Week 14: Whole-cell models; Digital evolution

- Genome-scale metabolic models
  - Reconstruction
  - Flux balance analysis
- **Artificial life**

Tracking the evolution of complex features

Difficult to provide a complete account of the origin of any complex feature:

- Extinction of the intermediate forms.
- Imperfection of the fossil record.
- Incomplete knowledge of the genetic & developmental mechanisms that produce such features.

Solution: Experimental evolution!

- *E. coli* long-term evolution experiment: Tracking genetic changes in 12 initially identical populations of asexual *Escherichia coli* bacteria since 24 February 1988.

  1. Every day, the cultures are propagated.
  2. Every 75 days (500 generations), mixed-population samples are frozen away.
  3. Estimate mean fitness (relative to the ancestor) using the mixed-pop samples.

- 50,000 generations in Feb 2010; 66,000 in November 2016; 10,000th transfer of the experiment on March 13, 2017.

Richard Lenski

myxo.css.msu.edu/ecoli/index.html

Tracking the evolution of complex features

Difficult to provide a complete account of the origin of any complex feature:

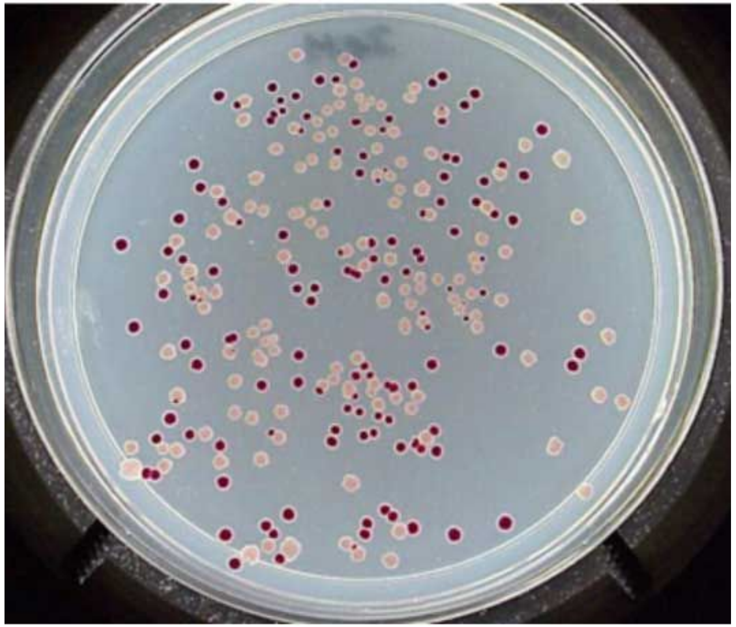
- Extinction of the intermediate forms.
- Imperfection of the fossil record.
- Incomplete knowledge of the genetic & developmental mechanisms that produce such features.

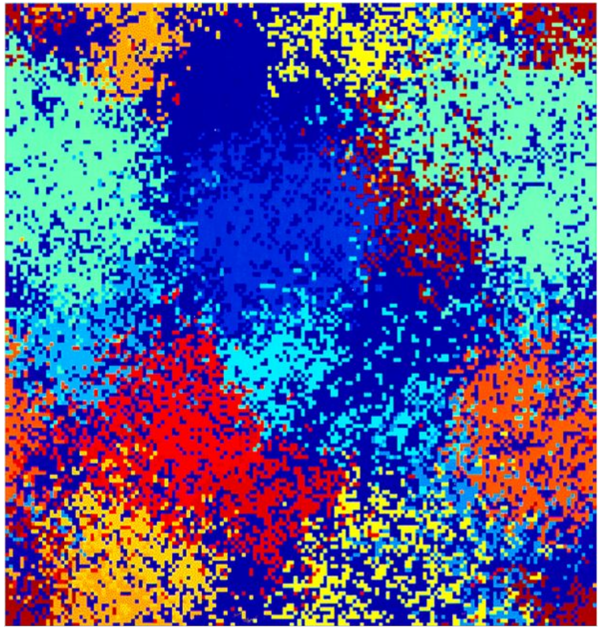
Solution: <u>Digital</u> evolution!

1. Create initial population of individuals with random ‘genotypes’
2. Figure out a way of evaluating ‘phenotype’ and estimate a fitness values for each individual.
3. Select a subset of individuals using fitness as a key criterion
4. Vary their genotypes by making random changes or recombining portions
5. Repeat from step 2 until a solution that is sufficiently good.

Track all genotypic and phenotypic changes during the evolution of a complex trait with enough replication to obtain statistically powerful results.

Tracking the evolution of complex features



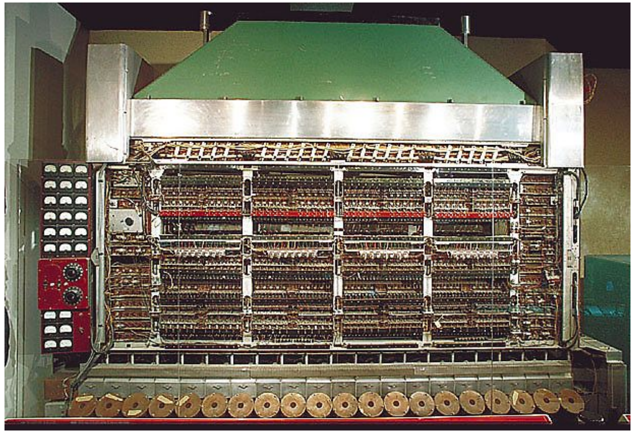


beacon-center.org/wp-content/uploads/2010/11/BEACON_Petri_Dish.jpg
beacon-center.org/wp-content/uploads/2010/11/BEACON_avida-grid.png

Artificial life

The IAS (von Neumann) machine built in 1951

- Binary computer; Memory: 1024 words (5.1 KB / 4 working).
- Addition time: 62 microsec; multiplication time: 713 microsec.



1953: Nils Barricelli, theoretical biologist.

