Local checkability: a self stabilization notion that impacts foundations of distributed computing

Shay Kutten
Technion
This is NOT a survey

Showing some broad impacts of a notion developed initially for self stabilization

Apologies in advance for important papers not mentioned.
Once upon a time there was a non-distributed computing. It was very poor in computational power. 😞

It was also lonely. 😞

But it was had a beautiful structural theory, involving verification e.g. NP: a graph 3 coloring witnesses 3-colorability. Another example: PCP
Are decision problems as important in distributed computing?

[Fraigniaud PODC’2010] invited talk about the “Impossibility because of asynchrony” party vs. the “Network computing” party:

Red decision problems ⇒ red structured hierarchies

BUT, for the blues, it seemed as if:

Multiple algorithms existed, unordered. Where were the elegant hierarchies?
Before local checking, or even distributed verification

Everybody here knows this example, but I want to highlight the locality
(American) Indians in a council

Invention of the IBM token ring network
Only holder of talking stick (token) is allowed to talk.
On finishing talking, pass the stick

Looks local, BUT,
Is a recovery local?

Locally, every node in Legal state

Chief

Your text here

Token
Industry recovery: **global detection**

Enough time for circling whole ring ➔ token must be lost

Chief

Token
Dijkstra alg. (1)

token
(at chief
8 \rightarrow 8)

Always a token-
all states equal \Rightarrow chief has token
Always a token-
states differ => regular guy has token
Faults: Dijkstra kept *locality* (token holders act) but no *detection*

The local states still look legal $\Rightarrow$ no detection

"self stabilization in spite of *Distributed* 
(local) control"
Bringing back detection, but distributed, local

- Dijkstra’s algorithm “magically” stabilized. No node actually detected fault
- [KP90]: general self stab. global checking
  - Collect all inf. to leader
  - leader detects faults
  - leader restarts computation
- [AKY90] self stabilizing leader election

Check & detect $\Rightarrow$ generality. However, global
“local detection” (also [AKY90,97])
(Renamed “local checking” [AV91,APV91]
following Blum’s “program checking”)

Equality is a **locally checkable** function:
If not all X values are equal, …
“local detection” (also [AKY90,97])
Renamed “local checking” [AV91,APV91]
following Blum’s “program checking”)

Equality is a locally checkable function:
If not all X values values are equal, at least one node can detect (inequality) in constant time (independent of diameter)
“local detection” (also [AKY90,97])
Renamed “local checking” [AV91,APV91]
following Blum’s “program checking”

Essence of local checkability
(and the [AKY90] criterion for rejection)

Equality is a **locally checkable** function:
If not all $X$ values are equal, **at least one node**
can detect (inequality) in **constant** time
(independent of diameter)
“local detection” (also [AKY90,97])
Renamed “local checking” [AV91,APV91]
following Blum’s “program checking”)
"local detection" (also [AKY90,97])
Renamed "local checking" [AV91,APV91] following Blum’s "program checking")

A cycle will be detected by a node seeing only its parent’s state and its own state.

State: Represents a variable PARENT in a node, pointing at a neighbor
“local detection” (also [AKY90,97])
Renamed “local checking” [AV91,APV91] following Blum’s “program checking”)

A cycle will be detected by a node seeing only its parent’s state and its own state.
“local detection” (also [AKY90,97])
Renamed “local checking” [AV91,APV91]
following Blum’s “program checking”)

A cycle will be detected by a node seeing only its parent’s state and its own state.

(This verification resembles the computation of BFS and routing[Belmman58,Ford62]: node distance ← 1+ min distance of a neighbor;

In self stab. [Dolev,Israel,Moran89],[Dolev,Israel,Moran90],[Arora,Gouda90] )
Local detection

A cycle will be detected by a node seeing Only its parent’s state and its own state.
More formally [KKP05]

Proof labeling scheme for graph family $\Gamma$ and predicate $f$ (e.g. $f_{\text{MST}}$)

- **Marker $M$**: Input: graph $G$ with nodes' states
  Output: $\text{Label } M(v) \ \forall v \in G_{\text{states}}$

- **Verifier $V_u$**: Output: Yes or No
  Input: 

Labels are witnesses!
Local checking with labels

∀ \text{G}_{\text{states}} \in \Gamma:

f(\text{G}_{\text{states}}) = 1 \Rightarrow \forall \nu, \text{Verifier(Marker}(\nu)) = \text{yes}

f(\text{G}_{\text{states}}) = 0 \Rightarrow \forall \text{Labels}, \exists u \in \text{G}_{\text{states}} \text{ s.t. } \text{Verifier}_u(\text{Labels}(u)) = \text{no}

Example: f=Equality, label=null, Verifier just compares
Local checking with labels

