Local Distributed Verification

A. Balliu, G. D'Angelo, P. Fraigniaud, and D. Olivetti

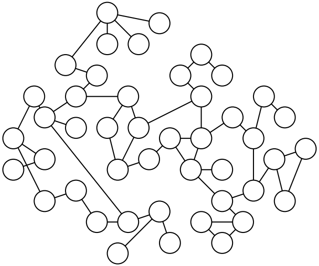
CNRS and University Paris Diderot
GSSI L'Aquila

Goal

- Classify problems according to their difficulty, i.e., build a complexity theory in the distributed setting.
- Build a hierarchy of complexity classes in the context of the LOCAL model.

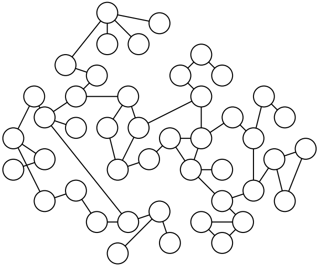
Local Model

- The distributed network is represented by a graph.



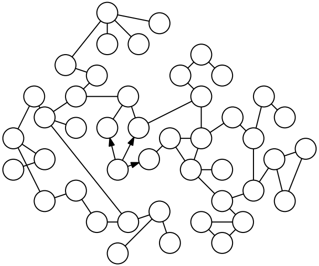
Local Model

- The distributed network is represented by a graph.
- Synchronous model.



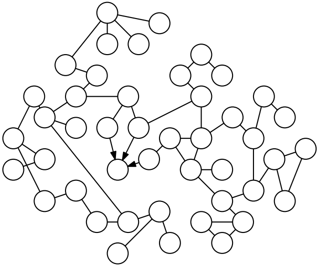
Local Model

- The distributed network is represented by a graph.
- Synchronous model.



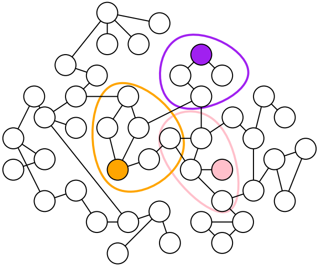
Local Model

- The distributed network is represented by a graph.
- Synchronous model.



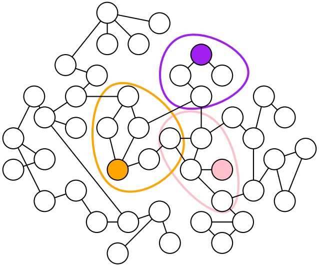
Local Model

- The distributed network is represented by a graph.
- Synchronous model.
- Equivalent to a model where each node sees the network up to distance t.



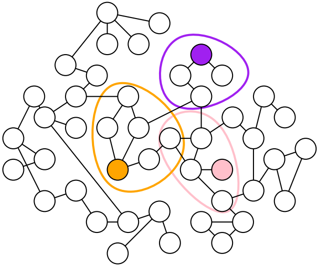
Local Model

- The distributed network is represented by a graph.
- Synchronous model.
- Equivalent to a model where each node sees the network up to distance t.
- The time complexity of a local algorithm $\mathcal{A}$ is determined by the range t that it needs to explore.



Local Model

- The distributed network is represented by a graph.
- Synchronous model.
- Equivalent to a model where each node sees the network up to distance t.
- The time complexity of a local algorithm $\mathcal{A}$ is determined by the range t that it needs to explore.
- We want t to be constant.

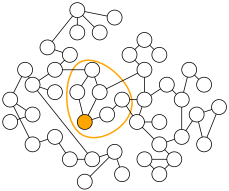


Decision Problems

- Decision Problems: the aim is to decide whether a global input instance satisfies some specific property.