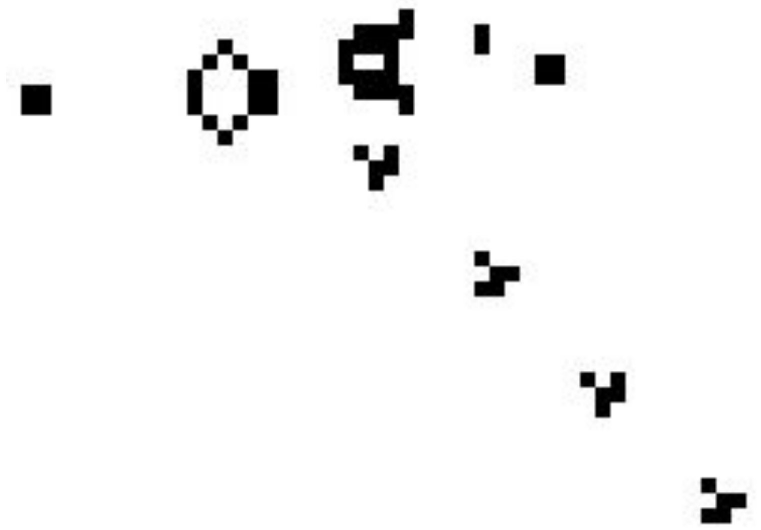
- Simulated the evolution of populations of **artificial organisms**.
  - Each organism: genome consisting of a string of numbers.
  - Random mutations and sexual exchange of genes caused populations to evolve.

- Speciation, parasitism, and predation arose spontaneously.
- Punctuated equilibrium, the tendency of dominant species to remain static for many generations and then suddenly give way to new dominant species of a different character.

Nils Aall Barricelli

nautil.us/issue/14/mutation/meet-the-father-of-digital-life
americanhistory.si.edu/collections/search/object/nmah_334741
Wikipedia

Conway's Game of Life



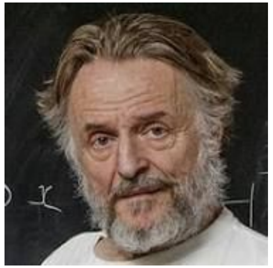
Zero-player game.

Universe: an infinite, two-dimensional orthogonal grid of square cells.

Each cell is in one of two possible states, alive or dead, (or populated and unpopulated, respectively).

Every cell interacts with its eight neighbours.

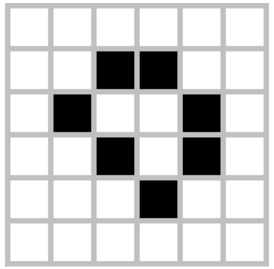
1970

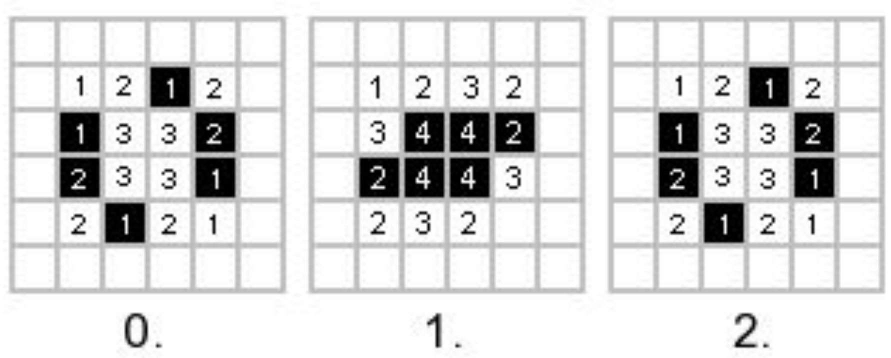


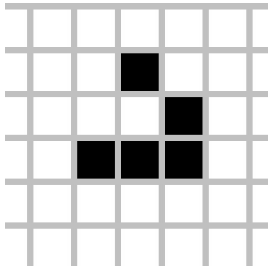
At each step in time, the following transitions occur:

1. Any live cell with < 2 live neighbours dies, as if by underpopulation.
2. Any live cell with 2 or 3 live neighbours lives on to the next generation.
3. Any live cell with > 3 live neighbours dies, as if by overpopulation.
4. Any dead cell with exactly 3 live neighbours becomes a live cell, as if by reproduction.

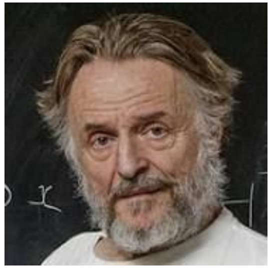
Conway's Game of Life







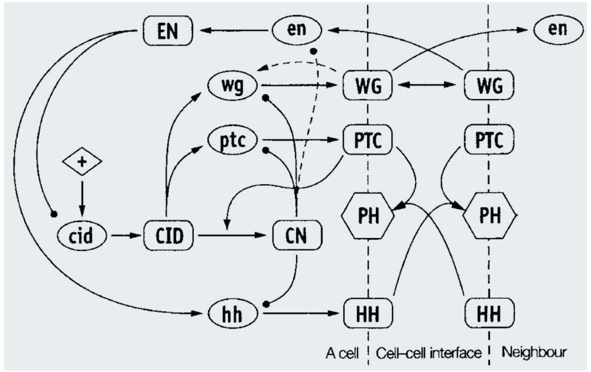
1970



At each step in time, the following transitions occur:

1. Any live cell with < 2 live neighbours dies, as if by underpopulation.
2. Any live cell with 2 or 3 live neighbours lives on to the next generation.
3. Any live cell with > 3 live neighbours dies, as if by overpopulation.
4. Any dead cell with exactly 3 live neighbours becomes a live cell, as if by reproduction.

