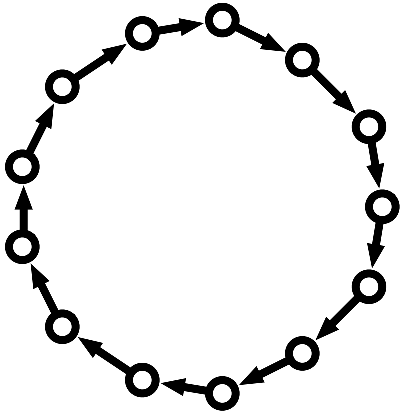
1. **Computer-inspired algorithms**
    - example: ***large cuts***
2. **Generalise by induction**
    - example: ***clock synchronisation***
3. **Direct synthesis for the general case**
    - example: $O(\log^* n)$***-time algorithms***

LCLs on cycles



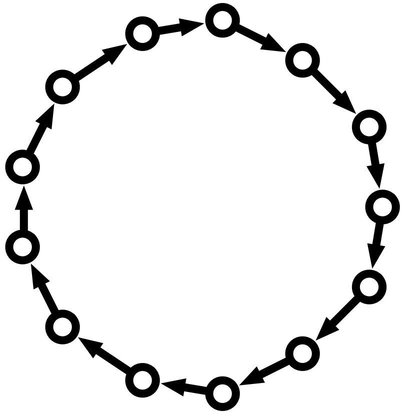
- Computer network = directed *n*-cycle
  - nodes labelled with ***O*(log *n*)-bit identifiers**
  - each round: each node exchanges (arbitrarily large) **messages** with its neighbours and updates its state
  - each node has to output its **own part of the solution**
  - ***time = number of rounds*** until all nodes stop
  - equivalently: ***time = distance*** (how far to look)

LCLs on cycles



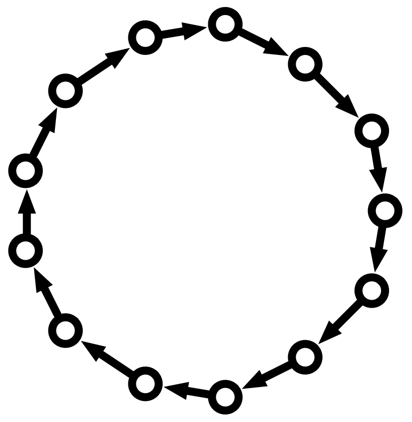
- LCL problems:
  - solution is globally good if it **looks good in all local neighbourhoods**
  - examples: vertex colouring, edge colouring, maximal independent set, maximal matching…
  - cf. class NP: solution ***easy to verify***, not necessarily easy to find

LCLs on cycles



- 2-colouring: inherently global
  - $\Theta(n)$ rounds
- 3-colouring: local
  - $\Theta(\log^* n)$ rounds

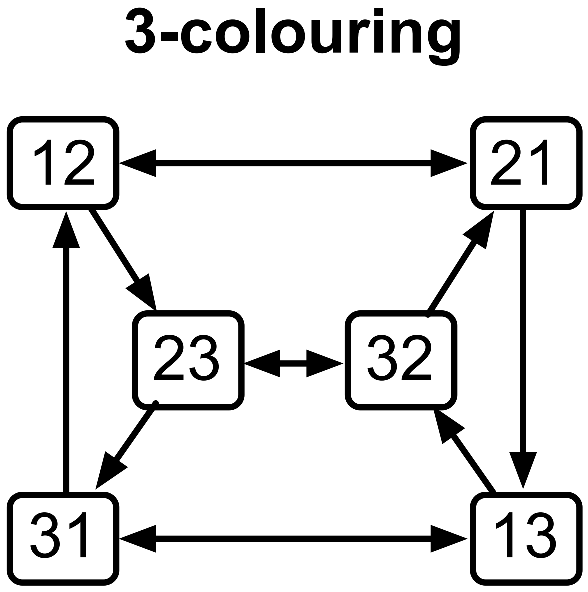
LCLs on cycles



- Given an algorithm, it may be very difficult to **verify**
  - easy to encode e.g. halting problem
  - running time can be any function of *n*
- However, given an LCL problem, it is very easy to **synthesise** optimal algorithms!