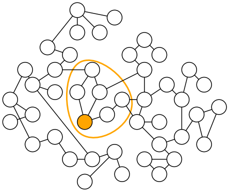
Decision Problems

- Decision Problems: the aim is to decide whether a global input instance satisfies some specific property.
- Each node:
  - gathers its local information from the network;



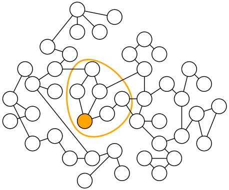
Decision Problems

- Decision Problems: the aim is to decide whether a global input instance satisfies some specific property.
- Each node:
  - gathers its local information from the network;
  - perform some local computation;



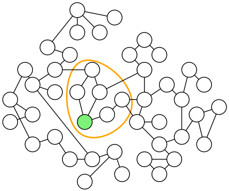
Decision Problems

- Decision Problems: the aim is to decide whether a global input instance satisfies some specific property.
- Each node:
  - gathers its local information from the network;
  - perform some local computation;
  - output its local decision:



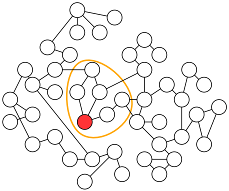
Decision Problems

- Decision Problems: the aim is to decide whether a global input instance satisfies some specific property.
- Each node:
    - gathers its local information from the network;
    - perform some local computation;
    - output its local decision: ”accept”



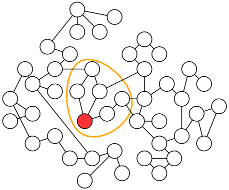
Decision Problems

- Decision Problems: the aim is to decide whether a global input instance satisfies some specific property.
- Each node:
  - gathers its local information from the network;
  - perform some local computation;
  - output its local decision: "accept" or "reject".



Decision Problems

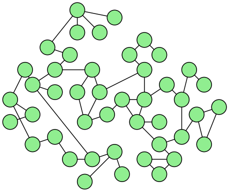
- Decision Problems: the aim is to decide whether a global input instance satisfies some specific property.
- Each node:
  - gathers its local information from the network;
  - perform some local computation;
  - output its local decision: ”accept” or ”reject”.



- $global\_output = \bigwedge_{v \in V} local\_output(v)$.

Decision Problems

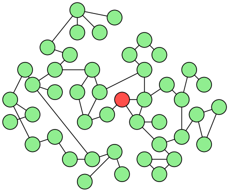
- Decision Problems: the aim is to decide whether a global input instance satisfies some specific property.
- Each node:
  - gathers its local information from the network;
  - perform some local computation;
  - output its local decision: ”accept” or ”reject”.



- $global\_output = \bigwedge_{v \in V} local\_output(v).$

Decision Problems

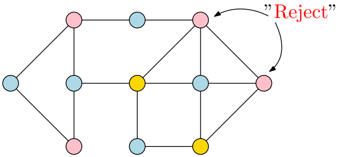
- Decision Problems: the aim is to decide whether a global input instance satisfies some specific property.
- Each node:
  - gathers its local information from the network;
  - perform some local computation;
  - output its local decision: "accept" or "reject".



- $global\_output = \bigwedge_{v \in V} local\_output(v)$.

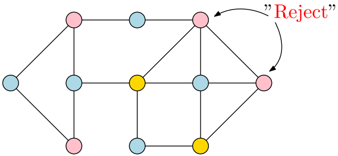
Example: Proper Coloring

- Node input: a color.
- Each node checks the colors of its neighbors.



Example: Proper Coloring

- Node input: a color.
- Each node checks the colors of its neighbors.



- *Local Decision (LD)* is the class of distributed languages that can be locally decided [NS '95].

LD Class

LD is the class of all distributed languages $\mathcal{L}$ for which there exists a local algorithm $\mathcal{A}$ satisfying the following: for every input instance (G, x),

$$\begin{array}{lcl} (G,x) \in \mathcal{L} & \Rightarrow & \forall \mathrm{id} \in \mathsf{ID}(G), \forall u \in V(G), \mathcal{A}(G,x,\mathrm{id},u) = \text{accept} \\ (G,x) \notin \mathcal{L} & \Rightarrow & \forall \mathrm{id} \in \mathsf{ID}(G), \exists u \in V(G), \mathcal{A}(G,x,\mathrm{id},u) = \text{reject} \end{array}$$

Verification Problems

- Verification problem: the aim is to **verify** whether a global input instance satisfies some specific property.