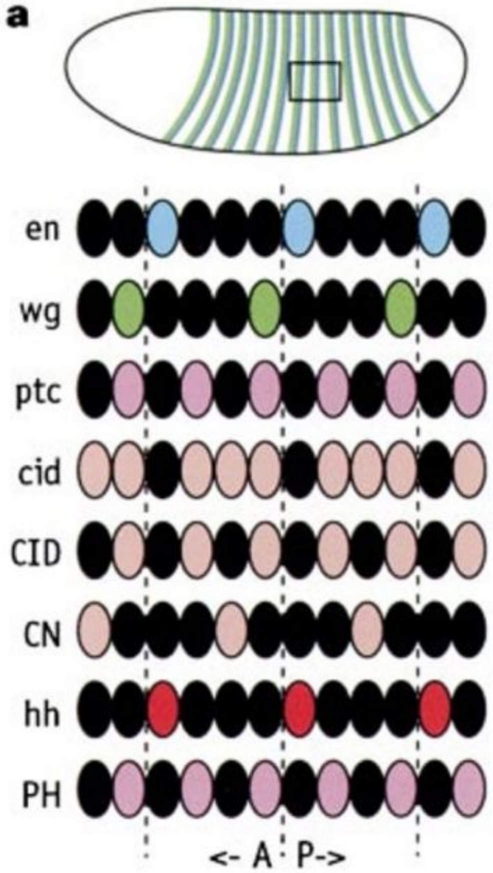
Modeling segment polarity

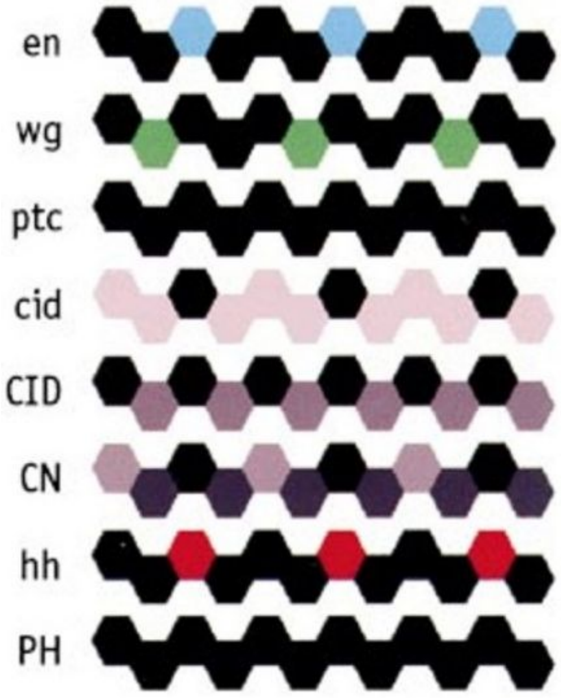


$$\frac{d[hh]_i}{dt} = T_{\max}\rho_{hh}\left[\frac{[EN]_i^{\nu_{ENhh}}}{K_{ENhh}^{\nu_{ENhh}} + [EN]_i^{\nu_{ENhh}}}\right] - \frac{[hh]_i}{H_{hh}}$$

$$\frac{d[HH]_{i,j}}{dt} = \frac{P_{\max}\sigma_{HH}[hh]_i}{6} - \frac{[HH]_{i,j}}{H_{HH}} - k_{PTCHH}[HH]_{i,j}[PTC]_{n,j+3}$$

$$\frac{d[PH]_{i,j}}{dt} = k_{PTCHH}[HH]_{n,j+3}[PTC]_{i,j} - \frac{[PH]_{i,j}}{H_{PH}}$$



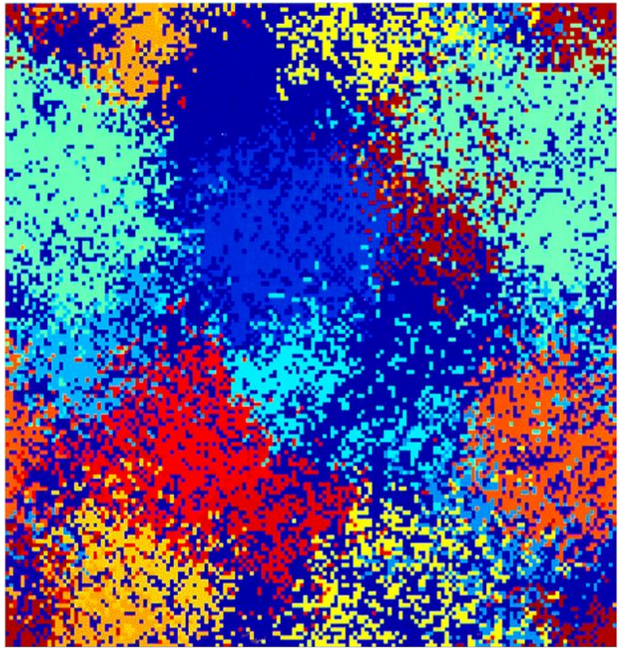


Von Dassow (2000) Nat.

AVIDA

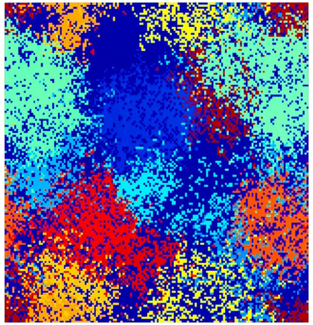
Software platform for experiments with self-replicating and evolving computer programs.

- Detailed control over experimental settings and protocols
- Array of measurement tools + methods to analyze & post-process experimental data.



discovermagazine.com/2005/feb/cover
avida.devosoft.org

AVIDA



Each digital organism:

