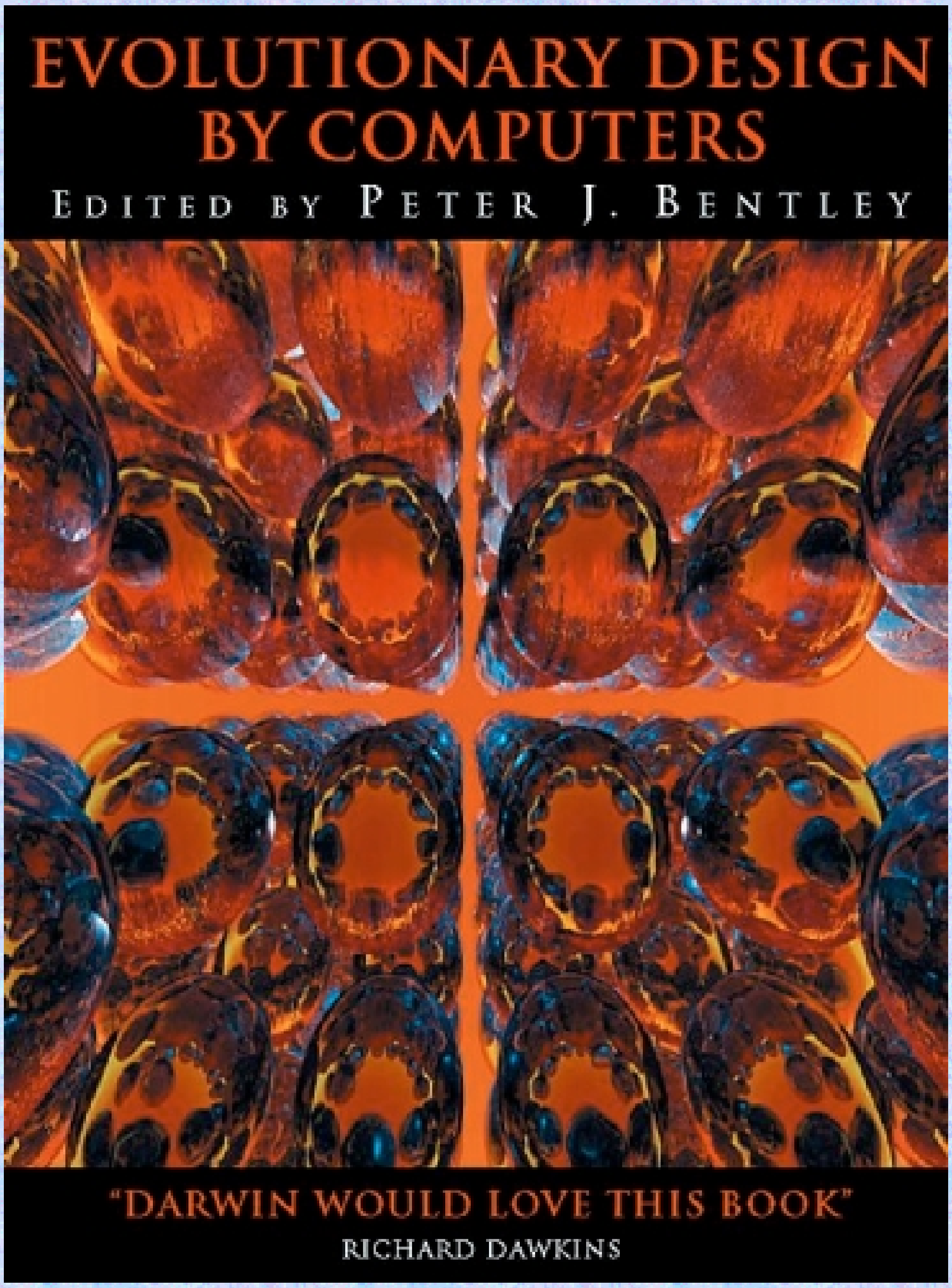
Evolutionary Design



EVOLUTIONARY DESIGN BY COMPUTERS

EDITED BY PETER J. BENTLEY

"DARWIN WOULD LOVE THIS BOOK"

RICHARD DAWKINS

Review

4 main types of Evolutionary Algorithms

- Genetic Algorithm - John Holland
- Genetic Programming - John Koza
- Evolutionary Programming - Lawrence Fogel
- Evolutionary Strategies - Ingo Rechenberg

Evolutionary Art

Computer Evolution of Buildable Objects

Evolutionary Art

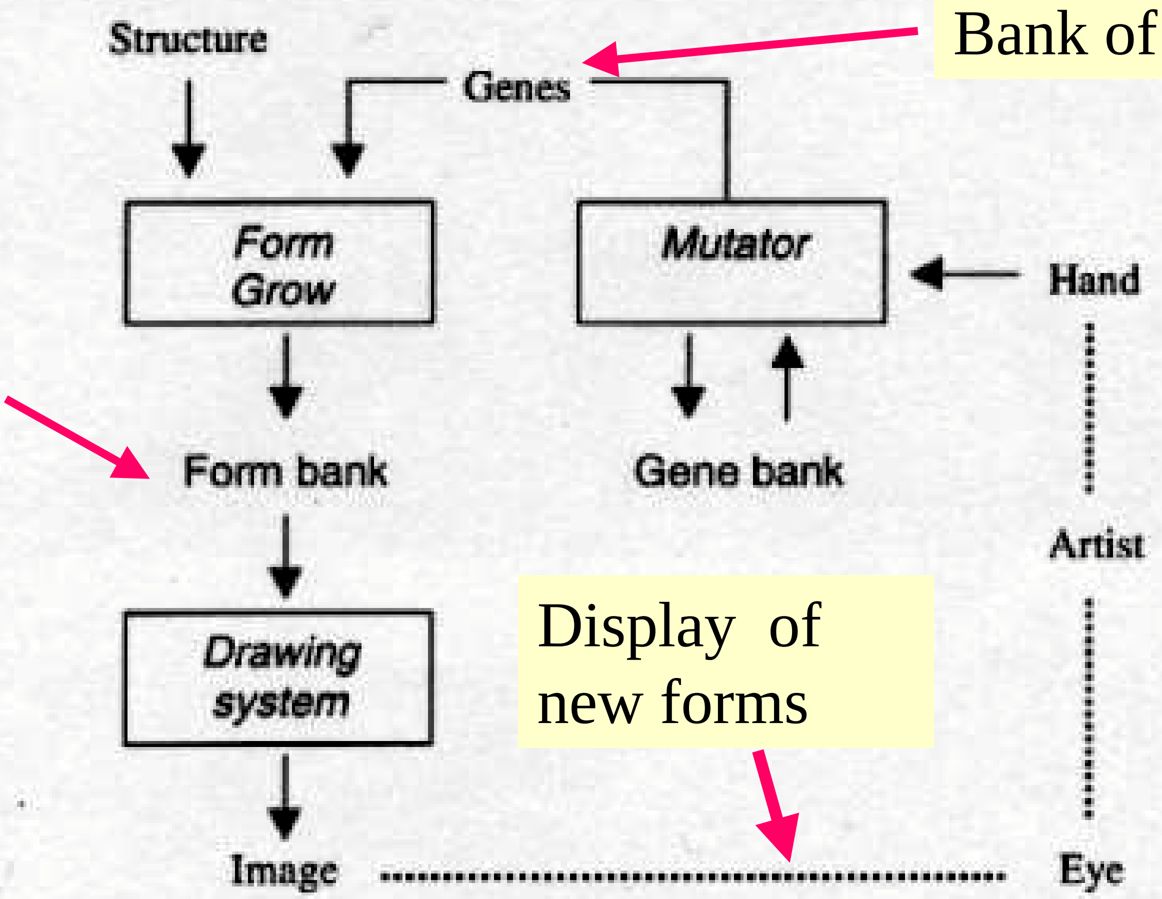
Stephen Todd and William Latham

built artistic system called "Mutator"

- Computer program based on mutation and natural selection to help an artist explore the world of three dimensional art forms.
- Produces horns, pumpkins, shells, mathematical shapes and many other shapes

