Artificial Chemistries:
Overview and Computing Aspects

Lidia Yamamoto
(KULeuven, Belgium)

This talk has been prerecorded!
watch the corresponding video on YouTube:
https://youtu.be/-JDFdIztGk0

presenting joint work with
Wolfgang Banzhaf
(Memorial University of Newfoundland, Canada)

Engineering and Control of Self-Organization, e-track CS-DC 2015
Contents

- Artificial Chemistries (ACs) in a Nutshell
  - Wet ("in vitro", "in vivo") vs. virtual ("in silico") ACs
  - Well-mixed vs. spatial ACs
  - Constructive vs. nonconstructive ACs
  - Simulation algorithms

- Computing with Artificial Chemistries
  - In silico, in vitro and in vivo chemical computing
  - Distributed computing and self-organization

- PyCellChemistry software package
  - www.artificial-chemistries.org

- Summary and Outlook

MIT Press, Summer 2015 (571 pages)
https://mitpress.mit.edu/books/artificial-chemistries
Artificial Chemistries (ACs)

- Man-made virtual or physical systems where objects are transformed in interactions, like molecules in chemical reactions
  
  \[ \text{abc} + d \rightarrow \text{abcd} \], \[ \text{0101100} \rightarrow \text{01} + \text{1100} \], \[ \text{40} + \text{10} \rightarrow \text{10} + \text{4} \]

- Spin-off of Artificial Life:
  - from “life as it could be” to “chemistry as it could be (imagined)?”

- Goals:
  - understand phenomena leading to the emergence of life
  - create new forms of synthetic life from the bottom up
    - “in vitro”, “in vivo”: “Wet” ACs in the laboratory
    - “in silico”: computational systems
      - high-level modelling and simulation of (real) chemistry and biology
      - chemistry as a metaphor for distributed and parallel computer algorithms
      - chemistry as a general model for interacting systems of objects: nuclear physics, language, music, economies
Wet ACs

- DNA computing
- Reaction-diffusion computers
- Synthetic life and protocells
- Computing with bacteria, slime mold, ...

Molecular Automaton
[Benenson2003][Shapiro2006]

Los Alamos Bug
[Rasmussen2003]

slime mold maze solver
[Adamatzky2010,2012]

self-propelled oil droplet
[Hanczyc2010]
Artificial Chemistries “in silico”

- Virtual, abstract ACs:
  - well-stirred: molecules as a “gas” or dissolved in well-mixed reactor
  - spatially-resolved: molecules move in 2D or 3D space
  - compartmentalized: molecules inside various (nested) containers

example of spatial AC: Organic Builder [Hutton2009]
Constructive vs. Nonconstructive ACs

- \( N \) = total number of possible molecular species
- \( M \) = number of species present in the reactor at a given moment
- Nonconstructive: \( M = N \) or close: fixed set of molecules
- Constructive: \( M \ll N \)
  - new molecules may be created, with potentially new interactions

example of nonconstructive AC: the Repressilator [Elowitz2000]

example of constructive AC: the Matrix Chemistry [Banzhaf1993]
Components of an Artificial Chemistry

- Triple (S,R,A)
  - S = set of molecules
  - R = set of reaction rules
  - A = algorithm that applies rules to molecules

- Some algorithms: (see book ch. 4 or [Yamamoto2013] for a survey)

<table>
<thead>
<tr>
<th>granularity</th>
<th>well mixed</th>
<th>spatial, compartmental</th>
</tr>
</thead>
<tbody>
<tr>
<td>individual molecules, single reactions</td>
<td>random molecular collisions: effective or elastic</td>
<td>move, collide, react (gas vs. fluid dynamics, lattice systems, crowding)</td>
</tr>
<tr>
<td>molecular species, effective reactions</td>
<td>reaction probability proportional to propensity (Gillespie SSA, next reaction method)</td>
<td>next subvolume method, multicompartment Gillespie</td>
</tr>
<tr>
<td>groups of molecules and reactions</td>
<td>fire groups of reactions together within interval tau (tau leaping)</td>
<td>spatial tau-leaping</td>
</tr>
<tr>
<td>concentration changes</td>
<td>numerical ODE integration</td>
<td>PDE integration</td>
</tr>
</tbody>
</table>
Contents