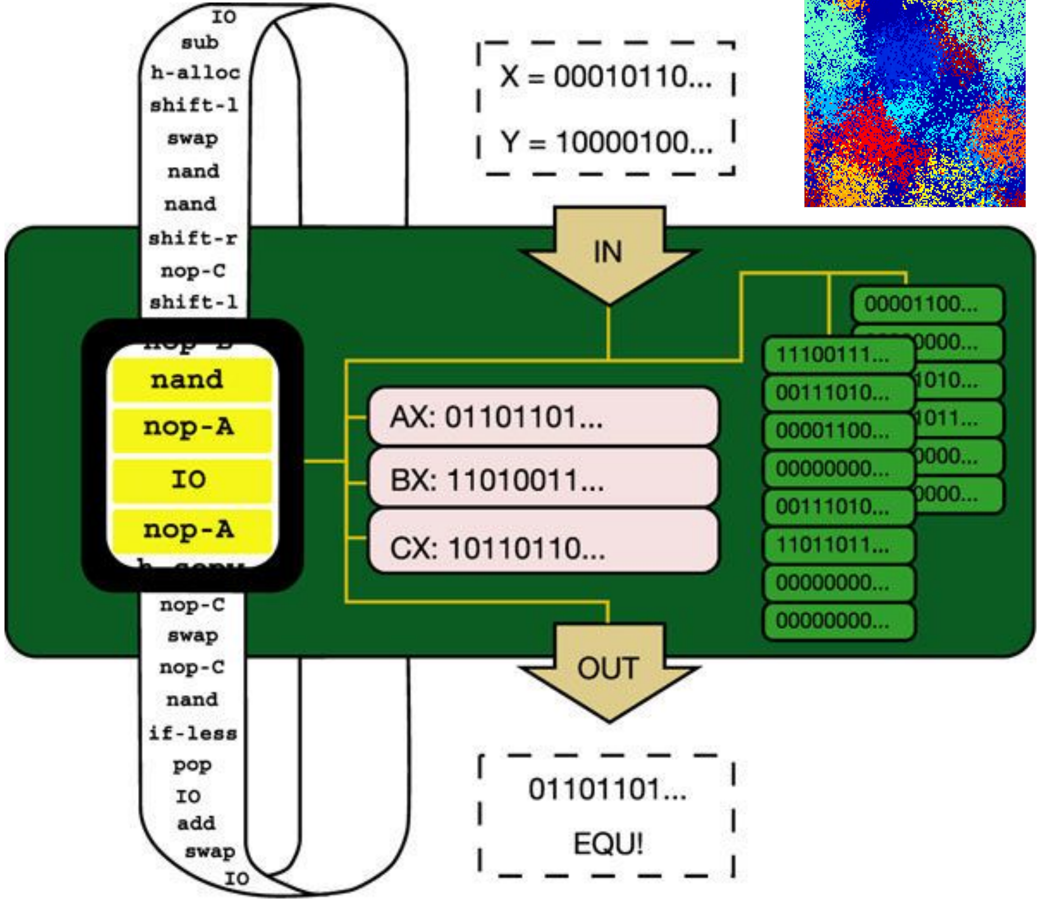
- Is a self-contained computing automaton w/ the ability to construct new automata.
- Is responsible for building the genome (computer program) that will control its offspring automaton & handing that genome to its (Avida) environment.
  - Avida will then construct virtual hardware to *run* the genome & determine how this new organism should be placed into the population.
- Competes for energy with other organisms and can obtain energy by performing logic functions.

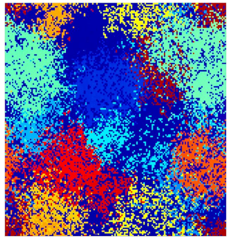
In a typical Avida experiment:

- A successful organism attempts to make an identical copy of its own genome.
- Avida randomly places that copy into the population (replacing another member).

discovermagazine.com/2005/feb/cover
avida.devosoft.org

Digital organism





Each individual organism has:

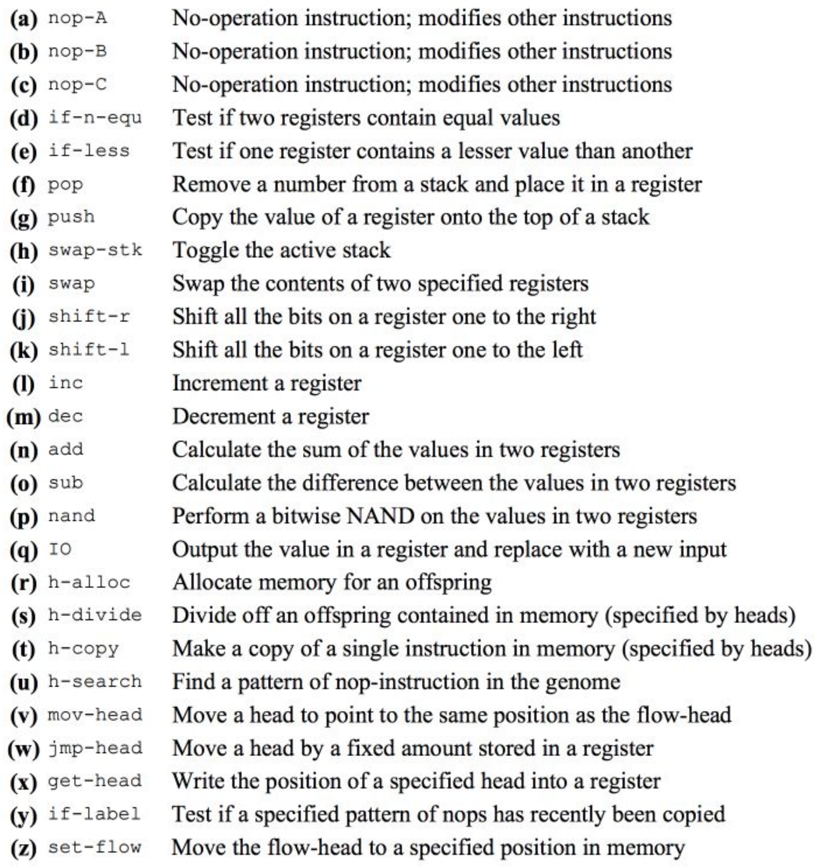
- A circular genome: sequence of instructions.
- A virtual CPU: two stacks & three registers that hold 32-bit strings.

Execution of the genomic program generates a computational metabolism:

- Numerical substrates input from env.
- Processed in stacks and registers.
- Resulting products output to env.