Mutator

Assists the artist to search for interesting forms and bank the results



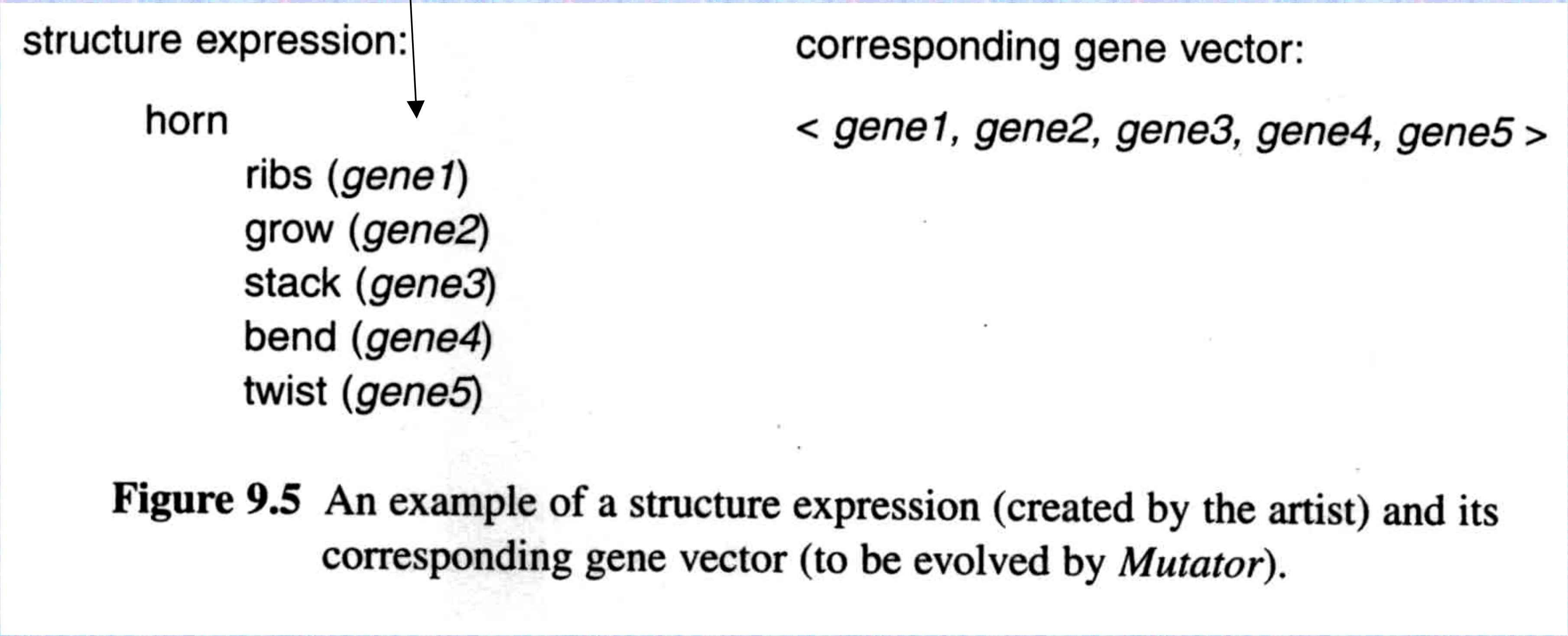
Bank of genes

Bank of forms of genes

Display of new forms

Mutator

Genotype --> phenotype



structure expression:

horn
    ribs (*gene1*)
    grow (*gene2*)
    stack (*gene3*)
    bend (*gene4*)
    twist (*gene5*)

corresponding gene vector:

< *gene1*, *gene2*, *gene3*, *gene4*, *gene5* >

Figure 9.5 An example of a structure expression (created by the artist) and its corresponding gene vector (to be evolved by *Mutator*).

Mutator

parent

9 children

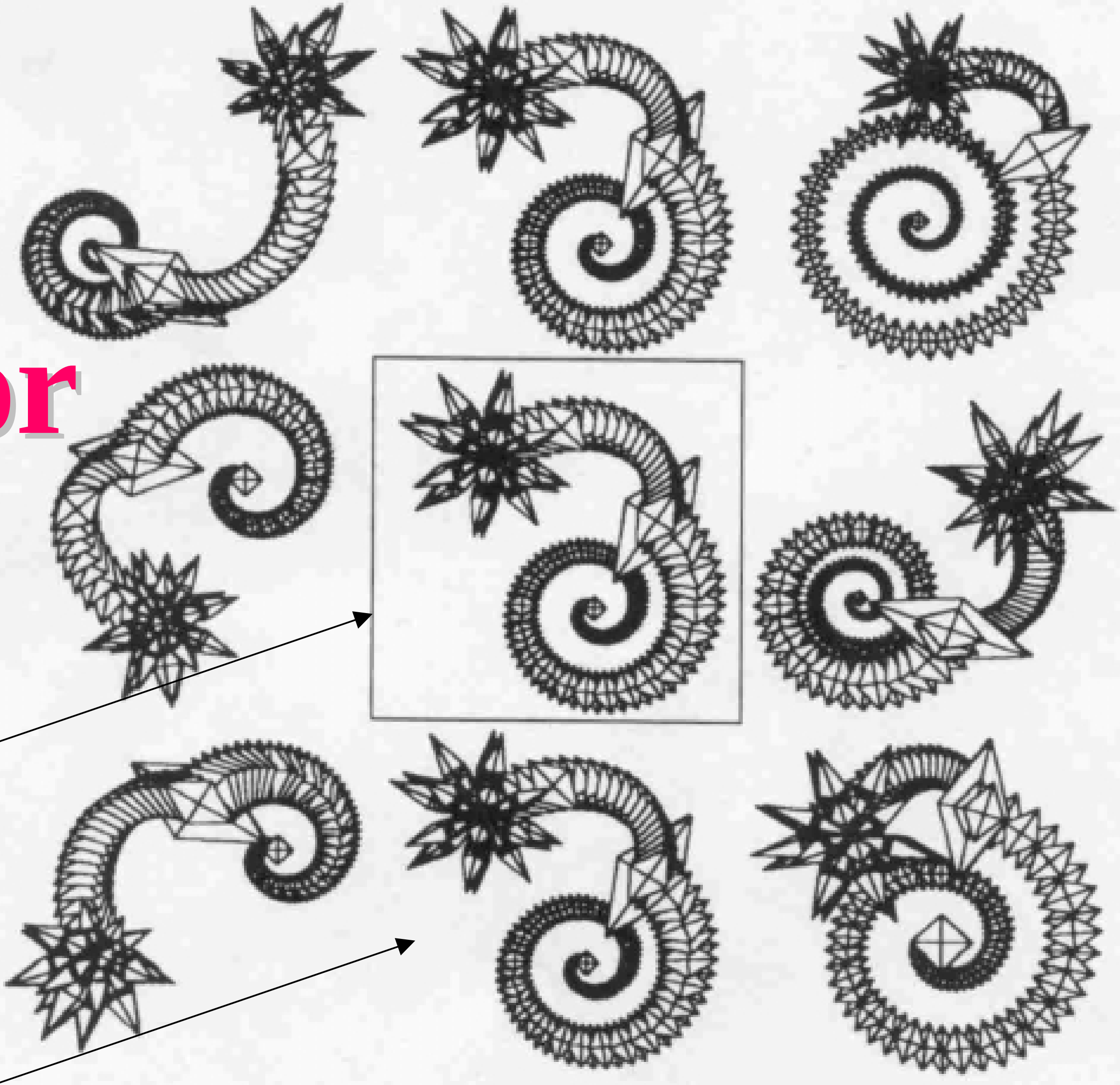
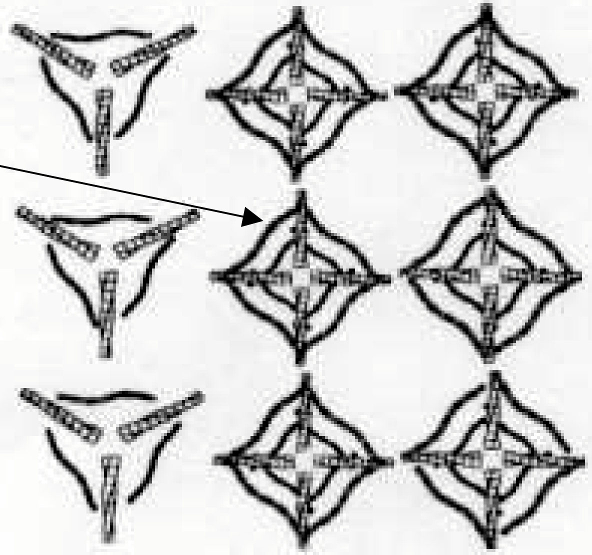


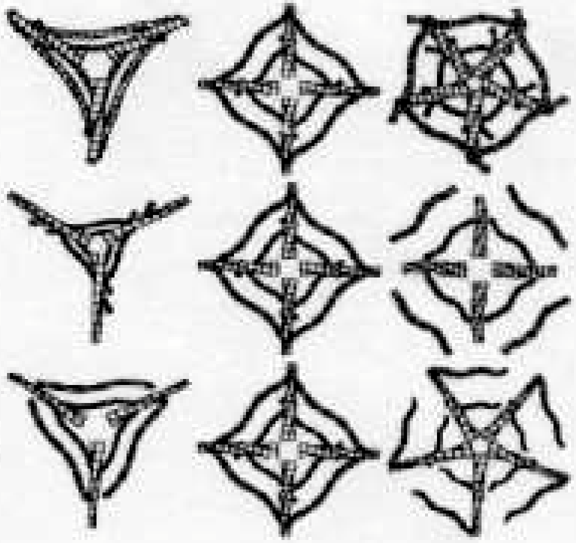
Figure 9.6 A frame of nine mutations. The parent is in the centre surrounded by offspring.