Verification Problems

- Verification problem: the aim is to **verify** whether a global input instance satisfies some specific property.
- Each node:
  - **has a certificate**, unbounded size and independent from the id assignment;

Verification Problems

- Verification problem: the aim is to **verify** whether a global input instance satisfies some specific property.
- Each node:
  - **has a certificate**, unbounded size and independent from the id assignment;
  - gathers its local information from the network;

Verification Problems

- Verification problem: the aim is to **verify** whether a global input instance satisfies some specific property.
- Each node:
  - **has a certificate**, unbounded size and independent from the id assignment;
  - gathers its local information from the network;
  - perform some local computation;

Verification Problems

- Verification problem: the aim is to **verify** whether a global input instance satisfies some specific property.
- Each node:
  - **has a certificate**, unbounded size and independent from the id assignment;
  - gathers its local information from the network;
  - perform some local computation;
  - output its local decision, that is ether ”accept” or ”reject”.

Verification Problems

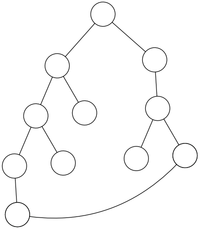
- Verification problem: the aim is to **verify** whether a global input instance satisfies some specific property.
- Each node:
  - **has a certificate**, unbounded size and independent from the id assignment;
  - gathers its local information from the network;
  - perform some local computation;
  - output its local decision, that is ether ”accept” or ”reject”.
- $global\_output = \bigwedge_{v \in V} local\_output(v)$.

Verification Problems

- Verification problem: the aim is to **verify** whether a global input instance satisfies some specific property.
- Each node:
  - **has a certificate**, unbounded size and independent from the id assignment;
  - gathers its local information from the network;
  - perform some local computation;
  - output its local decision, that is ether "accept" or "reject".
- $global\_output = \bigwedge_{v \in V} local\_output(v)$.
- Similar to PLS, but with id-independent certificates.

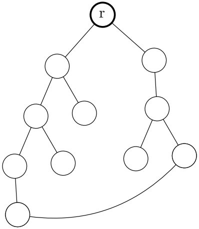
Example: is the given graph a tree?

- Not locally decidable, but locally verifiable.



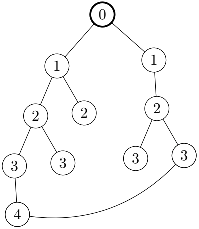
Example: is the given graph a tree?

- Not locally decidable, but locally verifiable.
- Choose a node to be the root.



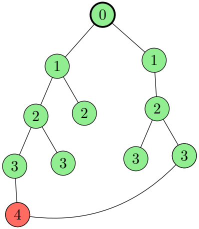
Example: is the given graph a tree?

- Not locally decidable, but locally verifiable.
- Choose a node to be the root.
- Certificate of a node v: its hop-distance from the chosen root.



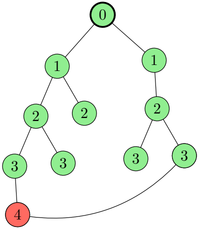
Example: is the given graph a tree?

- Not locally decidable, but locally verifiable.
- Choose a node to be the root.
- Certificate of a node v: its hop-distance from the chosen root.



Example: is the given graph a tree?

- Not locally decidable, but locally verifiable.
- Choose a node to be the root.
- Certificate of a node *v*: its hop-distance from the chosen root.



- *Nondeterministic LD (NLD)* is the class of distributed languages that can be locally verified [FKP '11].