Lenski (2003) Nature

Instruction set & Initial conditions



(a)	`nop-A`	No-operation instruction; modifies other instructions
(b)	`nop-B`	No-operation instruction; modifies other instructions
(c)	`nop-C`	No-operation instruction; modifies other instructions
(d)	`if-n-equ`	Test if two registers contain equal values
(e)	`if-less`	Test if one register contains a lesser value than another
(f)	`pop`	Remove a number from a stack and place it in a register
(g)	`push`	Copy the value of a register onto the top of a stack
(h)	`swap-stk`	Toggle the active stack
(i)	`swap`	Swap the contents of two specified registers
(j)	`shift-r`	Shift all the bits on a register one to the right
(k)	`shift-l`	Shift all the bits on a register one to the left
(l)	`inc`	Increment a register
(m)	`dec`	Decrement a register
(n)	`add`	Calculate the sum of the values in two registers
(o)	`sub`	Calculate the difference between the values in two registers
(p)	`nand`	Perform a bitwise NAND on the values in two registers
(q)	`IO`	Output the value in a register and replace with a new input
(r)	`h-alloc`	Allocate memory for an offspring
(s)	`h-divide`	Divide off an offspring contained in memory (specified by heads)
(t)	`h-copy`	Make a copy of a single instruction in memory (specified by heads)
(u)	`h-search`	Find a pattern of nop-instruction in the genome
(v)	`mov-head`	Move a head to point to the same position as the flow-head
(w)	`jmp-head`	Move a head by a fixed amount stored in a register
(x)	`get-head`	Write the position of a specified head into a register
(y)	`if-label`	Test if a specified pattern of nops has recently been copied
(z)	`set-flow`	Move the flow-head to a specified position in memory

Each position in the genome sequence of a digital organism is one of 26 possible instructions.

At the beginning of the experiment:

- One ancestor that can replicate but could not perform any logic functions.
- All organisms identical & obtain equal energy to execute their genomic programs.

Lenski (2003) Nature

Replication

Reproduction is asexual and occurs by binary fission.

Copy commands → genome replicates itself one instruction at a time.

- Subject to errors:
  - point mutations, insertions & deletions.
- Each mutation may change an organism's phenotype:
  - replication efficiency,
  - computational metabolism, &
  - robustness.

Selection

Genotypes vary in their expected reproductive success.

As in nature, selection in Avida depends on:

- phenotypic effects of a mutation...
- in its genetic context and
- in relation to the organism's environment.

Most mutations are deleterious or neutral, but a small fraction increases fitness.

Lenski (2003) Nature

Competition for resources

Digital organisms compete for the energy needed to execute instructions.

Energy: discrete quanta called ‘single-instruction processing’ units, or SIPs.

- Each SIP suffices to execute one instruction.

Execute instructions → Express phenotypes → Obtain more energy & copy its genome.

Competition for resources

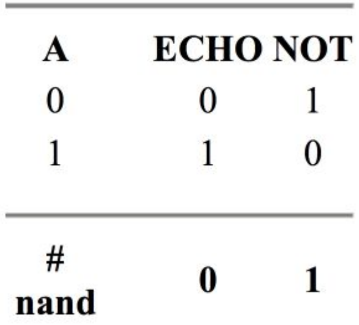
Two mechanisms to acquire energy:

1. Each organism receives SIPs in proportion to its genome length.
2. An organism can obtain additional SIPs by performing logic operations on 32-bit strings.
    a. Only one of the 26 instructions in the genetic code, nand (‘not and’), is itself a logic operator
        i. nand must be executed in coordination with IO (‘input–output’) instructions to perform the NAND function.
    b. All other logic functions can be constructed using one or more nand instructions within an integrated framework of other instructions.