Mutator

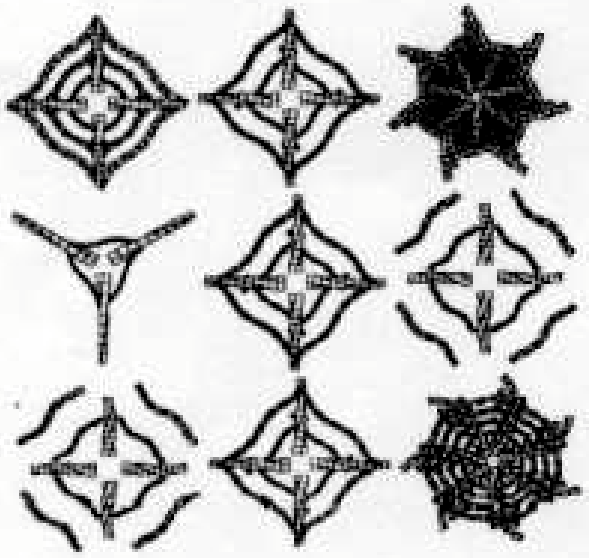
Parent of 9 offsprings



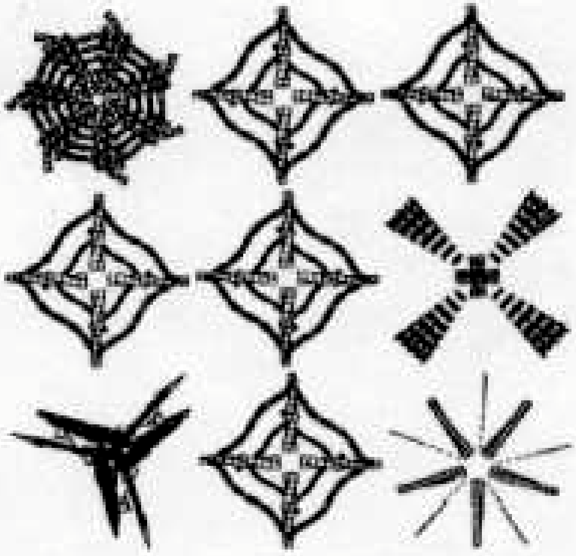
A
Very low mutation rate



B
Low mutation rate



C
Medium mutation rate

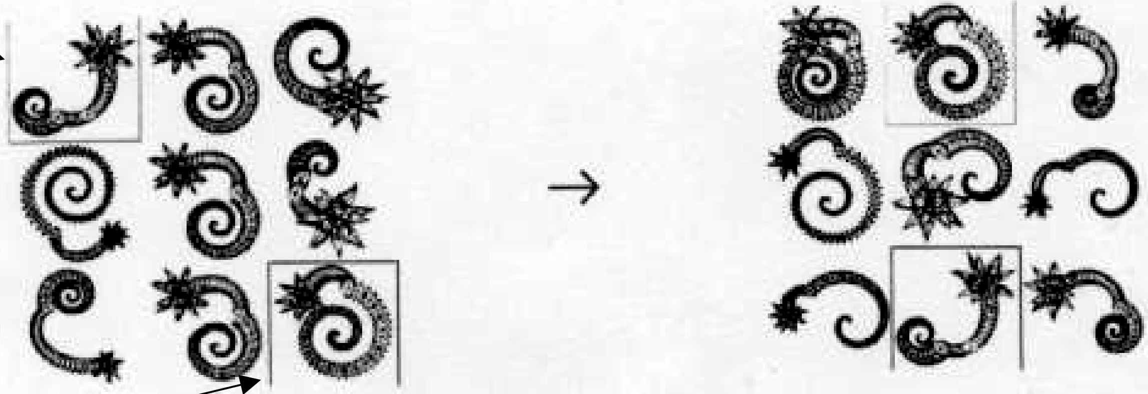


D
High mutation rate

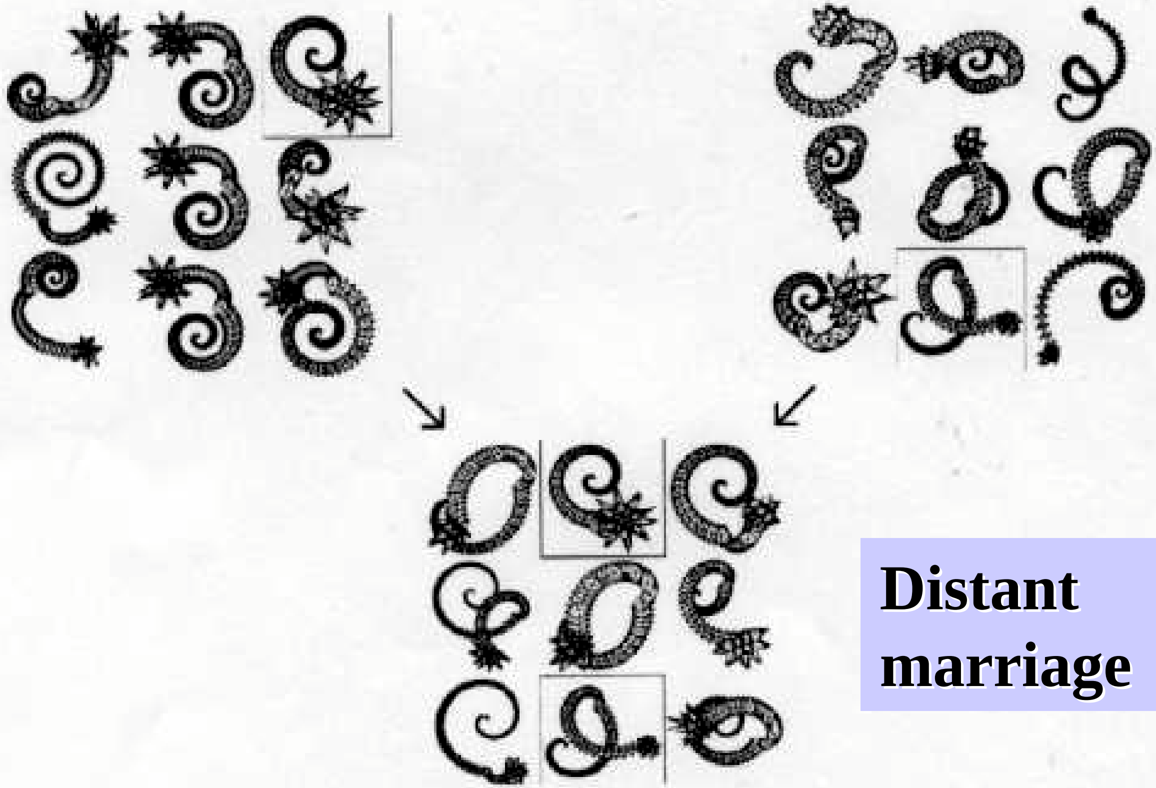
Mutator

Parent 1

Parent 2



