Two-Process Systems

Companion slides for
Distributed Computing
Through Combinatorial Topology
Maurice Herlihy & Dmitry Kozlov & Sergio Rajsbaum
Two-Process Systems

Two-process systems can be captured by elementary graph theory. A gentle introduction to more general structures needed later for larger systems.
Road Map

Elementary Graph Theory

Tasks

Models of Computation

Approximate Agreement

Task Solvability
Road Map

Elementary Graph Theory

Tasks

Models of Computation

Approximate Agreement

Task Solvability
A Vertex
A Vertex

Combinatorial: an element of a set.
A Vertex

**Combinatorial:** an element of a set

**Geometric:** a point in Euclidean Space
An Edge
An Edge

Combinatorial: a set of two vertexes.
An Edge

Combinatorial: a set of two vertexes

Geometric: line segment joining two points
A Graph
A Graph

Combinatorial: a set of sets of vertices.
A Graph

Combinatorial: a set of sets of vertices

Geometric: points joined by line segments
Graphs

finite set $V$ with a collection $\mathcal{G}$ of subsets of $V$, 
Graphs

finite set $V$ with a collection $G$ of subsets of $V$

vertices

simplices (singular: simplex)
Graphs

finite set $V$ with a collection $\mathcal{G}$ of subsets of $V$,

(1) If $X \in \mathcal{G}$, then $|X| \leq 2$
Graphs

Finite set $V$ with a collection $\mathcal{G}$ of subsets of $V$,

(1) If $X \in \mathcal{G}$, then $|X| \leq 2$

Vertex: $|X| = 1$
Edge: $|X| = 2$
Graphs

finite set $V$ with a collection $\mathcal{G}$ of subsets of $V$,

(1) If $X \in \mathcal{G}$, then $|X| \leq 2$

(2) for all $v \in V$, $\{v\} \in \mathcal{G}$
Graphs

finite set $V$ with a collection $\mathcal{G}$ of subsets of $V$,

(1) If $X \in \mathcal{G}$, then $|X| \leq 2$

(2) for all $v \in V$, $\{v\} \in \mathcal{G}$

(3) for all $X \in \mathcal{G}$, and $Y \subset X$, $Y \in \mathcal{G}$
Dimension

\[ \text{dim}(X) = |X| - 1. \]
Pure Graphs

pure of dim 0

pure of dim 1
Graph Coloring
Graph Coloring

\[ \chi: G \rightarrow C \]
Graph Coloring

\[ \chi : \mathcal{G} \rightarrow C \]

for each edge \( (s_0, s_1) \in \mathcal{G}, \chi(s_0) \neq \chi(s_1) \).
Graph Coloring

\(\chi: \mathcal{G} \rightarrow \mathcal{C}\) for each edge \((s_0, s_1) \in \mathcal{G}, \chi(s_0) \neq \chi(s_1)\).

chromatic graphs usually process names
Graph Labeling

Distributed Computing through
Combinatorial Topology
Graph Labeling

\[ f: G \rightarrow L \]
Graph Labeling

$f: G \rightarrow L$

usually values from some domain
Labeled Chromatic Graph

name(s) = $\chi(s)$

view(s) = $f(s)$
Simplicial Maps

Vertex-to-vertex map …

that also sends edges to edges.
Rigid Simplicial Maps

A simplicial map can send an edge to a vertex …
Rigid Simplicial Maps

A simplicial map can send an edge to a vertex ...

A simplicial map that sends distinct vertices to distinct vertices is \textit{rigid}.
A Path Between two Vertices
A graph is *connected* if there is a path between every pair of vertices.
A graph is *connected* if there is a path between every pair of vertices.
Theorem

The image of a connected graph under a simplicial map is connected.
Carrier Maps

For graphs $G$, $H$, a carrier map

$$\Phi: G \rightarrow 2^H$$

Carries each simplex of $G$ to a subgraph of $H$ …

satisfying monotonicity:
for all $\sigma, \tau \in G$, if $\sigma \subseteq \tau$, then $\Phi(\sigma) \subseteq \Phi(\tau)$. 
Strict Carrier Maps

Monotonicity

For all $\sigma, \tau \in \mathcal{G}$, if $\sigma \subseteq \tau$, then $\Phi(\sigma) \subseteq \Phi(\tau)$.