**Essence of local checkability:**

\[ f(G_{\text{states}}) = 1 \Rightarrow \forall v, \; \text{Verifier}(\text{Marker}(v)) = \text{yes} \]

\[ f(G_{\text{states}}) = 0 \Rightarrow \forall \text{Labels}, \; \exists u \in G_{\text{states}} \; \text{s.t.} \]

\[ \text{Verifier}_u(\text{Labels}(u)) = \text{no} \]

"Rejection:" local verifier in (at least) one node says "no"

(AKY90 rejection criterion)
The cycle freedom example, formally

(Graph family:  \( \Gamma = \{ G_s \} \) ;

- **States** induce subgraph \( H \)
  \( e \in H \iff \) endpoint points at \( e \)

- **Labels**: distances from root

- \( f(G_s) = 1 \iff \) the subgraph is a cycle free.

(Here \( f(G_s) = 0 \) because of the cycle A-B-C-D-A)
Various papers use different names
• (global) predicate
• (global) function
• (distributed) language
• marking (function)
• Labeling (function)
• (I will not use ”labeling”
To avoid confusion with PLS)

E.g. here the language is of all the markings (states)
Of edges (ports in nodes) not inducing cycle
My root is W

My root is v
My dist. is 2
parent is X

Labels for checking cycle freedom

labels for checking (root-id) equality
Applications in self stabilization

Constructing a general transformer: non self stabilizing algorithm $\rightarrow$ self stabilizing alg.

[AKY90, AV91, APV91, AKMPV93, APVD94]

Using (1) checking & detection (2) reset

{(3) Termination detection}
(Given) algorithm computed partial results, and sent messages (in transit)
Assume fault detected by some nodes

Partial results in local memory

LET THERE BE RESET

X = 5, y = 3

message
The effect of reset

LET THE (given) ALGORITHM START (from initial state)

X=Y=
The transformers

[AKY90, AV91, APV91, AKPV93, APVD94]

Using (1) checking & detection
   (2) reset [AKY90, AG90, AV91, APV91, APVD94]
   {(3) Termination detection}

(A) Execute non-self stabilizing Alg. (e.g. GHS for MST)
(B) “Wait until Alg. Terminates”
(C) Do forever: Local check
(D) If error detected, reset and go to A
Termination detection in [AV91]: synchronous network, count time (time complexity of Alg.)

(A) Execute non-self stabilizing Alg. (e.g. GHS for MST)
(B) Wait until Alg. Terminates
(C) Local check
(D) If error detected, reset and goto A
The transformers (2)

[AKY90, AV91, APV91, APVD94, AD97]

Termination detection in [AV91]:
- use synchronous network,
- count time (time complexity of Alg.)

New (self stab) termination detection, asynchronous
[Anais Durand, K, SIROCCO 2018 (BA)]

(Less messages and weaker assumptions then using self stabilizing synchronizer)
The transformers (3)

[AKY90, AV91, APV91, APVD94, AD97]

[APVD94] assumed, instead of termination:
- If global-state GS1 leads to global-state GS2
- then Verifier(GS1) \(\Rightarrow\) Verifier(GS2)

[Beauquier, Bernard, Burman, K., and Laveau]
BA in SSS’18
Self-stabilizing Stable Marriage
(improved complexity using the general transformer)
Original reset [Finn] (not self-stab) had a bug: reset message could **cycle**

(so [AKY90] solved also this problem by constructing a tree and electing a leader who was the only one to start a reset)
The original reset [Finn] (not self-stab) had a bug: the reset message could cycle.
The original reset [Finn] (not self-stab) had a bug:
the reset message could cycle
The original reset [Finn] (not self-stab) had a bug: the reset message could cycle.
The original reset [Finn] (not self-stab) had a bug: the reset message could **cycle**.
The original reset [Finn] (not self-stab) had a bug:
the reset message could **cycle**

*Diagram showing a network with nodes labeled C, D, E, F, G, H, I, B, and A, with arrows indicating the direction of the reset message.*
Reset is simpler on a tree, using self stabilizing Bcast&Echo

(PIF. e.g [Kruijer79, Varghese94, ABDT98]

Let there be reset

Freeze! message
Silent stabilization

Memory requirements for silent stabilization [Dolev, Gouda, Schneider (1999)]

Analyzed #bits needed for silent (stabilized) state, (values stored in the communication registers are fixed)

On proof labeling scheme vs. silent self-stabilizing algorithms [Blin, Fraignianud, Patt-Shamir (2014)]

Basically equivalent.
Local checking of other functions and Complexity issues

• Any graph marking function is (can be made) locally checkable (by adding labels)

• Any algorithm global state is locally checkable

• For any bit complexity $C$, $\exists$ function with local checking bit complexity (label size) $\Theta(C)$

• Some complexities for interesting marking functions are known

• Many other problems are open
Assume $M(v_i) < \frac{1}{2} \log n$ for every $i$.

