Bio-inspired computation, communication topologies, and computational complexity

Antonio E. Porreca
https://aeporreca.org
Outline

• The first and second machine classes
• Examples of parallel computing models
• Membrane systems
• Complexity theory of membrane systems
• Communication topologies and their role
• A research project
The first machine class

Definition (van Emde Boas). A machine model is first class iff it simulates and is simulated by a Turing machine in polynomial time

• Turing machines
• Random access machines with + and −
• Cellular automata with finite initial configuration
The second machine class

**Definition (van Emde Boas).** A machine model is second class iff it characterises **PSPACE** in polynomial time (deterministically and nondeterministically)

- Alternating Turing machines
- Vector machines
- Random access machines with $+ - \times \div$
- Parallel processes with `fork()` with unbounded number of processors
- Cellular automata over hyperbolic grids
Nondeterministic Turing machine
Nondeterministic Turing machine
Alternating Turing machine
Hyperbolic CAs

Hyperbolic tiling by Theon, used under CC BY-SA 3.0
Hyperbolic CAs

Hyperbolic tiling by Theon, used under CC BY-SA 3.0
Hyperbolic CAs

Hyperbolic tiling by Theon, used under CC BY-SA 3.0
Hyperbolic CAs

Hyperbolic tiling by Theon, used under CC BY-SA 3.0
Hyperbolic CAs

Hyperbolic tiling by Theon, used under CC BY-SA 3.0
Hyperbolic CAs

Hyperbolic tiling by Theon, used under CC BY-SA 3.0
Hyperbolic CAs

Hyperbolic tiling by Theon, used under CC BY-SA 3.0
Hyperbolic CAs

Hyperbolic tiling by Theon, used under CC BY-SA 3.0
Hyperbolic CAs

PSPACE

Hyperbolic tiling by Theon, used under CC BY-SA 3.0
First and second machine class

Many “concrete” sequential and parallel computing models are either first or second class machines
Membrane systems
Simple chemical reactions

\[ ab \rightarrow cd \]
Simple chemical reactions

$ab \rightarrow cd$
Communication between regions

\[ a \ [bc] \rightarrow [d] \ e \]
Communication between regions

\[ a \ [bc] \rightarrow [d] \ e \]
Monodirectional communication between regions

\[ [a] \rightarrow [ ] b \]
Monodirectional communication between regions

\[[a] \rightarrow [ ] b\]
Membrane division

\[ [a] \rightarrow [b] [c] \]
Membrane division

\[ [a] \rightarrow [b] [c] \]
Elementary membrane division

\[ [a] \rightarrow [b] [c] \]
Elementary membrane division

\[ [a] \rightarrow [b] [c] \]
Register/counter machines

1: dec \( x, 2, 4 \)
2: inc \( y, 3 \)
3: inc \( y, 1 \)
4: halt
Register/counter machines

1: dec $x$, 2, 4
2: inc $y$, 3
3: inc $y$, 1
4: halt
Register/counter machines

1: dec \(x, 2, 4\)
2: inc \(y, 3\)
3: inc \(y, 1\)
4: halt

\[p_2 \rightarrow y \ p_3\]
Register/counter machines

1: dec $x$, 2, 4
2: inc $y$, 3
3: inc $y$, 1
4: halt

$p_2 \rightarrow y \ p_3$

$p_1 \rightarrow p'_1 \bar{x}$
$p'_1 \rightarrow p''_1$

$x \bar{x} \rightarrow x'$
$p''_1 \bar{x} \rightarrow p_4$

$p'_1 \ x' \rightarrow p_2$
(Semi-)uniform families of membrane systems

\[ x \in \Sigma^* \]

M

\[ \Pi_x \]
Simulating Turing machines efficiently
Simulating Turing machines efficiently
Simulating Turing machines efficiently

\[ \delta(q, b) = (q', a, +1) \]
Simulating Turing machines efficiently

\[ \delta(q, b) = (q', a, +1) \]

\[ q_i \ b_i \rightarrow q'_{i+1} \ a_i \quad 1 \leq i \leq n \]
Simulating Turing machines efficiently

\[ \delta(q, b) = (q', a, +1) \]

\[ q_i b_i \rightarrow q'_{i+1} a_i \quad 1 \leq i \leq n \]
Simulating nondeterministic Turing machines

\[ \delta(q, b) = \begin{cases} (q', a, +1) \\ (q'', b, -1) \end{cases} \]

\[ [q_i \ b_i] \rightarrow [q'_{i+1} \ a_i] [q''_{i-1} \ b_i] \]
Simulating nondeterministic Turing machines
Simulating nondeterministic Turing machines
Simulating nondeterministic Turing machines
Simulating nondeterministic Turing machines
Simulating nondeterministic Turing machines
Simulating alternating Turing machines
Simulating alternating Turing machines
Simulating alternating Turing machines
Simulating alternating Turing machines