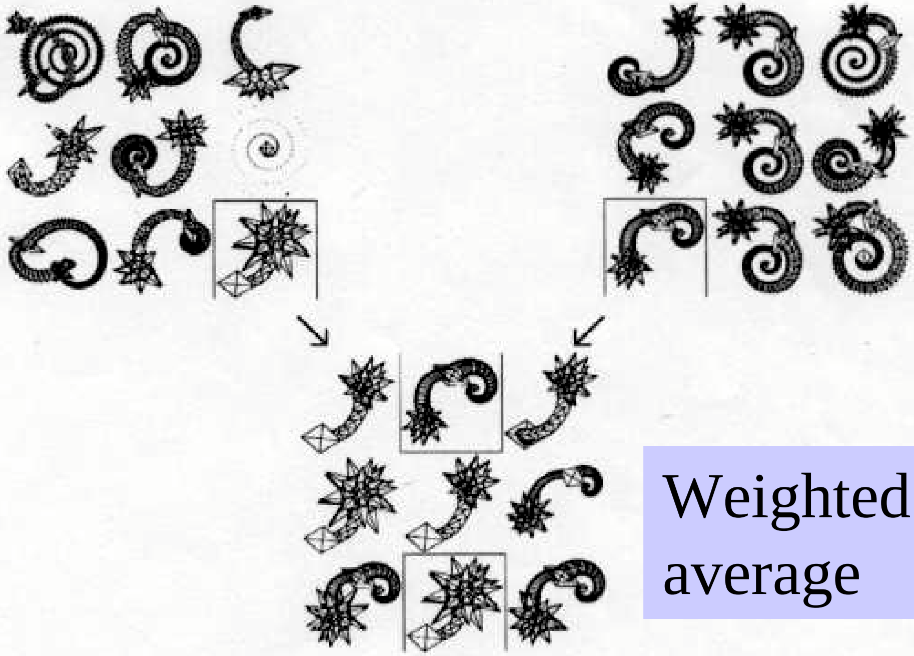
inbreeding



Distant marriage

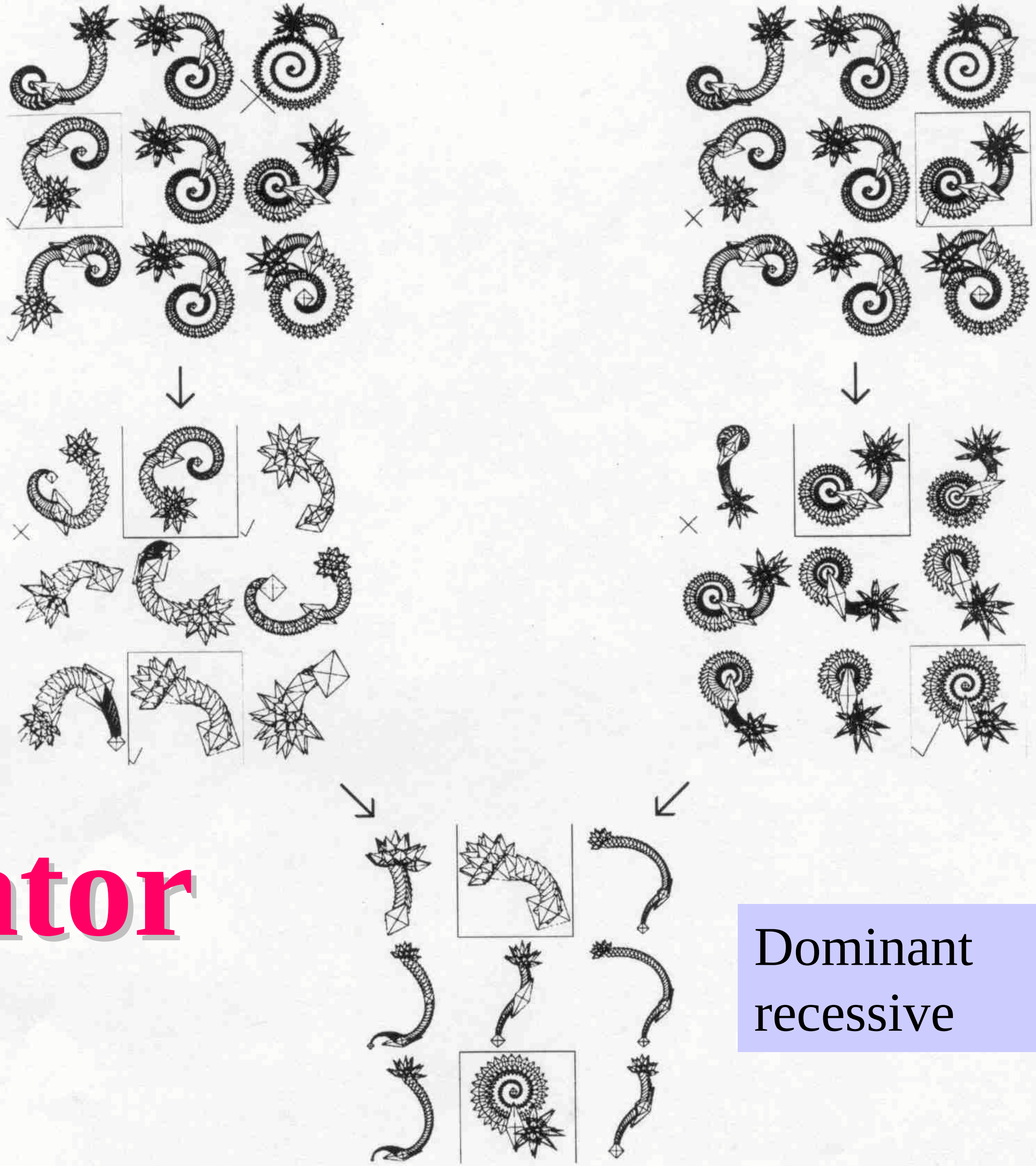
Mutator

spliced



Weighted average

Mutator



spliced

Weighted average

Dominant recessive

Mutator

Part of the evolutionary tree

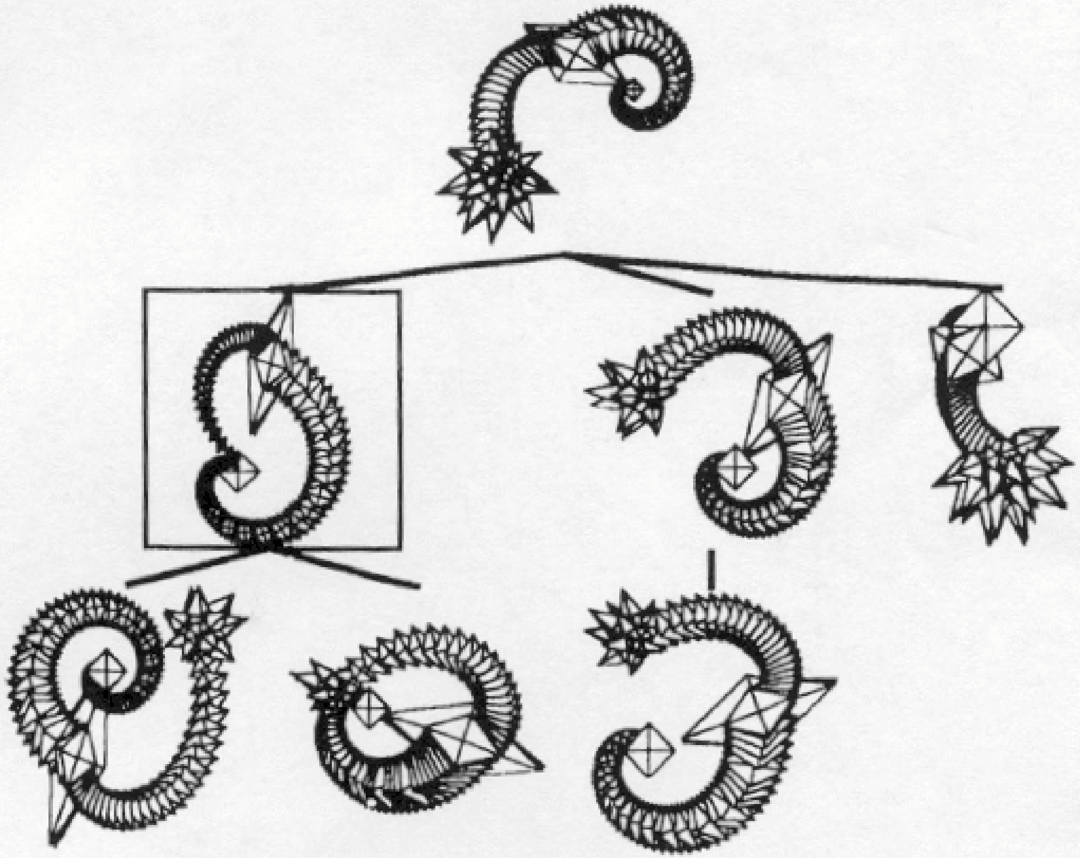
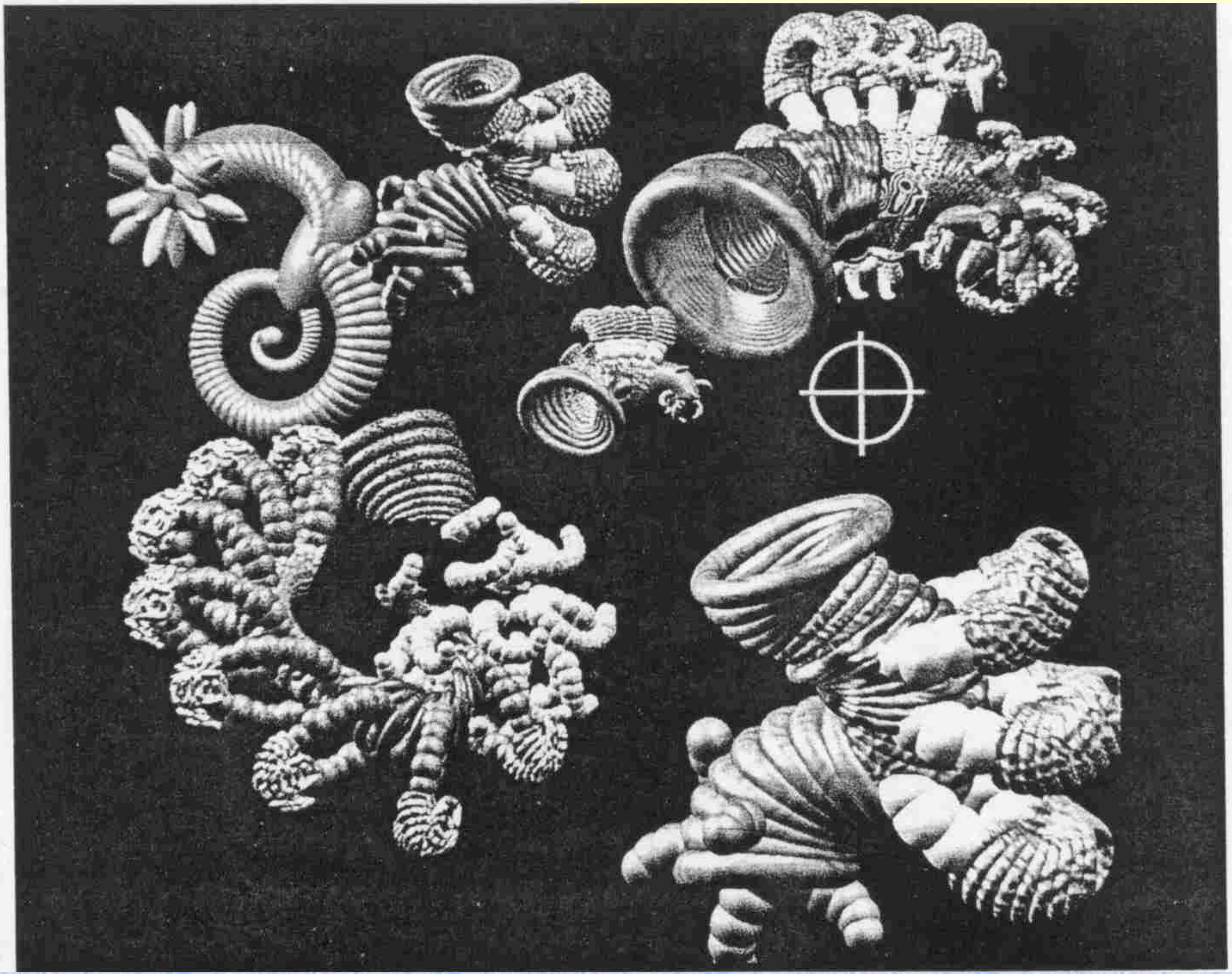