NLD Class

NLD is the class of all distributed languages $\mathcal{L}$ for which there exists a local algorithm $\mathcal{A}$ satisfying the following: for every input instance (G, x),

- $(G, x) \in \mathcal{L} \Rightarrow \exists c \in \mathcal{C}(G), \forall \mathrm{id} \in \mathrm{ID}(G), \forall u \in V(G),$
$$\mathcal{A}(G, x, c, \mathrm{id}, u) = \text{accepts}$$
- $(G, x) \notin \mathcal{L} \Rightarrow \forall c \in \mathcal{C}(G), \forall \mathrm{id} \in \mathrm{ID}(G), \exists u \in V(G),$
$$\mathcal{A}(G, x, c, \mathrm{id}, u) = \text{rejects}$$

NLD Class

NLD is the class of all distributed languages $\mathcal{L}$ for which there exists a local algorithm $\mathcal{A}$ satisfying the following: for every input instance (G, x),

- $(G,x) \in \mathcal{L} \Rightarrow \exists c \in \mathcal{C}(G), \forall \mathrm{id} \in \mathrm{ID}(G), \forall u \in V(G),$
  $\mathcal{A}(G, x, c, \mathrm{id}, u) = \text{accepts}$
- $(G,x) \notin \mathcal{L} \Rightarrow \forall c \in \mathcal{C}(G), \forall \mathrm{id} \in \mathrm{ID}(G), \exists u \in V(G),$
  $\mathcal{A}(G, x, c, \mathrm{id}, u) = \text{rejects}$

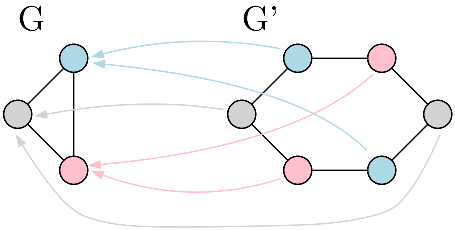
$L \in$ NP if there is a polynomial time algorithm A such that,

$$x \in L \iff \exists c \text{ s.t. } A \text{ accepts } x \text{ with } c.$$

More about NLD

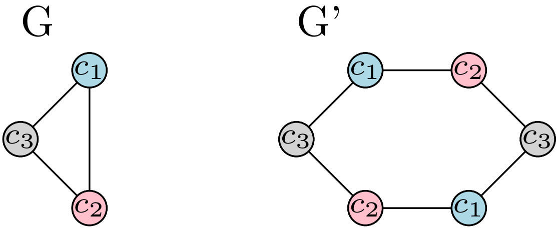
NLD is the class of all problems closed under lift [FKP '11].

- Let (G, x) and (G', x') be two input instances.
- (G', x') is a lift of (G, x) if there exists a function f such that: $f : V(G') \to V(G)$ preserving the local view of each node.



NLD is Closed Under Lift

- Let $\mathcal{L}$ be a language in NLD.
- If $(G, x) \in \mathcal{L} \wedge (G', x')$ is a lift of (G, x), then $(G', x') \in \mathcal{L}$.



Goal

- Build a hierarchy of complexity classes in the distributed setting.
- Distributed hierarchies in other setting:
  - [Reiter '14] in the context of automata;
  - [FFH '16] in a model inspired by the CONGEST one.

Complexity Classes

- LD $= \Sigma_0^{loc} = \Pi_0^{loc}$ (similar to P in the sequential setting).

Complexity Classes

- LD $= \Sigma_0^{loc} = \Pi_0^{loc}$ (similar to P in the sequential setting).
- NLD $= \Sigma_1^{loc}$ (similar to NP in the sequential setting).