$\exists$ existential nondeterministic choice
Simulating alternating Turing machines
Simulating alternating Turing machines

∃

universal nondeterministic choices
Simulating alternating Turing machines

∃ universal nondeterministic choices
Simulating alternating Turing machines

universal
nondeterministic
choices
Simulating alternating Turing machines
Simulating alternating Turing machines
Simulating alternating Turing machines
Simulating alternating Turing machines
Simulating alternating Turing machines
The polynomial hierarchy

P \subseteq \text{NP} \subseteq \text{NP}^\text{NP} \subseteq \ldots \subseteq \text{PSpace} \subseteq \text{PH}
(Semi-)uniform families of membrane systems

\[ x \in \Sigma^* \]

\[ \Pi_x \]
The polynomial hierarchy

\[
\begin{align*}
P & \subseteq NP \\
NP & \subseteq NP^{NP} \\
NP^{NP} & \subseteq NP^{NP^{NP}} \\
\vdots & \\
PSPACE & \subseteq \text{PH} \\
\end{align*}
\]
Simulating Turing machines with oracles
Simulating Turing machines with oracles
Simulating Turing machines with oracles
Simulating Turing machines with oracles
Simulating Turing machines with oracles
Simulating Turing machines with oracles
The polynomial hierarchy

\[
\begin{array}{c}
P \\
NP \\
NP^{NP} \\
\vdots \\
PSPACE
\end{array}
\]
The polynomial hierarchy

\[ \text{P}^\text{P} = \text{P} \]

\[ \text{P}^\text{NP} \supseteq \text{NP} \]

\[ \text{P}^\text{NP}^\text{NP} \supseteq \text{NP}^\text{NP} \]

\[ \vdots \]

\[ \text{P}^\text{PSPACE} = \text{PSPACE} \]
Nondeterministic Turing machine
Nondeterministic Turing machine
Counting Turing machine
Counting Turing machine

\[ f: \Sigma^* \rightarrow \mathbb{N} \]
Counting Turing machine

\[ f : \Sigma^* \rightarrow \mathbb{N} \]

\#P

![Diagram with nodes and edges representing a counting Turing machine.](image-url)
Counting Turing machine

\[ f : \Sigma^* \rightarrow \mathbb{N} \]

\#P
Simulating counting Turing machines
Simulating counting Turing machines
Simulating counting Turing machines
Simulating counting Turing machines
Simulating counting Turing machines
Simulating counting Turing machines
Simulating counting Turing machines
Simulating counting Turing machines
Simulating counting Turing machines
Simulating counting Turing machines
Simulating counting Turing machines
Simulating counting Turing machines
Simulating counting Turing machines
Simulating Turing machines with \#P oracles
Simulating Turing machines with $\#P$ oracles
Simulating Turing machines with \#P oracles
Simulating Turing machines with $\#P$ oracles
Simulating Turing machines with $\#P$ oracles
Simulating Turing machines with $\#P$ oracles
Simulating Turing machines with \#P oracles
Simulating Turing machines with $\#P$ oracles
Simulating Turing machines with \#P oracles
Simulating Turing machines with $\mathcal{NP}$ oracles
Simulating Turing machines with \#P oracles
Simulating Turing machines with \#P oracles
Simulating Turing machines with \#P oracles
Simulating Turing machines with \( \#P \) oracles
Simulating Turing machines with \#P oracles
Simulating Turing machines with \#P oracles
Simulating Turing machines with \#P oracles
The counting hierarchy

\[
\begin{array}{c}
\text{P} \\
\text{NP} \\
\text{NP}^\text{NP} \\
\vdots \\
\text{PSPACE} \\
\end{array}
\]

includes
The counting hierarchy

\[ \text{Toda's Theorem} \]

\[ \text{P} \supseteq \text{P}^\# \supseteq \text{PH} \]

\[ \text{P}^\# \supseteq \text{P}^\# \supseteq \cdots \]

\[ \supseteq \text{PSPACE} \]
Exact characterisation of $\mathsf{P^{#P}}$
Exact characterisation of $\mathsf{P}^{\mathsf{#P}}$

polynomial-time simulation by TM
Exact characterisation of $P^{\#P}$

polynomial-time simulation by TM

oracle queries
Query simulating dividing membranes

If the dividing membranes receive the sequence of inputs \((m_1, \ldots, m_t)\), how many instances of object \(a\) are sent out at time \(t + 1\)?
Query simulating dividing membranes