Equivalent to …

$\Phi(\sigma \cap \tau) \subseteq \Phi(\sigma) \cap \Phi(\tau)$

Definition

$\Phi$ is strict if $\Phi(\sigma \cap \tau) = \Phi(\sigma) \cap \Phi(\tau)$
Road Map

Elementary Graph Theory

Tasks

Models of Computation

Approximate Agreement

Task Solvability
Two Processes

Hello! I’m Alice

Hello! I’m Bob
Informal Task Definition

Processes start with input values …

They communicate …

They halt with output values …

legal for those inputs.
Formal Task Definition

Input graph $\mathcal{I}$

all possible assignments of input values
Formal Task Definition

Input graph $I$
- all possible assignments of input values

Output graph $O$
- all possible assignments of output values
Formal Task Definition

Input graph $\mathcal{I}$
- all possible assignments of input values

Output graph $\mathcal{O}$
- all possible assignments of output values

Carrier map $\Delta: \mathcal{I} \rightarrow 2^\mathcal{O}$
- all possible assignments of output values for each input
Task Input Graph: Consensus
Task Input Graph

---

Distributed Computing through Combinatorial Topology
Task Input Graph

Pure

Colored by process names

Labeled by input values
Task Output Graph

1 ——— 1

0

0 ——— 0
Task Carrier Map

\[ I \xrightarrow{\Delta} 2^\emptyset \]
Task Carrier Map

If Bob runs alone with input 1 … then he decides output 1.

\[ \Delta: \mathcal{I} \rightarrow 2^0 \]
If Bob and Alice both have input 1 …

then they both decide output 1.
If Bob has 1 and Alice 0 …

then they must agree, on either one.
Example: Coordinated Attack

Alice and Bob win if they both attack together.
Input Graph

Attack at dawn!

\[ \mathcal{I} \]

Attack at noon!

Indifferent
Output Graph

dawn!

failed!

noon!
Carrier Map

$I$

$\Delta$

$\emptyset$
Carrier Map

dawn!

\[ I \]

\[ \Delta \]

dawn!

failed!
Carrier Map

noon!

failed!

noon!

failed!

noon!
Carrier Map

\[ I \]

\[ \Delta \]

\[ O \]

dawn!

dawn!
Example: Coordinated Attack

Alice

Bob

Enemy
Example: Coordinated Attack

Alice and Bob realize that they do not need to agree on an exact time …
Example: Coordinated Attack

Alice and Bob realize that they do not need to agree on an exact time ... they will win if attack times are sufficiently close.
Coordinated Attack Graphs

\[ I \]

\[ \Delta \]

\[ O \]
Coordinated Attack Graphs

$\mathcal{I}$

0 1

0 1/5 2/5 3/5 4/5 1

$\mathcal{O}$

Distributed Computing through Combinatorial Topology
Coordinated Attack Graphs

\[ \mathcal{I} \]

\[ \Delta \]

\[ \emptyset \]
Coordinated Attack Graphs

\[ I \]

\[ \Delta \]

\[ 0 \quad 1/5 \quad 2/5 \quad 3/5 \quad 4/5 \quad 1 \]
Road Map

- Elementary Graph Theory
- Tasks
- Models of Computation
- Approximate Agreement
- Task Solvability
Protocols

Models of Computation
Alice’s Protocol

shared mem array 0..1 of Value
view: Value := my input value;
for i: int := 0 to L do
    mem[A] := view;
    view := view + mem[A] + mem[B];
return δ(view)