$$\Rightarrow \exists \text{ Two pairs } M(v_i), M(v_{i+1}) = M(v_k), M(v_{k+1})$$
Assume $M(v_i) < \frac{1}{2} \log n$ for every $i$

$f(path) = 1$  

Assume, for simplicity that each $v_j$ sees only $v_{j+1}$

$\Rightarrow \exists$ Two pairs $M(v_i), M(v_{i+1}) = M(v_k), M(v_{k+1})$
Assume $M(v_i) < \frac{1}{2} \log n$ for every $i$

$\implies \exists$ Two pairs $M(v_i), M(v_{i+1}) = M(v_k), M(v_{k+1})$
Lower Bound (proof label for tree)

Assume $M(v_i) < \frac{1}{2} \log n$ for every $i$

\[ f(path) = 1 \]

Assume, for simplicity that each $v_j$ sees only $v_{j+1}$

\[ \Rightarrow \exists \text{ Two pairs } M(v_i), M(v_{i+1}) = M(v_k), M(v_{k+1}) \]
Local checkability naturally started in the context of self stabilization.
Impacts outside self stabilization
(1) structural complexity

Once upon a time there was a
LCL: Locally Checkable \{Languages\} \[NS93\] vs.

**PLS: Proof Labeling Schemes**

LCL: functions checkable with label=null
e.g. equality, MIS, node coloring, edge coloring
(but using the \[AKY90\] criterion for rejection)

Any function $F \rightarrow \text{Proof Labeling Scheme} \rightarrow \text{LCL}(F) \rightarrow F$
e.g., “cycle free” not LCL, but “cycle free with distance labeling” is (here, cost: $O(\log n)$ bits)
Localy in **checking AND computing**

[NS93]: What **LCLs** can be **also** **computed** locally?  
(LCL: Locally Checkable Languages using [AKY90] criterion for rejection, but with no label)

- Surprisingly, there are some non-trivial “local” LCLs
- Characterizing “local” LCLs?  
  - Undecidable whether an LCL is “local”
- Randomization cannot make LCL “local”
- Any “local” LCL has ID order invariant local algorithm
Some structural complexity studies for Classes defined by local checkability

- ...exponential separation...(LCL)
- New classes (LCL) problems...
[Cheng,Kopelowitz,Pettie16], [Balliu, Korhonen,Lempiäinen,Olivetti,Suomella18], ...

- Many problems $\Theta(1)$, $\Theta(\log^* n)$, $\Theta(\log n)$, $\Theta(n^{1/k})$ time
- also $\Theta(\log^\alpha n)$, $\Theta(\log^\alpha \log^* n)$, for $\alpha > 1$
- $2^{\Theta(\log^\alpha n)}$, $2^{\Theta(\log^\alpha \log^* n)}$ , $\Theta((\log^* n)^\alpha$ for $\alpha < 1$
- $\Theta(n^{\alpha})$, for $\alpha < 1$
- Gaps $\omega(1)$ to $o(\log \log^* n)$ , $\omega(\log^* n)$ too($\log \log^* n$)

- Specific graph families (e.g. trees), different structures
Locally Checkable vs. NP (Non-Deterministic Polynomial)

- **NP**: guessing solution \(\rightarrow\) easy checking
  - e.g. 3 coloring (possibly long time to compute)

- **LCL** [NS93]: guessing function \(\rightarrow\) easy checking
  - e.g. 2 coloring (\(\theta(n)\) in cycles)

- **PLS** [AKY90, KKP05]:
  - guessing function + label \(\rightarrow\) easy checking
  - e.g. spanning tree (\(\theta(n)\) in large diameter graphs)
Locally Checkable vs. NP (Non-Deterministic Polynomial)

- **NP**: reminder: “easy” means polynomial #steps

Distributed computing:

“easy” means local
  (in LOCAL model, node outputs based on inf. from near by nodes)

“hard”: need information from far (e.g. $\Omega(n^\alpha)$ distance
Structural complexity studies for Classes defined by local checkability: classes by label size

Locally Checkable Proofs [Göös, Suomela(11)]

Rephrasing PLS in terms of problem classes (label size in Proof Labeling Schemes)

Mostly look at the class of problems that can be made locally checkable with a label of $O(\log n)$ bits
Multi interactions PLS

[Feuilloley, Fraigniaud, Hirvonen16]
“A Hierarchy of Local Decision”

• Interaction between prover and disprover
  • Reduces e.g. MST label from $\Omega(\log^2 n)$ to $O(\log n)$
  • “Polynomial local hierarchy”
    • $(G; x) \in L \iff$
    • $\exists \ell_1 \forall \ell_2 \exists \ell_3 \ldots Q\ell_k: \text{Acc}(G, X, \ell_1, \ell_2, \ldots Q\ell_k)$
    • (AKY90 Criterion)