Lenski (2003) Nature

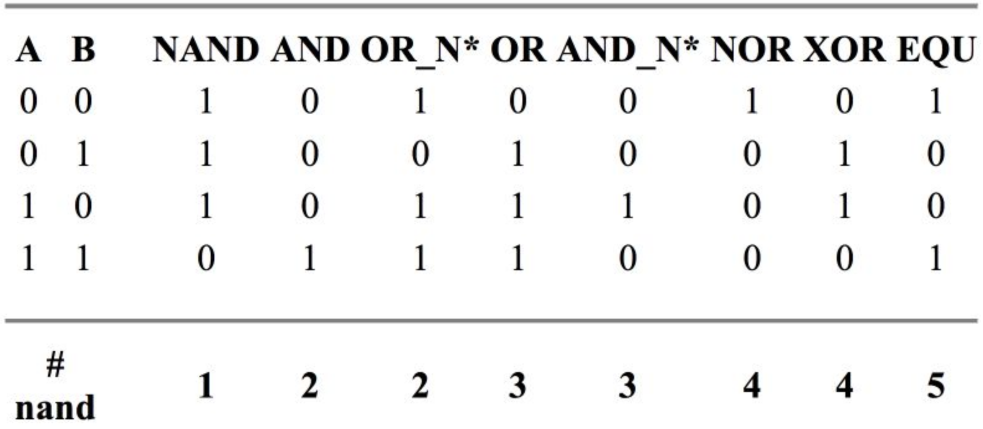
Logic operations

One-input Output



A	ECHO	NOT
0	0	1
1	1	0
# nand	**0**	**1**

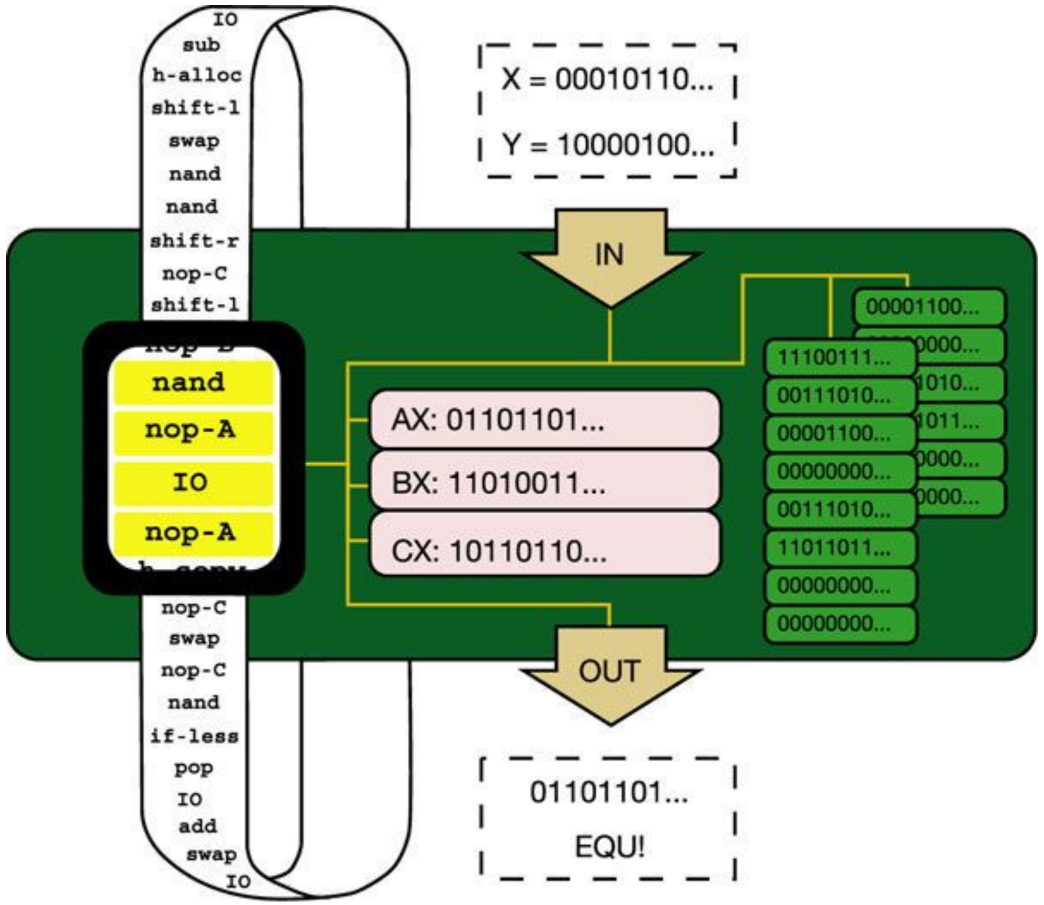
Two-input Output



A	B	NAND	AND	OR_N*	OR	AND_N*	NOR	XOR	EQU
0	0	1	0	1	0	0	1	0	1
0	1	1	0	0	1	0	0	1	0
1	0	1	0	1	1	1	0	1	0
1	1	0	1	1	1	0	0	0	1
# nand		**1**	**2**	**2**	**3**	**3**	**4**	**4**	**5**

Lenski (2003) Nature

Executing instructions



Example instruction

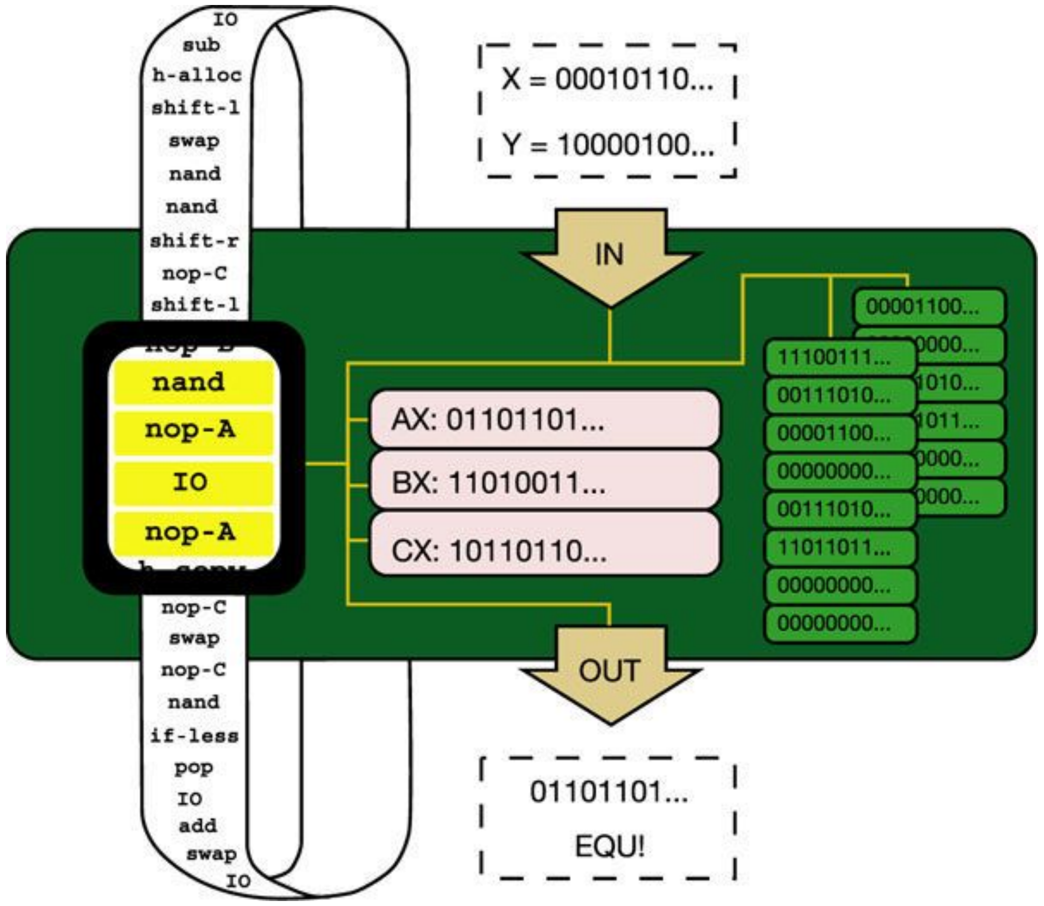
nand, immediately followed by the modifying **nop-A** (‘no operation’):

1. Combine bit-strings in the BX and CX registers according to the nand operator
2. Write the result to the AX register.

The subsequent IO instruction and its modifying nop-A cause the string in the AX register to be **output**.