If the dividing membranes receive the sequence of inputs \((m_1, \ldots, m_t)\), how many instances of object \(a\) are sent out at time \(t + 1\)?
If the dividing membranes receive the sequence of inputs \((m_1, \ldots, m_t)\), how many instances of object \(a\) are sent out at time \(t + 1\)?
Query simulating dividing membranes

If the dividing membranes receive the sequence of inputs \((m_1, \ldots, m_t)\), how many instances of object \(a\) are sent out at time \(t + 1\)?
If the dividing membranes receive the sequence of inputs \((m_1, \ldots, m_t)\), how many instances of object \(a\) are sent out at time \(t + 1\)?
Query simulating dividing membranes

If the dividing membranes receive the sequence of inputs \((m_1, \ldots, m_t)\), how many instances of object \(a\) are sent out at time \(t + 1\)?
Query simulating dividing membranes

If the dividing membranes receive the sequence of inputs \((m_1, \ldots, m_t)\), how many instances of object \(a\) are sent out at time \(t + 1\)?
Exact characterisation of $\mathsf{P}^\#\mathsf{P}$

**Theorem.** Membrane systems where membranes can only divide if they do not contain recursively other membranes characterise $\mathsf{P}^\#\mathsf{P}$ in polynomial time.
Solving PSPACE efficiently in the “binary tree space”
Solving PSPACE efficiently in the “binary tree space”
Solving PSPACE efficiently in the “binary tree space”
Solving PSPACE efficiently in the “binary tree space”
Solving PSPACE efficiently in the “binary tree space”
Solving PSPACE efficiently in the “binary tree space”
Communication topologies and complexity classes

P

PSPACE

P#P

\ldots
Automata networks over infinite graphs
Automata networks over infinite graphs
Automata networks over infinite graphs

\[ f : \text{Multisets}(Q) \rightarrow Q \]
1D cellular automata as generalised Turing machines
1D cellular automata as generalised Turing machines

\[
\begin{array}{c}
a \ b \ b \ a \\
\end{array}
\]

\[
\begin{array}{c}
a \ b \ (b, q) \ a \\
\end{array}
\]
Things to do

- Choose restrictions on the class of local transition functions $f : \text{Multisets}(Q) \rightarrow Q$
e.g., threshold functions
- Choose a way to encode the input in the initial configuration
Defining new complexity classes

**Definition.** Given an infinite graph $G$, let
- $P(G)$ be the problems solved by automata networks over $G$ in polynomial time
- $\text{PSPACE}(G) =$ polynomial space over $G$
- $\text{EXPTIME}(G) =$ exponential time over $G$
- etc.

but also
- $\text{LOGTIME}(G) =$ logarithmic time over $G$
- etc.

because automata networks are parallel
Preliminary results

- $P(\text{linear graph}) = P$
- $\text{PSPACE}(\text{linear graph}) = \text{PSPACE}$

because of the equivalence with Turing machines

- $P(\text{efficiently computable graph in } \mathbb{R}^d) = P$
Expected results

- $P(\text{infinite binary tree}) = \text{PSPACE}$
- $P(\text{infinite star or variant thereof}) = P^{\#P}$

- $P(\text{non-computable graph})$ includes undecidable problems
Hopeful developments of the theory

- Find graphs (or classes of graphs) characterising all standard complexity classes
- Find new intermediate complexity classes using “natural” graphs
Hopeful developments of the theory

- Find graphs (or classes of graphs) characterising all standard complexity classes
- Find new intermediate complexity classes using “natural” graphs

- Find algorithms working on all graphs or on certain classes of graphs
- Discover how graph-theoretic or geometric properties of the graphs can speed up or slow down algorithms e.g., algorithms running in time $\Theta(n^{f(d)})$ in $\mathbb{R}^d$
Applications

• Same applications as automata networks (e.g., biology)
• Theory (and practice) of distributed algorithms
• Low-level hardware design
• Machine learning (variants of deep learning? non-Euclidean learning?)

and, of course

• Computability and complexity theory
Thanks for your attention!

Merci de votre attention!

Any questions?