Finite program

Bob’s protocol is symmetric
Alice’s Protocol

```plaintext
shared mem array 0..1 of Value

for i: int := 0 to L do
    mem[A] := view;
    view := view + mem[B];
return δ(view)
```
shared mem array 0..1 of Value
view: Value := my input value;
for i: int
   mem[A] := view;
   view := view + mem[B];
return δ(view)
shared mem array 0..1 of Value
view: Value := my input value;
for i: int := 0 to L do
    mem[A] := view;
    view := view + mem[B];
return δ(view)

Run for L layers
Alice’s Protocol

shared mem array 0..1 of Value
view: Value := my input value;
for i: int := 0 to L do
    mem[A] := view;
    view := view + mem[B];
return

Alice writes her value, read Bob’s value, and concatenate it to my view
shared mem array 0..1 of Value
view: Value := my input value;
for i: int := 0 to L do
  mem[A] := view;
  view := view + mem[B];
return δ(view)

Alice writes her value, read Bob’s value, and concatenate it to my view.

(full-information protocol)

Distributed Computing through Combinatorial Topology
Alice’s Protocol

shared mem array 0..1 of Value
view: Value := my input value;
for i: int := 0 to L do
    mem[A] := view;
    view := view + mem[B];
return \(\delta(\text{view})\)

finally, apply task-specific decision map to view
Formal Protocol Definition

Input graph $\mathcal{I}$

all possible assignments of input values
Formal Protocol Definition

Input graph $\mathcal{I}$
- all possible assignments of input values

Protocol graph $\mathcal{P}$
- all possible process views after execution
Formal Protocol Definition

Input graph $\mathcal{I}$
all possible assignments of input values

Protocol graph $\mathcal{P}$
all possible process views after execution

Carrier map $\Xi: \mathcal{I} \rightarrow 2^\mathcal{O}$
all possible assignments of views
One-Round Protocol Graph

\( \mathcal{I} \)

\[ \begin{array}{c}
0 \\
\downarrow \\
01 \\
\downarrow \\
01 \\
\downarrow \\
\bot 1 \\
\end{array} \]

\( \mathcal{P} \)
One-Round Protocol Graph

\[ \mathcal{P} \]

Colored by process names

Labeled with final views
One-Round Protocol Graph

Alice finishes before Bob starts, doesn’t see his value
Alice and Bob run together, she sees his value.
One-Round Protocol Graph

/alice finishes, then bob starts/
One-Round Protocol Graph

Alice and Bob run together

Distributed Computing through Combinatorial Topology
One-Round Protocol Graph

Bob can’t tell whether Alice saw him
Execution Carrier Map
Execution Carrier Map

\[ \Xi : I \rightarrow 2^P \]

\[ \Xi(\sigma) \cap \Xi(\tau) = \Xi(\sigma \cap \tau) \]

strict carrier map
The Decision Map

Protocol graph

Output graph
All Together

\[ \Delta \]

\[ \delta \]

Distributed Computing through Combinatorial Topology
Definition

Decision map \( \delta \) is *carried by carrier map* \( \Delta \) if

- for each input vertex \( s \),
  \[ \delta(\Xi(s)) \subseteq \Delta(s) \]

- for each input edge \( \sigma \),
  \[ \delta(\Xi(\sigma)) \subseteq \Delta(\sigma) \]
Solving a Task

Definition

The protocol \((I,\mathcal{P},\Xi)\) solves the task \((I, O, \Delta)\)
Solving a Task

**Definition**

The protocol \((I, P, \Xi)\) solves the task \((I, O, \Delta)\)

if there is …
Solving a Task

Definition

The protocol \((\mathcal{I}, \mathcal{P}, \Xi)\) solves the task \((\mathcal{I}, \mathcal{O}, \Delta)\) if there is ... a simplicial decision map \(\delta: \mathcal{P} \rightarrow \mathcal{O}\)
Solving a Task

Definition

The protocol \((I, P, \Xi)\) solves the task \((I, O, \Delta)\) if there is ...

a simplicial decision map

\[\delta: P \rightarrow O\]

such that \(\delta\) is carried by \(\Delta\).