Lenski (2003) Nature

Executing instructions



Output evaluation

Match it with the correct answer for a reward function.

Perfect match → organism's rate of energy acquisition, and hence the execution of its genomic program, is accelerated by the factor.

In this case, EQU!

(A and B) or (~A and ~B)

No reward if any of the 32 bits in the string were incorrect, or if it had already been rewarded in its lifetime for performing EQU.

Lenski (2003) Nature

Logic operations

Rewards for performing logic functions

Function name	Logic operation	Computational merit
NOT	~A; ~B	2
NAND	~(A and B)	2
AND	A and B	4
OR_N	(A or ~B); (~A or B)	4
OR	A or B	8
AND_N	(A and ~B); (~A and B)	8
NOR	~A and ~B	16
XOR	(A and ~B) or (~A and B)	16
EQU	(A and B) or (~A and ~B)	32

- EQU has largest reward. It is more complex than any other one- or two-input logic function, given the available genomic instructions.
- Exhaustive search shows that the minimum number of nand operations to perform EQU is five > for any other one- or two-input logic function.

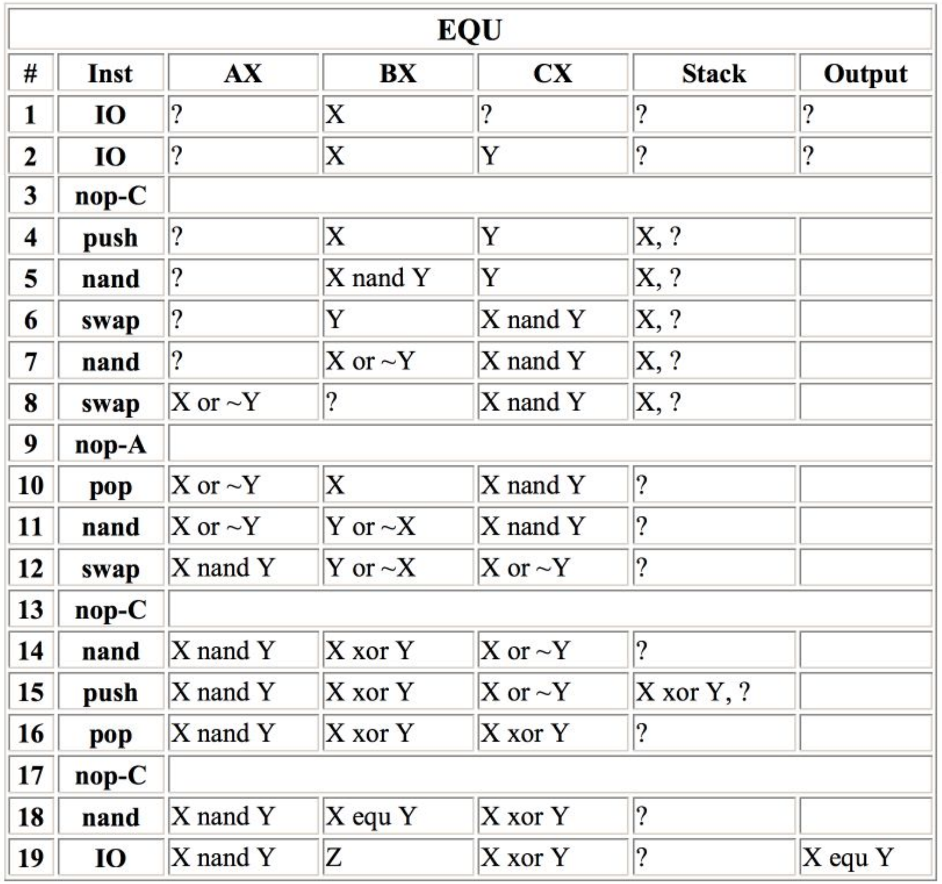
The symbol '~' denotes negation.

The reward for computational merit increases with 2^n, where *n* is the minimum number of nand operations needed to perform the listed function.

No added benefit is obtained for performing any function multiple times.

Lenski (2003) Nature

Shortest hand-written EQU program



EQU

#	Inst	AX	BX	CX	Stack	Output
1	IO	?	X	?	?	?
2	IO	?	X	Y	?	?
3	nop-C					
4	push	?	X	Y	X, ?	
5	nand	?	X nand Y	Y	X, ?	
6	swap	?	Y	X nand Y	X, ?	
7	nand	?	X or ~Y	X nand Y	X, ?	
8	swap	X or ~Y	?	X nand Y	X, ?	
9	nop-A					
10	pop	X or ~Y	X	X nand Y	?	
11	nand	X or ~Y	Y or ~X	X nand Y	?	
12	swap	X nand Y	Y or ~X	X or ~Y	?	
13	nop-C					
14	nand	X nand Y	X xor Y	X or ~Y	?	
15	push	X nand Y	X xor Y	X or ~Y	X xor Y, ?	
16	pop	X nand Y	X xor Y	X xor Y	?	
17	nop-C					
18	nand	X nand Y	X equ Y	X xor Y	?	
19	IO	X nand Y	Z	X xor Y	?	X equ Y

Program of length 19 that performs EQU but does not replicate.

Potentially the shortest one to perform EQU.

Lenski (2003) Nature

Evolving the capability to perform EQU

Ancestor could replicate but could not perform any logic function.

An organism that evolved one or more of nine logic functions would obtain further energy.

The benefit increased exponentially with the approximate difficulty of each function.

- Functions could be performed in any order during an individual's life
- No extra energy was obtained by repeatedly performing the same function.