- Artificial Chemistries (ACs) in a Nutshell
  - Wet ("in vitro", "in vivo") vs. virtual ("in silico") ACs
  - Well-mixed vs. spatial ACs
  - Constructive vs. nonconstructive ACs
  - Simulation algorithms

- Computing with Artificial Chemistries
  - In silico, in vitro and in vivo chemical computing
  - Distributed computing and self-organization

- PyCellChemistry software package

- Summary and Outlook

cover image: molecular model of a ribosome [theasis,iStockphotoLP]
Molecular Machines and Turing Tapes

- Natural and synthetic information processing on multiple substrates

Turing Machine

Enzymatic Turing Machine [Bennett1985]

Artificial Molecular Machine [Laing1975]

mRNA translation in a ribosome [Wikipedia]
Computing with Artificial Chemistries

- Information processing occurs in nature, in a self-organizing way
  - How to harness these natural processes for our benefit: **engineer and control self-organization**

- In silico, in vitro and in vivo chemical computing
  - ACs for the modelling and simulation of wet biochemical computers
  - ACs as a metaphor for distributed and parallel computing

- Chemical computing inherently faces emergent phenomena
  - **Emergent computing** [Banzhaf1996][Forrest1990]
  - Properties at the higher levels emerge from interactions at the lower levels: molecular collisions, reactions, cell-to-cell communication...
  - Less intuitive for computer scientists: “think chemically”
    - **parallelism** at the microscopic scale (reactions in parallel)
    - **dynamical system** behavior at the macroscopic scale [Lones2014][Stepney2012]
    - program and data encoded in molecules
    - communication via molecules: cell signalling networks
    - compute by concentration changes, steer the flow of molecules
Molecular Automaton

- Implementation of a Finite State Machine using DNA and enzymes
- Proof-of-concept: simple FSM with 2 states and 2 possible input symbols

**Ingredients:**

- DNA “tape” with input symbols encoded as short segments
  
  \[ \langle s_0, b \rangle \]
  
  - DNA: `CTAGG` `TGCTG` `GCAGG` `ACCGA` `CCTAC` `CGTCC` `....`
  - Terminator

- FokI enzyme loaded with DNA fragments:
  - “cuts tape” to expose next state and symbol

\[ \text{s}_0 \xrightarrow{a} \text{s}_0 \]
\[ \text{s}_1 \xrightarrow{b} \text{s}_0 \]
Example operation of enzyme on DNA input “tape”

- DNA sticky end: current state and input symbol combination

\[
\begin{array}{c}
\langle s_0, b \rangle \\
\text{CAGG} & \text{TGGCT} & \text{GCAGG} \\
\text{ACCGA} & \text{CGTCC} & \text{....} \\
\end{array}
\]

- several enzyme-DNA complexes compete to bind to the exposed DNA sticky end
- only those complementary to the single-stranded sticky end can bind in a stable way
Example operation of enzyme on DNA input “tape”

“winning” enzyme-DNA complex binds to complementary DNA fragment representing input symbol

[Benenson2003] [Shapiro2006]
Example operation of enzyme on DNA input “tape”

enzyme cleaves DNA at “scissor” position
(transition to the next state starts)
Molecular Automaton

- Example operation of enzyme on DNA input “tape”

![Diagram of DNA tape and enzyme action](image)

- Cleaved portion is removed from input tape and discarded:
- Next input symbol is exposed (transition to the next state is complete)

[Benenson2003] [Shapiro2006]
Example operation of enzyme on DNA input “tape”
Example operation of enzyme on DNA input “tape”

Another enzyme-DNA complex binds to exposed input symbol, and cleaves the DNA input strand at the marked positions.
Example operation of enzyme on DNA input “tape”

Another enzyme-DNA complex binds to exposed input symbol, and cleaves the DNA input strand at the marked positions.
Molecular Automaton

- Example operation of enzyme on DNA input “tape”

the computation proceeds until a terminator symbol is found: the configuration of the terminator strand determines the output
Potential application: “DNA doctor in a cell” [Shapiro2006]
  • disease diagnosis: probabilistic operation due to competing biochemical pathways inside a cell, and fluctuating concentrations of molecules that trigger state transitions