Figure 9.16 Extract from an evolutionary tree. The tree has become too large to display clearly, so the artist has restricted the display to include only frames between one level above and one level below the current frame. Cousin frames are not displayed.

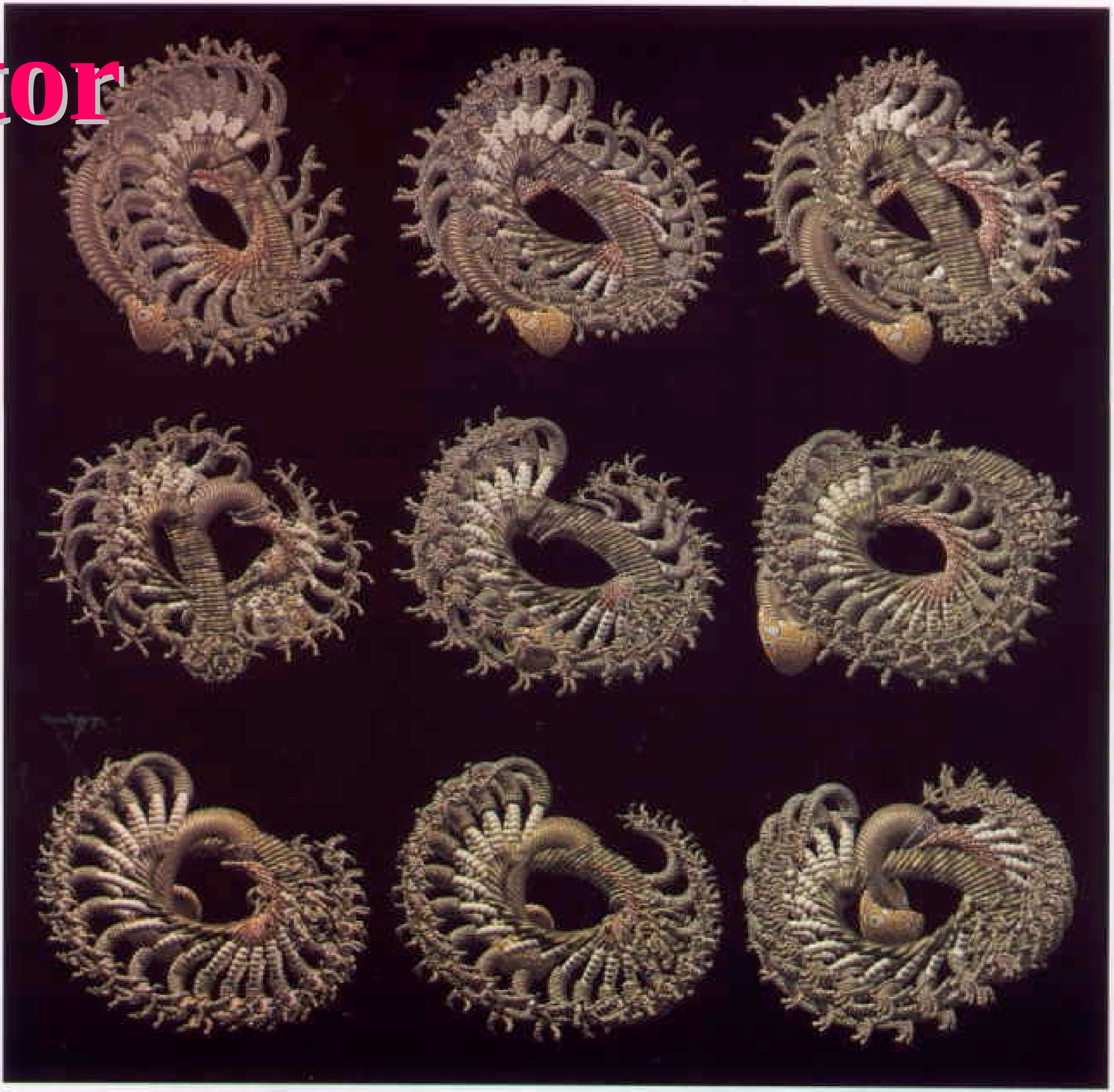
Mutator

Mutator

- Continuous Mutator session
- as in animation from film “Mutations”