LCLs on cycles

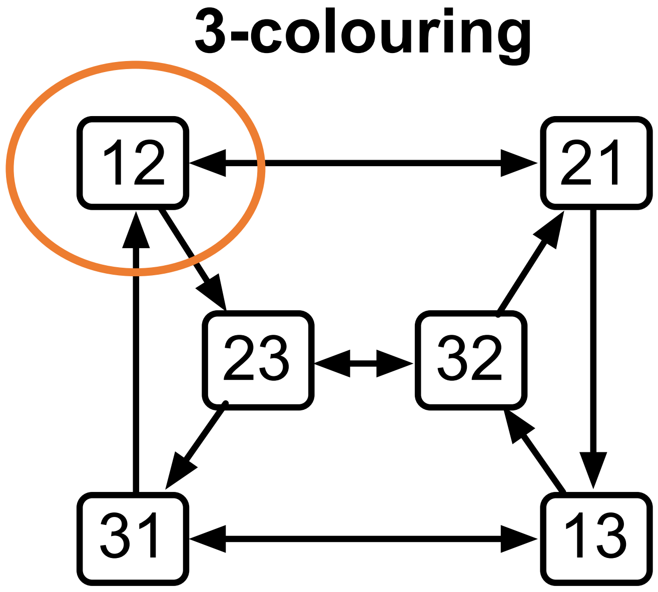
- LCL problem ≈ set of feasible local neighbourhoods in the solution
- Can be encoded as a graph:
  - node = neighbourhood
  - edge = “compatible” neighbourhoods
  - walk ≈ sliding window



LCLs on cycles

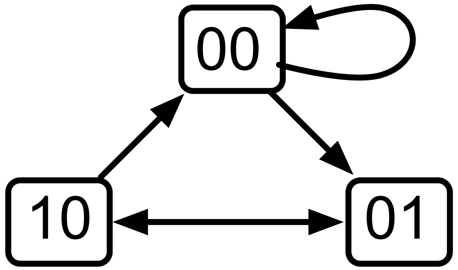
Neighbourhood *v* is "***flexible***" if for all sufficiently large *k* there is **a walk *v* → *v* of length *k***

- equivalent: there are walks of coprime lengths
- "**12**" is flexible here, $k \geq 2$