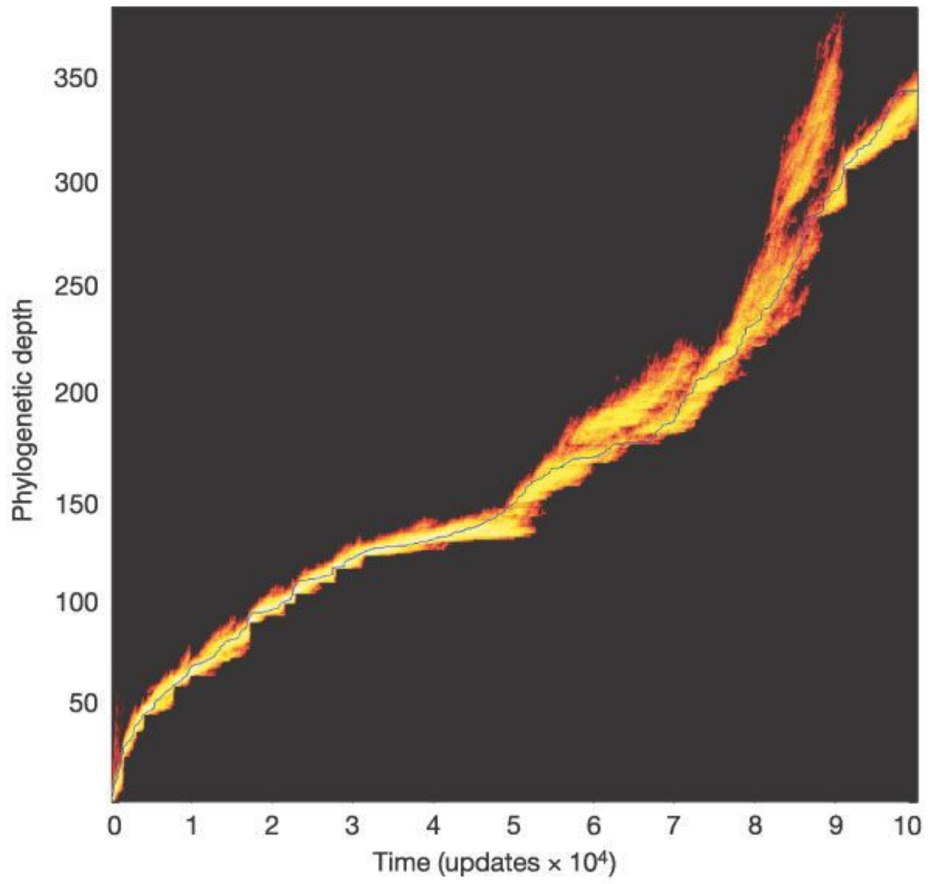
No single mutation in the ancestor can produce even the simplest of these functions.

- Several mutant instructions must appear in the same lineage, and such that they are coordinately. executed, to perform even a simple function.

Many populations evolved the capacity to perform EQU, the most complex of these functions!

Lenski (2003) Nature

Evolving the capability to perform EQU



Trajectory of population divergence from ancestor.

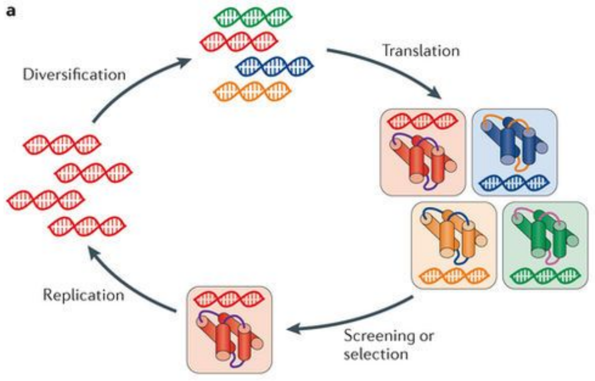
- Phylogenetic depth: cumulative number of generations in which an organism's genotype differs from its parent.
- Colours indicate the relative abundance of genotypes; yellow more abundant than red.
- Blue line: Line of descent leading to the most abundant final genotype.

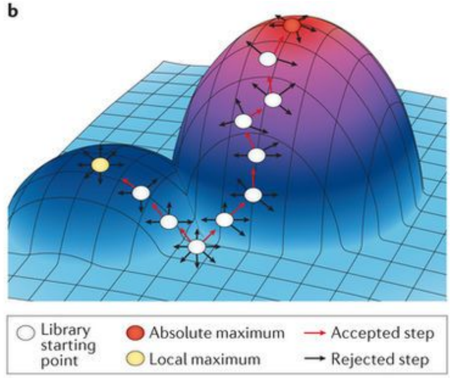
Final dominant type:

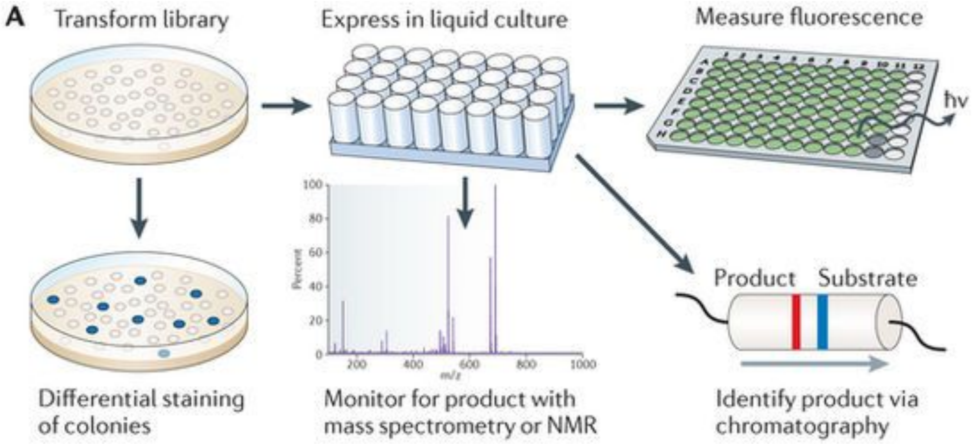
- 344 steps removed from its ancestor
- Genome is 83 instructions long (the ancestral length was 50)
- Performs all 9 logic funcs that provide energy.

Lenski (2003) Nature

Directed evolution






A
Transform library
Express in liquid culture
Measure fluorescence
hv
Differential staining of colonies
Monitor for product with mass spectrometry or NMR
Product
Substrate
Identify product via chromatography