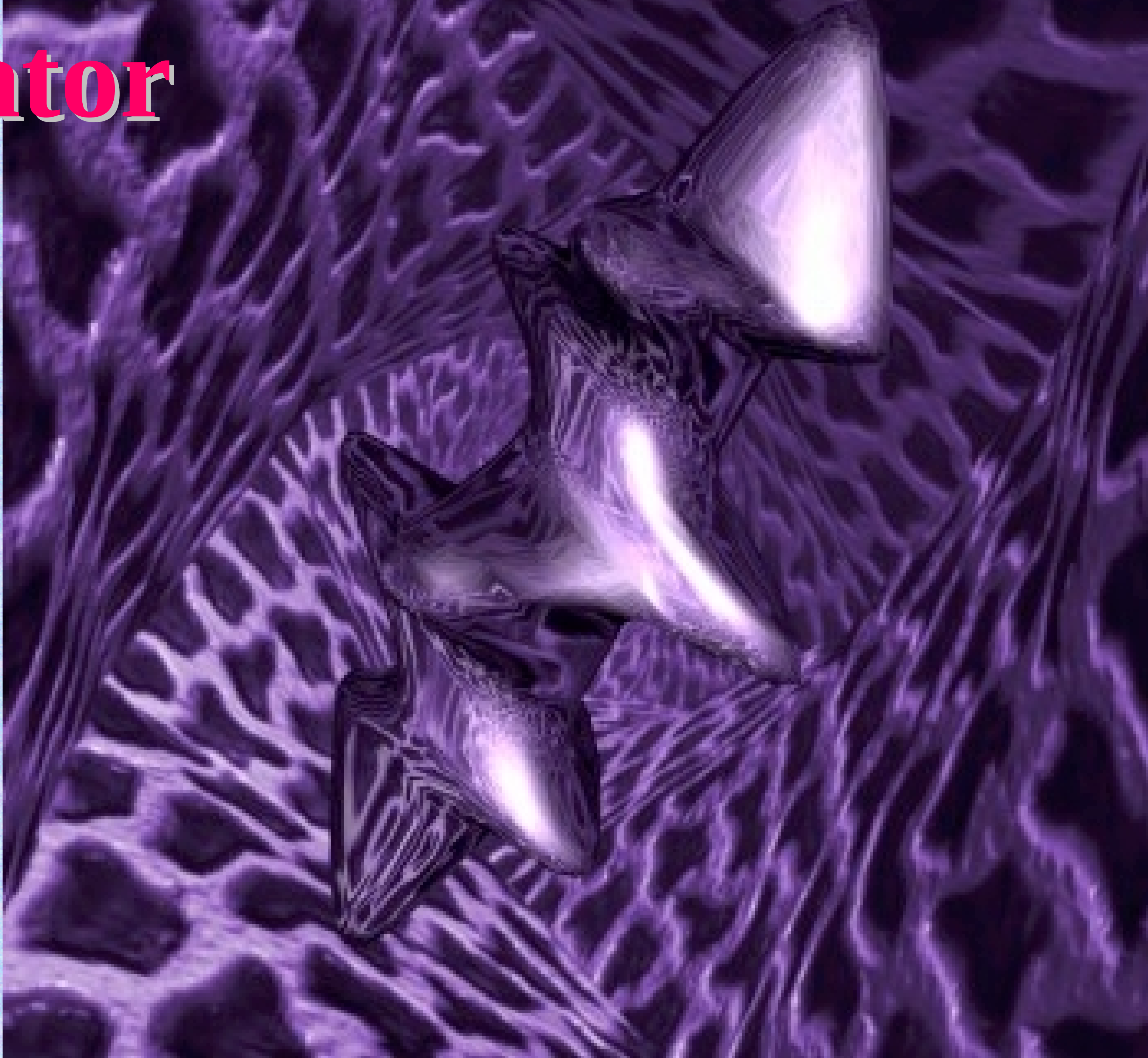


Mutator

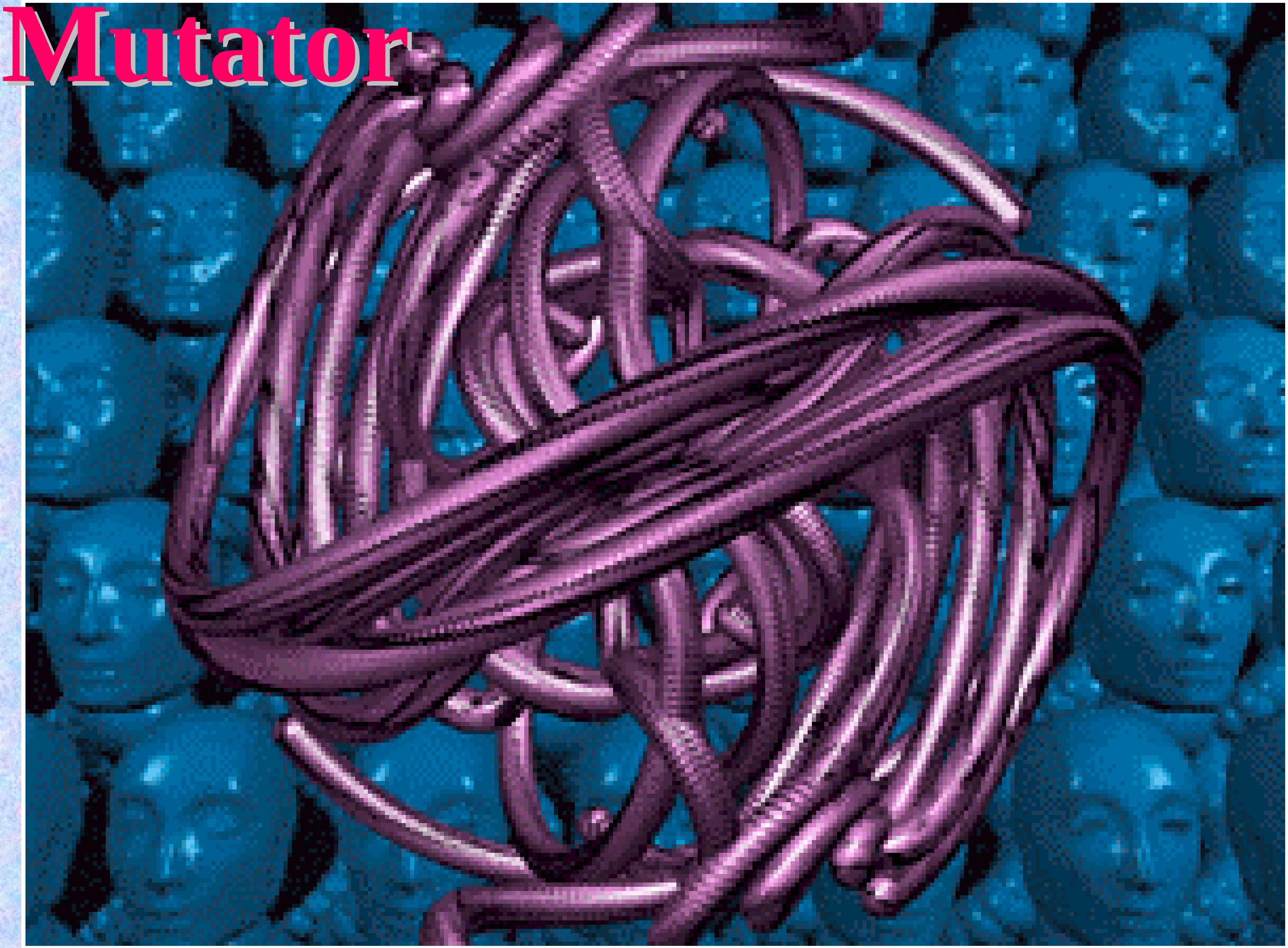


Mutator

Mutator

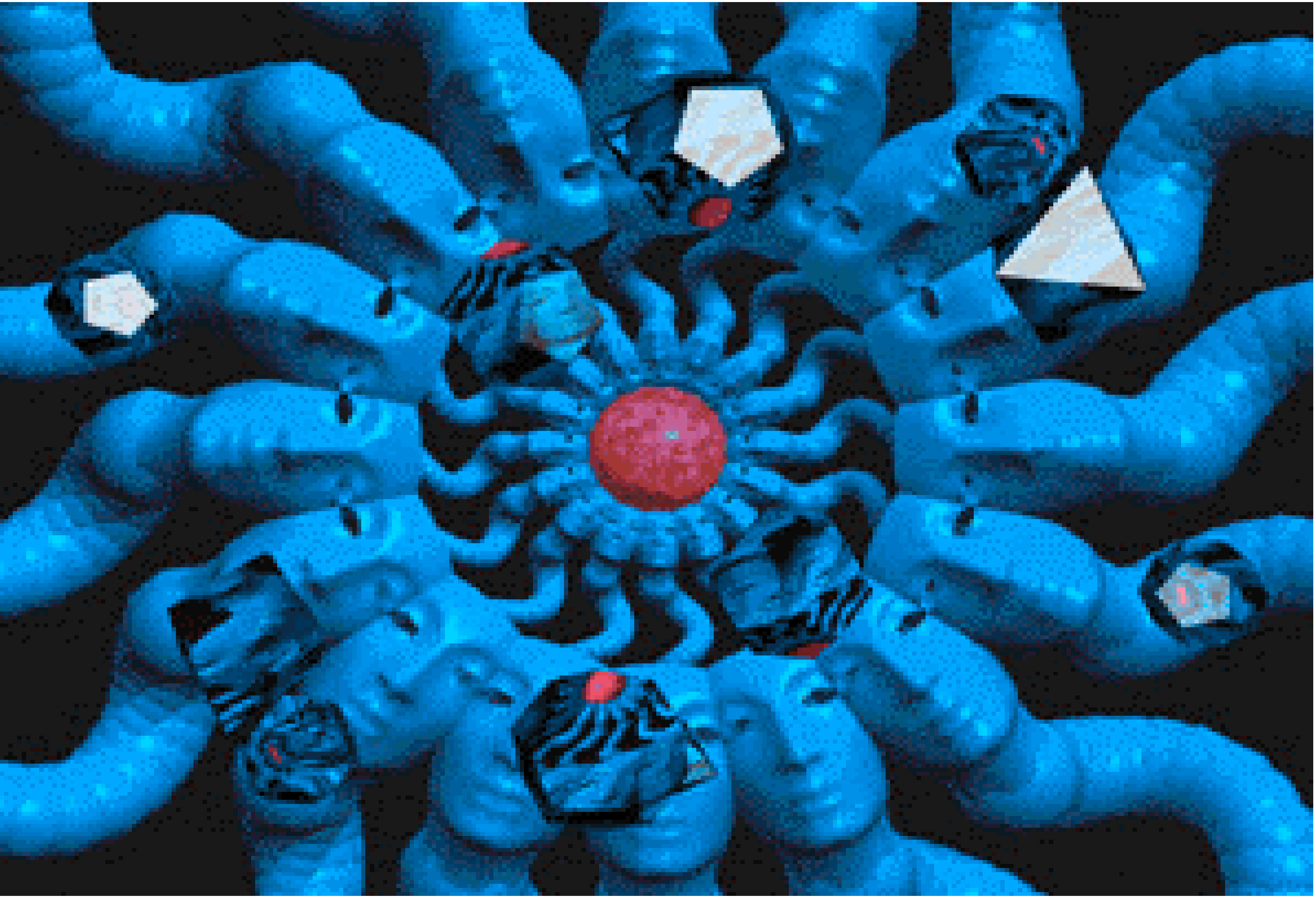


Mutator



Mutator

Mutator



Mutator

Computer Evolution of Buildable Objects

- Project of Pablo Funes and Jordan Pollack

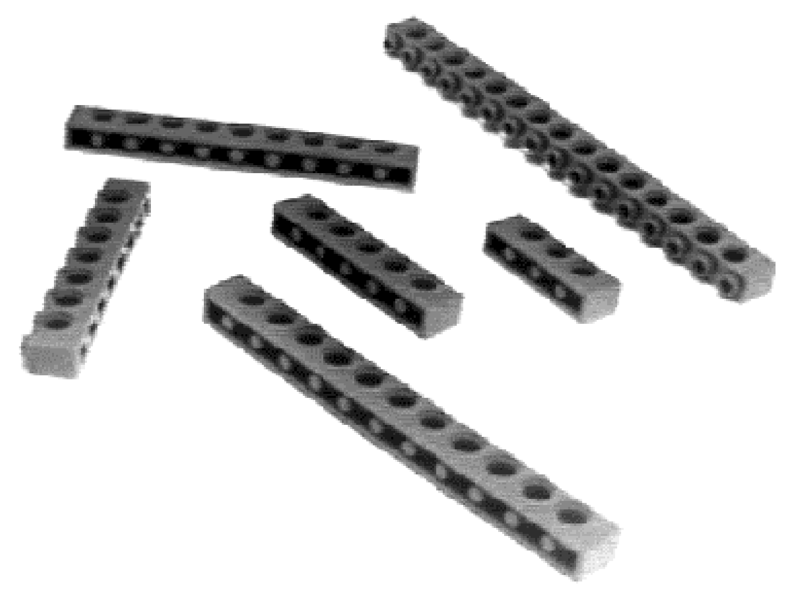
Taken from http://www.cs.brandeis.edu/~pablo/indexe.html

Taken from http://www.cs.brandeis.edu/~pollack/

Project Details

- Used computers to generate 2-D and 3-D objects in simulation that would perform correctly in the real world.