• Original labeling scheme (AKY, KKP)
  Just: $\exists \ell_1$

• Original local checkability (AKY, APV, NS, APVD): Just $F(G, X)$
Towards a Complexity Theory for Local Distributed Computing

[Fraigniaud,Korman,Peleg(JACM)13]

Some of the results related to local checkability:

- **Classes of problem DL(t)** - PLS or LCL: t = constant
- **Showing a complete problem in DL(1)**
- **Analyzing the power of non-determinism** – only no IDs prevent from deciding any language (similar to results in PLS)
Towards a Complexity Theory for Local Distributed Computing
[Fraigniaud, Korman, Peleg (JACM) 13]

**Definition:** Distributed Algorithm Alg decides language L

\((G; x) \in L \rightarrow \text{for every identity assignment } \text{Id, Alg. returns “yes” for every node}\)

\((G; x) \notin L, \rightarrow \text{for every identity assignment } \text{Id, Alg. returns “no” for at least one node } v\)

Recall "**Essence of local checkability**" slide (and the [AKY90] criterion for rejection)
Detection using distance $t$ (vs. constant) was studied in self stabilization too

[Beauquier, Delaët, Dolev, Tixeuil, (1998)]

Transient fault detectors.

To what distance does one need to look in the output to discover inconsistency, or
To what distance does one need to look at the history
Other impacts outside self stabilization
(2) Trade off- label size quality of approximation

[Censor-Hillel, Paz, Perry 17]
“Approximating Proof-Labeling Schemes”

- Global predicates: $\psi < \varphi$ ( $\psi, \varphi: \mathcal{F} \rightarrow \mathbb{N}$ )
- If $\psi > \alpha \varphi$ then at least one node detects
- (The [AKY90] criterion)
- Trade-off: approximation ratio vs. PLS label size

* (Useful for self stab. approx. alg. using transformers)
Distributed checking results imply hardness of approximation results

[Das Sarma, Holzer, Kor, Korman, Nanongkai, Pandurangan, Peleg, Wattenhofer11]

(Checking not necessarily local)
(3) Distributed Property Testing

- **Centralized** well established theory
- Deciding "cheap" whether
  - graph (a) has property $P$ or (b) is "far" from $P$

- **Distributed**: [Levi, Medina, Ron (PODC’2018)]
  [Even, Fischer, Fraigniaud, Gonen, Levi, Medina, Montealegre, Olivetti, Oshman, Rapaport, Todinca (DISC’2016)]
  - Different meaning of "cheap"

( Using [AKY90] criterion for rejection)
(4) Computing with advice

[Fraigniaud, Gavoille, Ilcinkas, Pelc]

- **Marker M**: Input: graph G with nodes’ states
  Output: \( \text{Label } M(v) \) \( \forall v \in G_{\text{states}} \)

- **Verifier V**: Input:
  Output: Yes or No

**Interpreter I**

\[ \text{Label}(u) \]

\[ \text{Label}(x) \]

\[ \text{Label}(y) \]

weight \((u,x)\)

state \(u\)

weight \((u,y)\)
Distributed Computing with Advice

Characterizing label length (or “advice”, or “oracle”)

- Broadcast, efficient broadcast, trade-off vs. advice size
- Coloring, 3 coloring of a ring
- MST, topology learning, searching
- “Folding data structures” [KormanK10]

Online Computing with Advice

[Emek,Fraigniaud,Korman,Rosén 2011].
Applications to P2P data structures

- e.g. Distributed Hash Tables
- nodes come and leave, and the data structure should converge
- not entirely self stabilizing
  - (using some stronger assumptions)
- Still using local checkability (and the [AKY] criterion)
  - e.g. [Jacob,Richa,Scheideler, Schmid, Täubig14]:
    - "Skip+: A self-stabilizing skip graph."
    - *JACM* 61, no. 6 (2014): 36.
P2P: algorithms can manipulate (virtual) network graph

Weakly connected directed graph

(TCP) connection graph

Address “known” to Node A ("knowledge graph")
(1) Node A learns C’s addr. From B, thus, directed edge (A → C) is added to knowledge graph.
(1) Node A learns C’s addr. From B, thus, directed edge (A \rightarrow C) is added to knowledge graph.

(2) Node D who knows of C, change PTR\textsubscript{D} to point at C, thus, pointer graph changes.
(1) Node A learns C’s addr. From B, thus, directed edge \((A \rightarrow C)\) is added to knowledge graph.

(2) Node D who knows of C, change \(\text{PTR}_D\) to point at C, thus, pointer graph changes.
Conclusion

• Checking of global state, and especially local checking are natural to self stabilization.

• They started in self stabilization and turned out to capture fundamental notions in distributed computing.

• Those are rich and largely unexplored areas for research, in which people in our community have a built in advantage.
Thanks and
Domo arigato gozai mas