(\(\delta\) agrees with \(\Delta\))
Layered Read-Write Model
Layered Read-Write Protocol

shared mem array 0..1,0..L of Value
view: Value := my input value;
for i: int := 0 to L do
  mem[i][A] := view;
  view := view + mem[A] + mem[B];
return δ(view)
Layered Read-Write Protocol

shared mem array 0..1,0..L of Value
view: Value := my input value;
for i: int := 0 to L do
    mem[i][A] := view;
    view := view + mem[A] + mem[B];
return δ(view)

As before, run for L layers
Layered Read-Write Protocol

shared mem array 0..1,0..L of Value
view: Value := my input value;
for i: int := 0 to L do
  mem[i][A] := view;
  view := view + mem[A] + mem[B];
return δ(view)

Each layer uses a distinct, “clean” memory
Layered R-W Protocol Graph

\[ \mathcal{I} \]

\[ \mathcal{P} \]
Layered R-W Protocol Graph

\[ \mathcal{I} \xrightarrow{E} \mathcal{P} \]

\( \mathcal{P} \) is always a subdivision of \( \mathcal{I} \)
Road Map

Elementary Graph Theory

Tasks

Models of Computation

Approximate Agreement

Task Solvability
Alice’s 1/3-Agreement Protocol

\[
\begin{align*}
\text{mem}[A] & := 0 \\
\text{other} & := \text{mem}[B] \\
\text{if } \text{other} & = \bot \text{ then} \\
& \quad \text{decide } 0 \\
\text{else} & \\
& \quad \text{decide } 1/3
\end{align*}
\]
Alice’s 1/3-Agreement Protocol

```plaintext
mem[A] := 0
if mem[B] == 1
    decide 0
else
    decide 1/3

Alice writes her value to memory
```
Alice’s 1/3-Agreement Protocol

```
mem[A] := 0
if mem[B] == ⊥ then
decide 0
If she doesn’t see Bob’s value, decide her own.
decide 1/3
```
Alice’s 1/3-Agreement Protocol

```plaintext
mem[A] := 0
if mem[B] == ⊥ then
    decide 0
else
    decide 1/3
```

If she see’s Bob’s value, jump to the middle
Distributed Computing through Combinatorial Topology
One-Layer 1/3-Agreement Protocol

\[ I \]

\[ \delta \]

\[ P \]
No 1-Layer 1/5-Agreement Protocol

\[ \delta \]

\[ \delta \]

(no map possible)
2-Layer 1/5-Agreement

I

layer 1

layer 2

δ

Distributed Computing through Combinatorial Topology
Fact

In the layered read-write model,

The $1/K$-Agreement Task

Has a $\lceil \log_3 K \rceil$-layer protocol
Road Map

Elementary Graph Theory

Tasks

Models of Computation

Approximate Agreement

Task Solvability
Fact

The protocol graph for any $L$-layer protocol with input graph $\mathcal{I}$ is a subdivision of $\mathcal{I}$, where each edge is subdivided $3^L$ times.
Main Theorem

The two-process task \((\mathcal{I}, \mathcal{O}, \Delta)\) is solvable in the layered read-write model if and only if there exists a connected carrier map \(\Phi: \mathcal{I} \rightarrow 2^{\mathcal{O}}\) carried by \(\Delta\).
Corollary

The consensus task has no layered read-write protocol
Corollary

Any $\epsilon$–agreement task has a layered read-write protocol
This work is licensed under a Creative Commons Attribution-ShareAlike 2.5 License.

• **You are free:**
  – to **Share** — to copy, distribute and transmit the work
  – to **Remix** — to adapt the work

• **Under the following conditions:**
  – **Attribution.** You must attribute the work to “Distributed Computing through Combinatorial Topology” (but not in any way that suggests that the authors endorse you or your use of the work).
  – **Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

• For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to
  – [http://creativecommons.org/licenses/by-sa/3.0/](http://creativecommons.org/licenses/by-sa/3.0/).

• Any of the above conditions can be waived if you get permission from the copyright holder.

• Nothing in this license impairs or restricts the author's moral rights.