Computer simulation using an Artificial Chemistry based on pattern matching and recombination [Tominaga2007]
Fraglets: An AC for Computer Networks

- Fraglets programming language [Tschudin2003]
  - fraglet = computation fragment
  - molecule = string of symbols = set of instructions and data
  - chemical reaction = bind to matching tag (head symbol), consume it, expose next symbol
  - constructive AC

- Automatic evolution of communication protocols for computer networks [Yamamoto2005]
Computing with Reaction-Diffusion

- Turing patterns [Turing1952]
  - Morphogens: chemicals that diffuse and react through tissue
  - Equilibrium instability leads to pattern formation: spots, stripes, waves
- Spatial AC, non-constructive
- Applications:
  - Reaction-diffusion computers [Adamatzky2005]
  - Models of distributed computation inspired by chemistry

- Evolution of decentralized cluster head election in sensor networks, on GPU hardware [Yamamoto2011]
- Voronoi diagrams [Adamatzky2005]
- BZ reaction for image processing [Kuhnert1989]
Contents

- Artificial Chemistries (ACs) in a Nutshell
  - Wet (“in vitro”, “in vivo”) vs. virtual (“in silico”) ACs
  - Well-mixed vs. spatial ACs
  - Constructive vs. nonconstructive ACs
  - Simulation algorithms

- Computing with Artificial Chemistries
  - In silico, in vitro and in vivo chemical computing
  - Distributed computing and self-organization

- PyCellChemistry software package
  - www.artificial-chemistries.org

- Summary and Outlook
An Artificial Chemistry in Python

- **PyCellChemistry**: Python package to let users program their own ACs
  - [www.artificial-chemistries.org](http://www.artificial-chemistries.org)

- Basic system:
  - multisets (bags) of molecules
  - chemical reactions
  - conversion from chemical reactions to ODE/PDE and Gillespie SSA
  - hierarchical cell compartments

- Example ACs:
  - basic: chameleons, prime number chemistry, matrix chemistry
  - biochemical circuits: dimerization, logistic growth, repressilator
  - ecology and evolution: Lotka-Volterra, quasispecies, NK landscapes
  - distributed & parallel computing: molecular TSP, fraglets, disperser
  - spatial ACs: reaction-diffusion
A Non-Constructive AC: Lotka-Volterra

class LotkaVolterra:
    def __init__( self, usestoch ):
        reactionstrs = [
            "rabbit + grass --> 2 rabbit + grass , k=1",
            "fox + rabbit   --> 2 fox            , k=1",
            "fox            -->                  , k=1"
        ]

        if usestoch:
            self.reactor = GillespieVessel(nav=40)
        else:
            self.reactor = WellStirredVessel()
        self.reactor.parse(reactionstrs)
        self.reactor.deposit('rabbit', 5.0)

    def run( self ):
        while (not self.extinct() and not self.exploded() and \
            self.reactor.vtime() <= 40.0):
            self.reactor.integrate(dt=0.001)
Lotka-Volterra: Deterministic vs. Stochastic

- Deterministic simulation via ODE integration:

![Graph showing Lotka-Volterra model with concentrations over time for grass, rabbit, and fox]
Lotka-Volterra: Deterministic vs. Stochastic

- Stochastic simulation via Gillespie SSA for $V = 40 / N_A$:
Traveling Salesman Problem (TSP):
- find the tour of minimum cost that visits all the cities on a map
- use only the available roads
- visit each city only once
- known to be NP-hard:
  - cannot be solved in general within a polynomial number of operations
  - typically heuristic algorithms are used: find approximate solutions
A Constructive AC: The Molecular TSP

- **Molecular TSP** [Banzhaf1990]: TSP heuristic inspired by chemistry
  - 2 types of molecules: **machines** and **tours**
    - tour: list of cities in the order they are visited, e.g. [1 2 5 4 6 3 1]
  - Machines (“enzymes”) operate on tours (“substrates”)
    - **E-machine**: swaps two random cities in a tour
    - **C-machine**: cuts a tour segment and pastes it elsewhere in tour
    - **I-machine**: cuts and inverts the segment before pasting it
    - **R-machine**: recombination (crossover) between 2 tours
  - Start with a “chemical soup” of random tours
  - Machines operate on tours independently (potentially in parallel)
    - draw 1 random molecule (2 for R-machine), perform operation
    - evaluate cost of each tour (educts and products)
    - inject best tour (2 best for R-machine) into soup, discard rest
  - Result: progressive selection of best tours
Molecular TSP: Initial Population