- **Used Lego to build and test the designs**

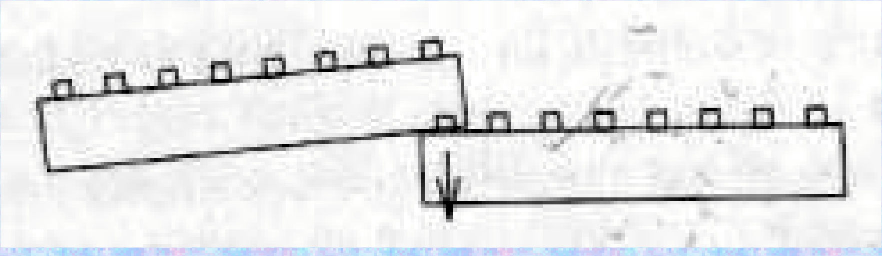


Why use Lego?

- Can easily build cheap and handy structures
- Have a property which simplifies the experiment and eases design consideration
- What property?
  - “The ***resistance*** of Lego blocks far surpasses the force necessary to either join two of them together or break their unions.”

Simplification of Model

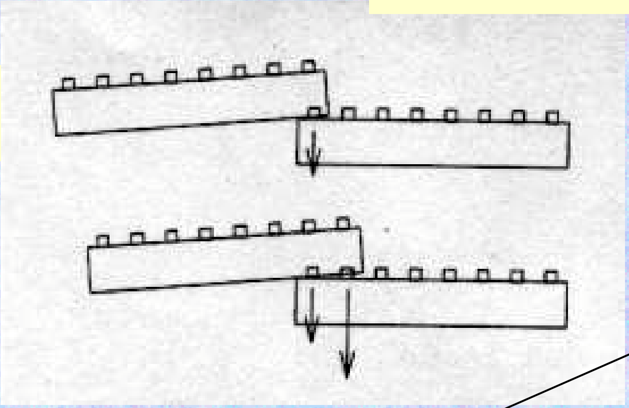
- To simplify the model, only the ‘fulcrum’ effect acting on a pair of Lego blocks was considered.



- It was assumed that radial forces (such as vertical pulls) would not occur.

Minimal Torque Capacities

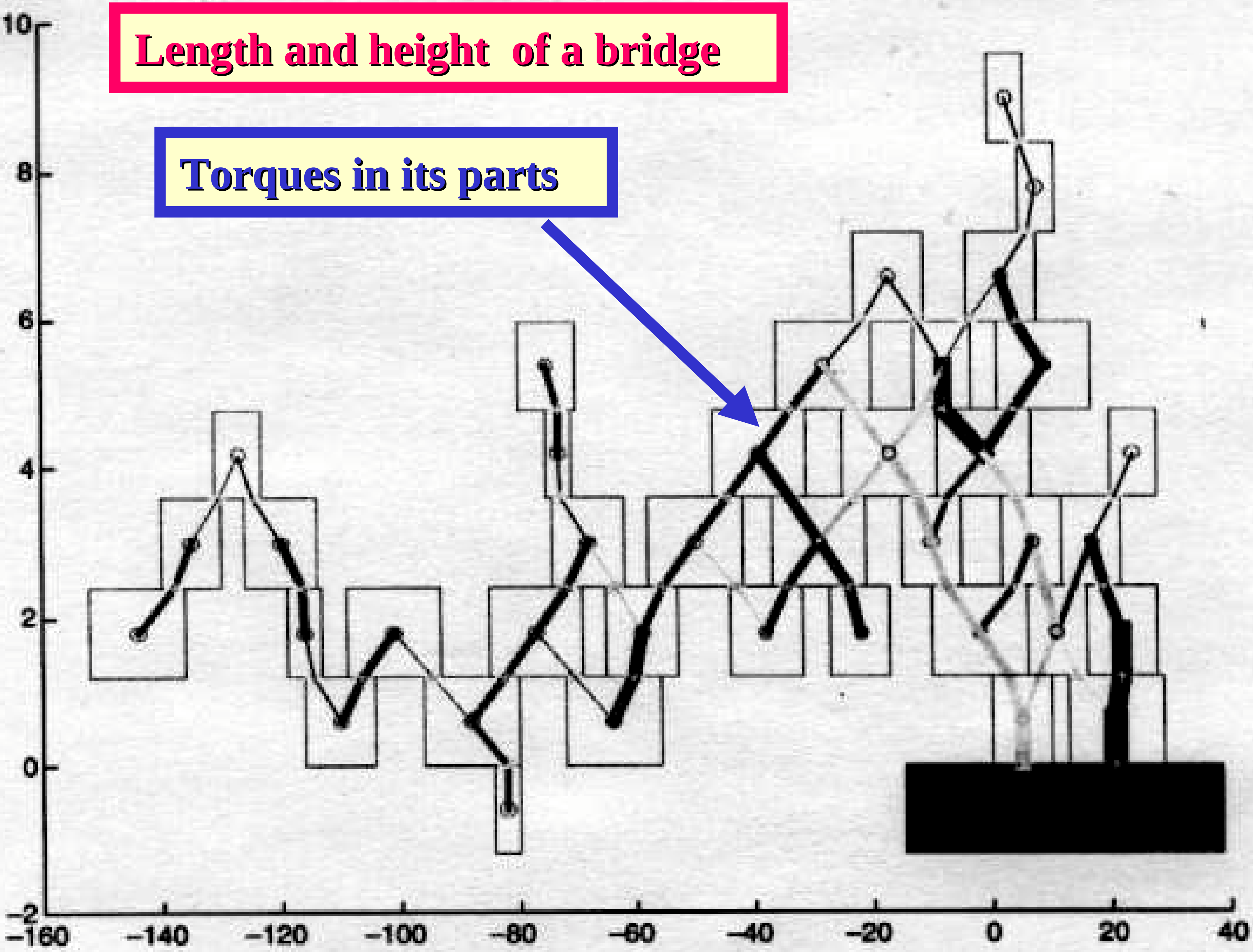
As functions of numbers of knobs in a connection



One knob

Two knobs

Joint Size (knobs)	Approximate Torque Capacity (N-m * 10^{-6})
1	10.4
2	50.2
3	89.6
4	157.3
5	281.6
6	339.2
7	364.5



Length and height of a bridge

Torques in its parts

Operating Heuristic

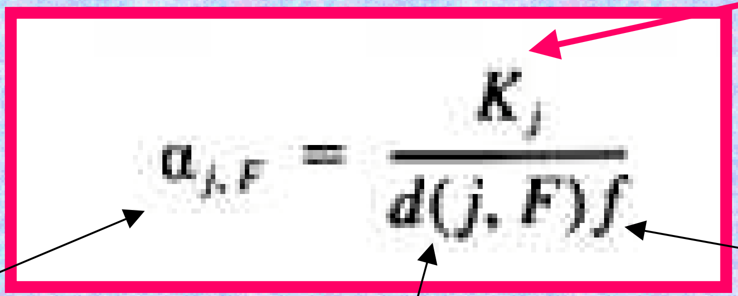
The structure will not break, if:

there is a way to distribute the weights among the network of bricks such that *no joint is stressed beyond its maximum* capacity