Complexity Classes

- LD $= \Sigma_0^{loc} = \Pi_0^{loc}$ (similar to P in the sequential setting).
- NLD $= \Sigma_1^{loc}$ (similar to NP in the sequential setting).
- Σ_k^{loc}: An input instance satisfies a certain property in Σ_k^{loc} iff

$$\exists c_1, \forall c_2, \ldots, Qc_k, \text{ all nodes accept.}$$

Complexity Classes

- LD $= \Sigma_0^{loc} = \Pi_0^{loc}$ (similar to P in the sequential setting).
- NLD $= \Sigma_1^{loc}$ (similar to NP in the sequential setting).
- Σ_k^{loc}: An input instance satisfies a certain property in Σ_k^{loc} iff

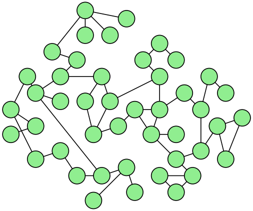
$$\exists c_1, \forall c_2, \ldots, Qc_k, \text{ all nodes accept.}$$

- Π_k^{loc}: An input instance satisfies a certain property in Π_k^{loc} iff

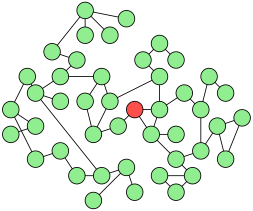
$$\forall c_1, \exists c_2, \ldots, Qc_k, \text{ all nodes accept.}$$

Complementary Classes

In a class:

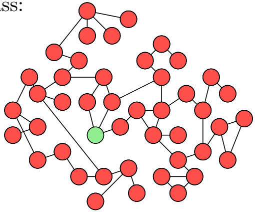


A globaly accepted input instance.

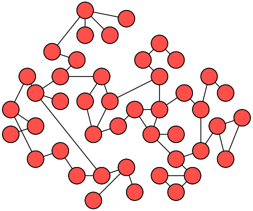


A globaly rejected input instance.

In a complementary class:



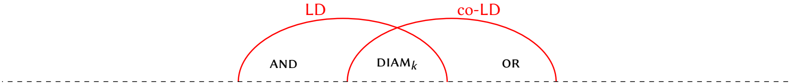
A globaly accepted input instance.



A globaly rejected input instance.

Lever 0 of the Hierarchy

- AND : $|\{u \in V(G) : x(u) = 1\}| = 0$
- OR : $|\{u \in V(G) : x(u) = 1\}| \geq 1$



Π_1^{loc}: The Role of the Last Universal Quantifier

- Π_1^{loc}:

$$(G, x) \in \mathcal{L} \Leftrightarrow \forall c \text{ all nodes accept.}$$

- LD:

$$(G, x) \in \mathcal{L} \Leftrightarrow \text{ all nodes accept}$$

Π_1^{loc}: The Role of the Last Universal Quantifier

- Π_1^{loc}:

$$(G, x) \in \mathcal{L} \Leftrightarrow \forall c \text{ all nodes accept.}$$

- LD:

$$(G, x) \in \mathcal{L} \Leftrightarrow \text{ all nodes accept}$$

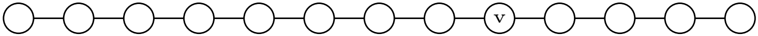
- Problems that can be solved only if a specific node knows (an upper bound of) the size of the network!

ITER

- Let f be a function and a and b two non-negative integers.

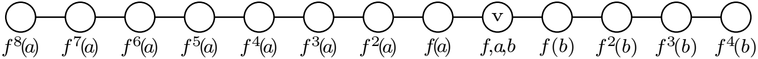
ITER

- Let f be a function and a and b two non-negative integers.
- A configuration in ITER consists in a path $P = LvR$ with a special node v (*pivot*).



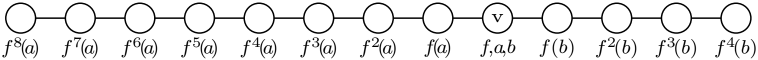
ITER

- Let f be a function and a and b two non-negative integers.
- A configuration in ITER consists in a path $P = LvR$ with a special node v (*pivot*).
- Nodes in L (resp., in R) are given as input $f, f^i(a)$ (resp., $f, f^i(b)$); to v is given in input f, a, b.