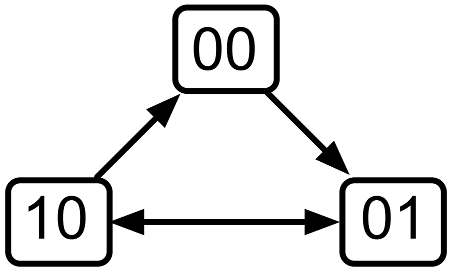


LCLs on cycles

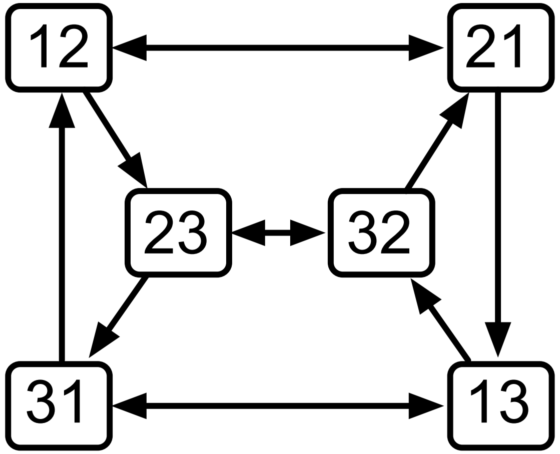
independent set



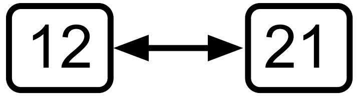
maximal independent set



3-colouring



2-colouring



self-loops:
$O(1)$

flexible states:
$\Theta(\log^* n)$

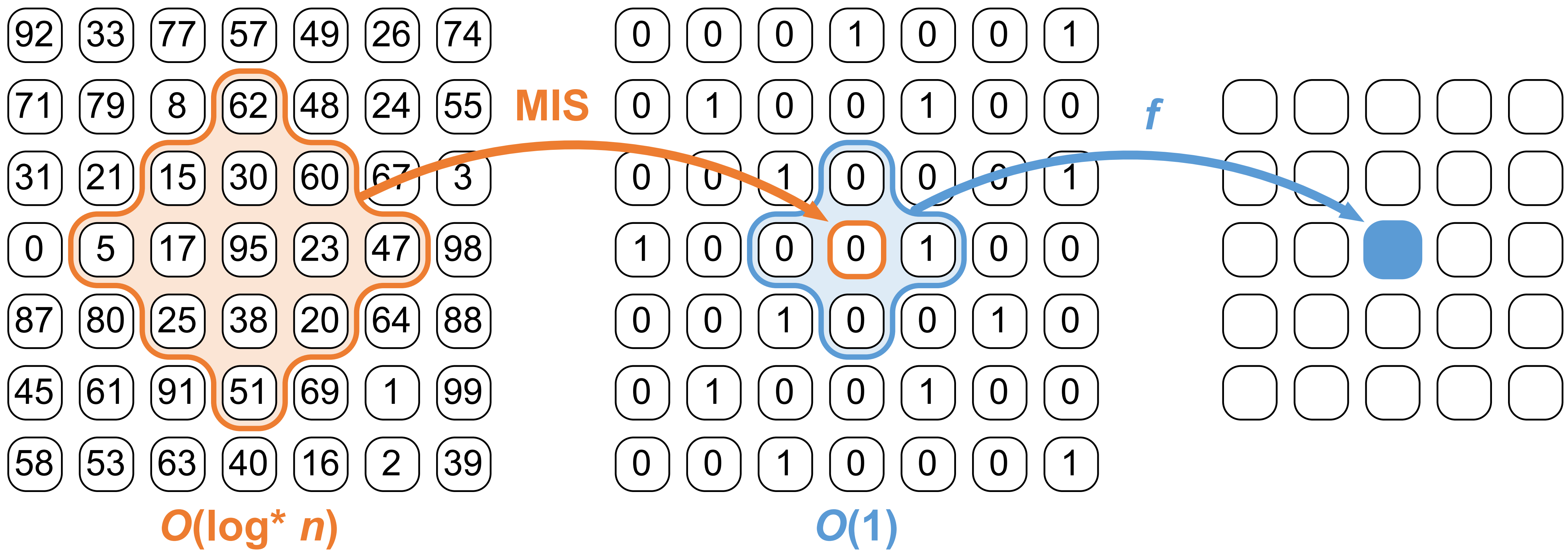
otherwise:
$\Theta(n)$

LCLs on cycles

- Verification hard but synthesis easy:
  - construct graph, analyse its structure
- "**Compactification**":
  - any LCL problem can be represented ***concisely*** as a graph
  - seemingly open-ended problem of finding an efficient algorithm is reduced to a simple graph problem

Beyond cycles

- Classification **undecidable** on 2D grids
  - “is this problem solvable in $O(\log^* n)$”
- But **1 bit of advice** is enough!
  - just tell me whether it is solvable in time $O(\log^* n)$
  - then I can find an optimal algorithm — at least in principle, but often also in practice
  - key insight: “***normal form***” for any such algorithm


MIS
f
O(log* n)
O(1)

Future

- How far can we push these techniques?
  - immediate next steps: distributed algorithms in much more general graph families
- More focus on ***meta-algorithmics***?
  - how to design algorithms for designing algorithms
- Algorithms for ***lower bounds***?