- Some random tours selected out of a population of 100 molecules:
Molecular TSP: After 4000 Generations

- Random tours selected out of the final population of 100 molecules:
class **MolecularTSP** ( HighOrderChem ):

    def __init__( self, ncities ): ...

    tsp = **TSPgraph**(ncities, ...)  # create road map;

    for i in range(popsize):  # produce random tours:
        mol = self.**randomMolecule**()  # each tour is a molecule
        self.mset.inject(mol)  # injected in reactor;

    rule = 'self.**exchangeMachine**(%s)'  # machines are reaction
    self.rset.inject(rule, count)  # rules in same reactor

    rule = 'self.**cutMachine**(%s)'
    self.rset.inject(rule, count)

...

    def run( self ): ...

    for gen in range(self.maxgen):
        for j in range(genops):
            self.**iterate**()  # pick rules and tours for reaction
            (bfit, bmol) = self.**bestMolecule**()  # best of generation
Spatial ACs: Reaction Diffusion Demos

- **Gray-Scott** [Gray1990][Pearson1993]
  - V: activator (autocatalyst)
  - U: substrate
  - $U + 2V \xrightarrow{k_1} 3V$
  - $V \xrightarrow{k_2} \emptyset$
  - $\emptyset \xrightarrow{F} U$
  - $U \xrightarrow{F} \emptyset$
  - $\frac{\partial u}{\partial t} = -uv^2 - Fu + F + Du \nabla^2 u$
  - $\frac{\partial v}{\partial t} = uv^2 - (F + K)v + D_v \nabla^2 v$

Other demos:
- Activator-Inhibitor [Koch1994]
- Activator-Substrate [Meinhardt1982]
- Dichotomous branching [Meinhardt1982]
The Gray-Scott Demo

- Some example of patterns:
from ReactionDiffusion import *

class GrayScottDemo():
    def __init__( self ) :
        reactionstrs = [ "U + 2 V --> 3 V",
                        " V -->    ",
                        "  --> U ",
                        "  U -->    "  ]
        self.rsys = ReactionDiffusionSystem(sizex, sizey, dx)
        self.rsys.parse(reactionstrs)
        self.rsys.set_coefficient(1, F+K)  ....  # kinetic coefs.
        self.rsys.set_diffcoef('U', DU)    ....  # diffusion coefs.
        self.rsys.deposit('V', initconc, posx, posy)  # initial cond.
    def run( self, finalvt=2000.0, dt=0.1 ) :
        while (self.rsys.vtime() <= finalvt):
            self.rsys.integrate(dt)    # numerical PDE integration
            self.rsys.animate(...)    # animation in VPython
Summary and Outlook

- Brief overview of artificial chemistries with a few examples
  - Focus on computing applications
  - Many more ACs exist (our book contains almost 1000 citations)
- What can we learn from ACs? Are they just toy chemistries?
  - Engineering approach: learn how things work by building them: build complexity starting from the bottom up
    - PyCellChemistry as a software tool to facilitate learning, practice and experimentation with various ACs
  - Natural computing and emergent computation: computation is embedded in the chemical system
    - ACs make such tight association more clear
  - Understand emergent phenomena through mathematical analysis:
    - formalizing ACs: Chemical Organization Theory, RAF theory (reflexively autocatalytic sets), Chemical Reaction Automata (DNA computing), P systems, Brane calculi, ...
Summary and Outlook

Towards a discipline of AC: challenges

- AC field not mature yet:
  - borders still not clearly delimited, no coherent big picture
- Barely scratching the surface of
  - commonalities among emergent phenomena (shared challenge with complex systems research)
  - computing with self-organization and emergence
  - how to move upwards in complexity, encapsulating the acquired emergent properties

Future:

- Tigher interdisciplinarity and integration between wet and virtual
- Fuzzy line between virtual and real, more and more hybrid systems
- Seamless programming: compile chemistry? chemical computers?
References

References

References