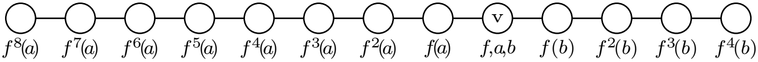
ITER

- Let f be a function and a and b two non-negative integers.
- A configuration in ITER consists in a path $P = LvR$ with a special node v (*pivot*).
- Nodes in L (resp., in R) are given as input $f, f^i(a)$ (resp., $f, f^i(b)$); to v is given in input f, a, b.
- f is s.t. $f(0) = 0$ and $f(1) = 1$

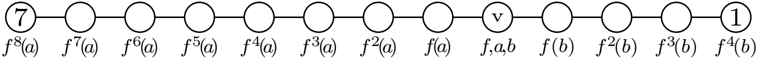


ITER

- Let f be a function and a and b two non-negative integers.
- A configuration in ITER consists in a path $P = LvR$ with a special node v (*pivot*).
- Nodes in L (resp., in R) are given as input $f, f^i(a)$ (resp., $f, f^i(b)$); to v is given in input f, a, b.
- f is s.t. $f(0) = 0$ and $f(1) = 1$
- An input instance is in ITER if and only if:
  - $f^{|L|}(a) \in \{0, 1\}$ and $f^{|R|}(b) \in \{0, 1\}$
  - $f^{|L|}(a) = 0$ or $f^{|R|}(b) = 0$

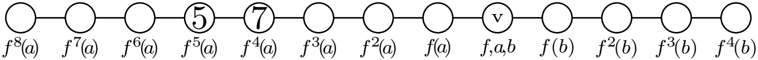


ITER



- An endpoint node rejects only if it has in input something different from 1 or 0; otherwise accepts.
- In this case, the left endpoint node rejects.

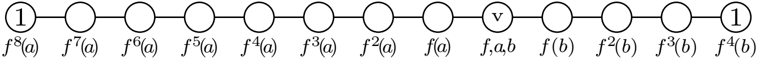
ITER



$$f(7) = 6$$

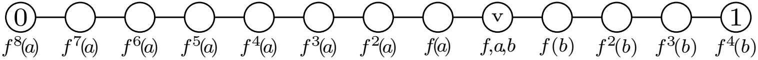
- Nodes reject if they notice local inconsistencies.

ITER



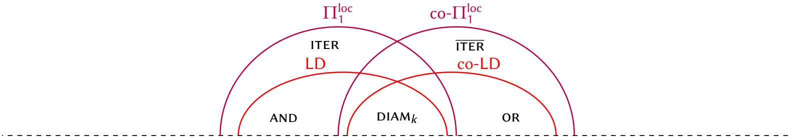
- $(G, x) \notin \mathcal{L} \Rightarrow \exists c$ s.t. at least one node rejects.
- v rejects only if $f^{|L|}(a) = f^{|R|}(b) = 1$; otherwise accepts.
- Certificate of node v: un upper bound of the size of the network.

ITER

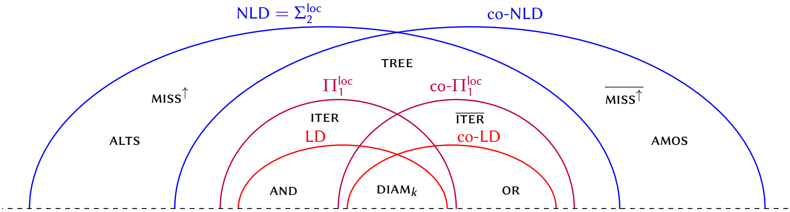


- $(G, x) \in \mathcal{L} \Rightarrow \forall c$ s.t. all nodes accept.
- Whatever certificate v has, it will never compute $f^{|L|}(a) = f^{|R|}(b) = 1$.

Local Hierarchy



Local Hierarchy



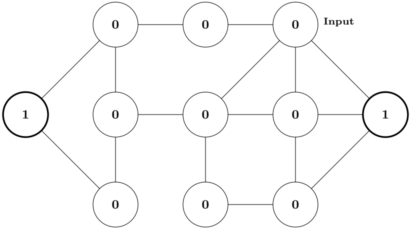
Π_2^{loc} Class

- Π_2 class: An input instance satisfies a certain property in Π_2 iff

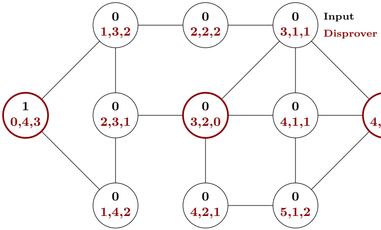
$$\forall c_1, \exists c_2, \text{ all nodes accept.}$$

- Two party game between a *disprover* and a *prover*.

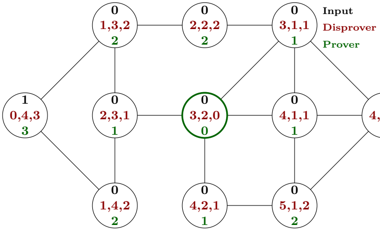
Exactly Two Selected



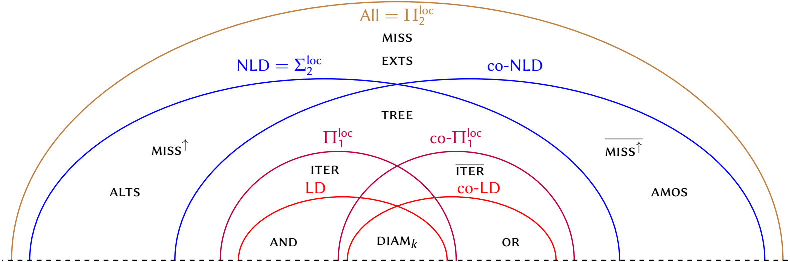
Exactly Two Selected



Exactly Two Selected



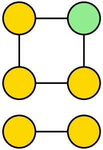
Local Hierarchy



$$\mathsf{LD} \subset \Pi_1^{loc} \subset \mathsf{NLD} = \Sigma_2^{loc} \subset \Pi_2^{loc} = \mathsf{All} \text{ (all inclusions are strict).}$$

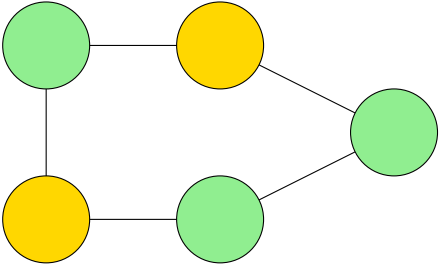
MISS: a Π_2^{loc}-complete Problem

- Every node u of (G, x) is given a family $\mathcal{F}(u)$ of input instances, each described by
  - An adjacency matrix representing a graph;
  - array representing the inputs to the nodes of that graph.



MISS: a Π_2^{loc}-complete Problem

- Every node u of (G, x) is given a family $\mathcal{F}(u)$ of input instances, each described by
  - An adjacency matrix representing a graph;
  - array representing the inputs to the nodes of that graph.
- Every node u has an input string $x'(u) \in \{0,1\}^*$ (notice that (G, x') is also an input instance).

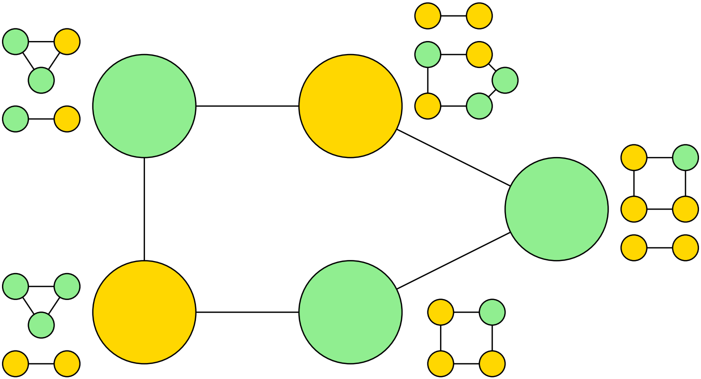


MISS: a Π_2^{loc}-complete Problem

- Every node u of (G, x) is given a family $\mathcal{F}(u)$ of input instances, each described by
  - An adjacency matrix representing a graph;
  - array representing the inputs to the nodes of that graph.
- Every node u has an input string $x'(u) \in \{0,1\}^*$ (notice that (G, x') is also an input instance).
- The current (G, x) is legal if (G, x') is missing in all families $\mathcal{F}(u)$ for every $u \in V(G)$.

$$\text{MISS} = \{(G, x) : \forall u \in V(G), x(u) = (\mathcal{F}(u), x'(u)) \text{ and } (G, x') \notin \mathcal{F}\}$$

MISS: a Π_2^{loc}-complete Problem



Reduction to MISS

- Each node u with identity $\mathrm{id}(u)$ and input $x(u)$ computes its *width* $\omega(u) = 2^{|\mathrm{id}(u)|+|x(u)|}$.

Reduction to MISS

- Each node u with identity $\mathrm{id}(u)$ and input $x(u)$ computes its *width* $\omega(u) = 2^{|\mathrm{id}(u)|+|x(u)|}$.
- Each node u generates $\mathcal{F}(u)$, i.e., all $(H, y) \notin \mathcal{L}$
  - At most $\omega(u)$ nodes;
  - $y(v)$ has value at most $\omega(u)$.

Reduction to MISS

- Each node u with identity $\mathrm{id}(u)$ and input $x(u)$ computes its *width* $\omega(u) = 2^{|\mathrm{id}(u)|+|x(u)|}$.
- Each node u generates $\mathcal{F}(u)$, i.e., all $(H, y) \notin \mathcal{L}$
  - At most $\omega(u)$ nodes;
  - $y(v)$ has value at most $\omega(u)$.
- If $(G, x) \in \mathcal{L}$
  - $(G, x) \notin \mathcal{F}$ since only illegal instances are in $\mathcal{F}$;
  - all nodes will accept.

Reduction to MISS

- Each node u with identity $\mathrm{id}(u)$ and input $x(u)$ computes its *width* $\omega(u) = 2^{|\mathrm{id}(u)|+|x(u)|}$.
- Each node u generates $\mathcal{F}(u)$, i.e., all $(H, y) \notin \mathcal{L}$
  - At most $\omega(u)$ nodes;
  - $y(v)$ has value at most $\omega(u)$.
- If $(G, x) \in \mathcal{L}$
  - $(G, x) \notin \mathcal{F}$ since only illegal instances are in $\mathcal{F}$;
  - all nodes will accept.
- If $(G, x) \notin \mathcal{L}$
  - There exists u with $\mathrm{id}(u)$ or $x(u)$ big enough, which guarantees that u generates the graph G, i.e., $(G, x) \in \mathcal{F}(u)$;
  - at least one node will reject.

Open Problems

- Unbounded size id-independent certificates:
  - find a complete problem for Π_1^{loc} and co-Π_1^{loc};
  - find a problem in the intersection between the classes Π_1^{loc} and co-Π_1^{loc}.
- Bounded size ($O(\log n)$) id-dependent certificates
  - we don't know if the hierarchy collapses;
  - there are no separating problems for Σ_2^{loc} and Σ_3^{loc} (neither for classes higher in the hierarchy).

Thank you!