No complete algorithm has been found

Greedy Algorithm

- Does not guarantee that all solutions will be found.
- Each <u>joint</u> ***j*** can support a certain fraction ***a*** of a force on the network.
- This fraction is given by:



$$\alpha_{j,F} = \frac{K_j}{d(j,F)f}$$

maximum capacity of the joint

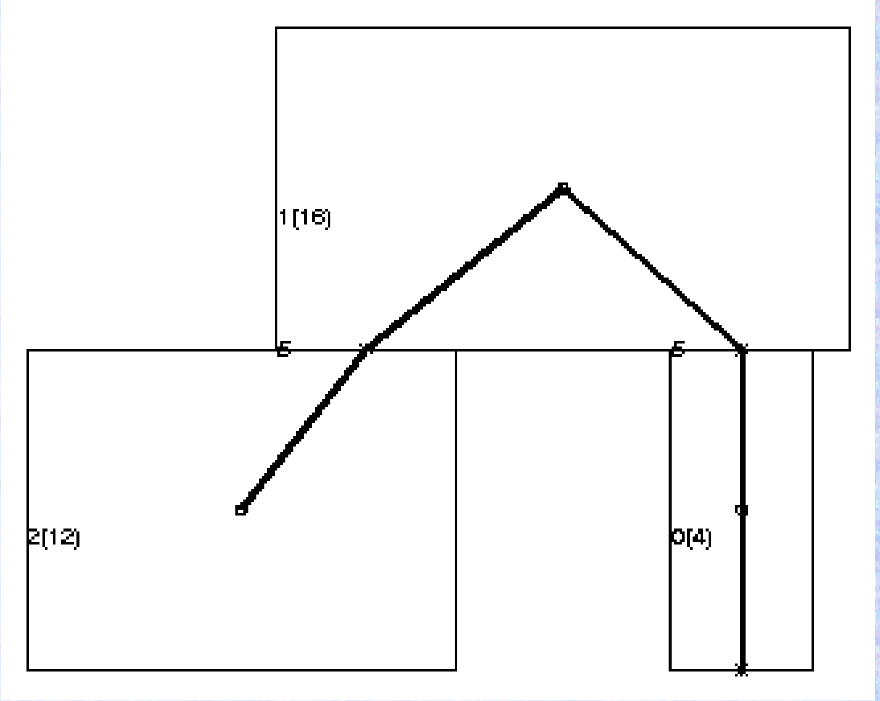
magnitude of the force

fraction ***a*** of a force

distance between the line generated by the force vector and the joint

Greedy Algorithm

- Find a solution for the distribution of the first mass
- Fixe this solution
- The remaining capacity for each joint is computed.
- This gives a reduced network that must support the next force.

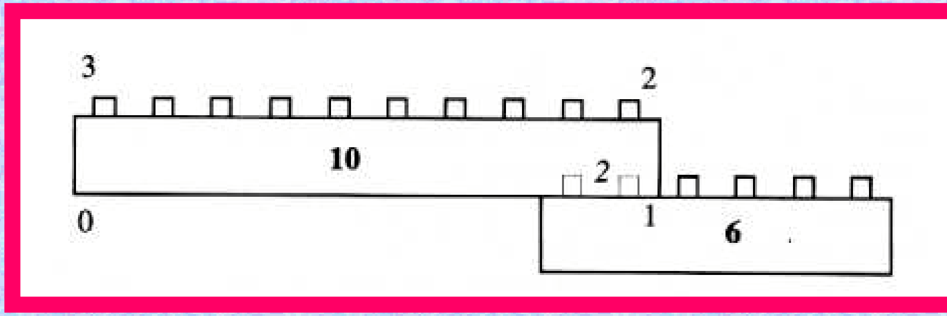


Evolutionary Algorithm?

- A *steady-state*, genetic algorithm was used to solve the problem
- Initialized with a population of a **single brick**
- Through mutation and crossover, a population of 1000 individuals was generated

Encoding Explained for 2-D structure

- Uses pseudo-Lisp notation: structure=Lisp expression
- Individual brick:
  - (10 nil nil nil nil)
- Joined by two knobs:
  - (10 nil (2 (??)) nil nil)
- With a **6 knob brick**:
  - (10 nil (2 **(6 nil nil nil)**) nil nil)



Encoding for 2-D structure

Number of corner

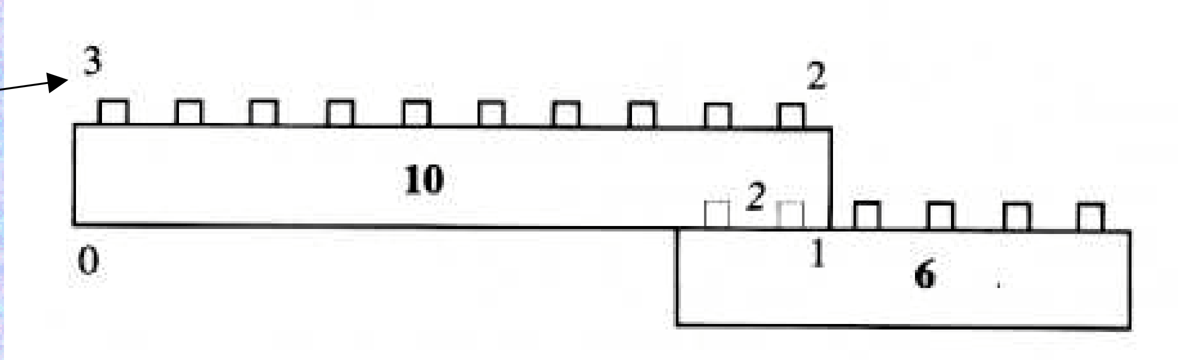


Figure 17.7 Example of 2D genetic encoding of bricks.

- This join was encoded as:

(10 nil (2 (6 nil nil nil)) nil nil)

Block with 10 knobs

Connected by 2 knobs

Located in corner 1

Mutation and Crossover

- Mutation operates by:
  - either random modification of a brick's parameters (size, position, orientation)
  - or addition of a random brick.
- Crossover involves two parent trees out of which *random subtrees are selected*.
- The offspring generated has the first subtree removed and replaced by the second.

Fitness Function

- No any knowledge about good design or common engineering practices that would bias the results
- Provides measures of feasibility and functionality

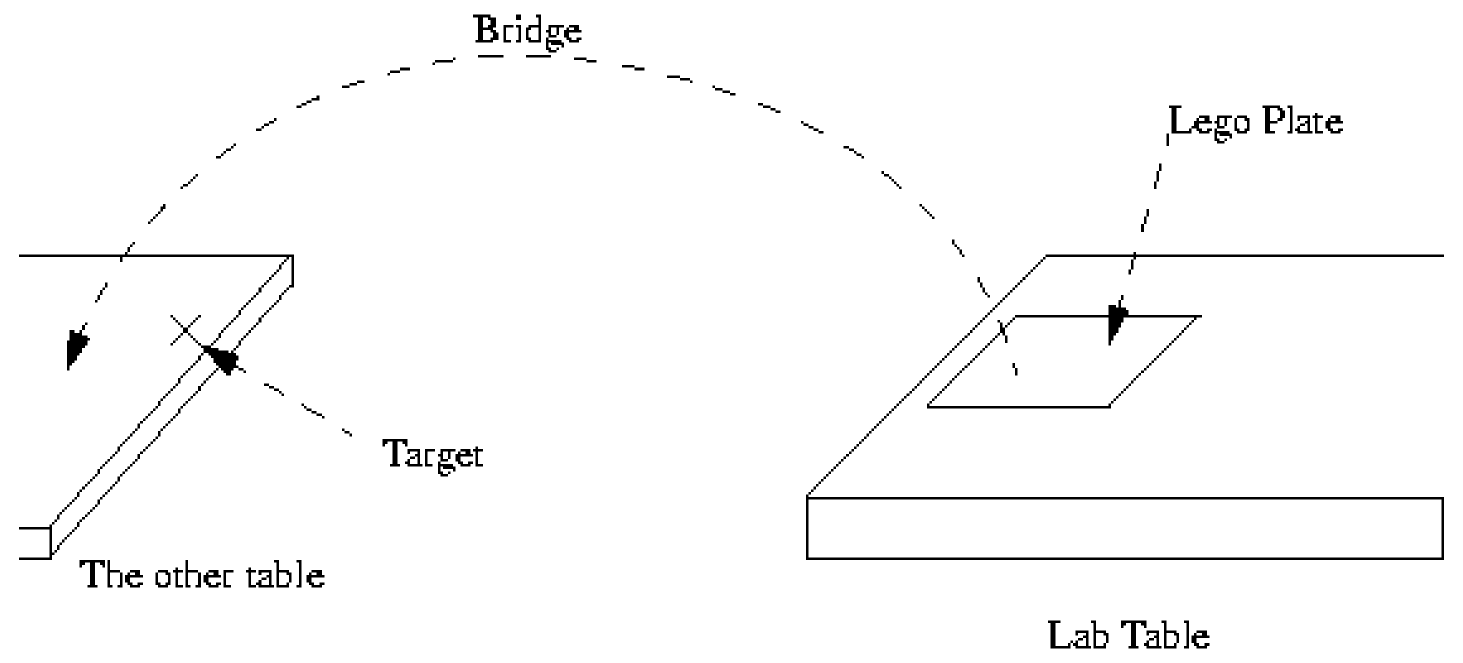
Algorithm

```
While maximum fitness < Target fitness
Do      Randomly select mutation or crossover
        Select 1 (for mutation) or 2 (for crossover) random individual(s)
                with fitness proportional probability
        Apply mutation or crossover operator
        Generate physical model and test for gravitational load.
        If the new model will support its own weight
                Then replace a random individual with it
                (chosen with inverse fitness proportional probability)
```

Practical Examples

- Reaching a target point:
  - Bridges and Scaffolds
- External Loads:
  - Horizontal Crane Arm
- Constraining the space:
  - Diagonal Crane Arm

Reaching a Target Point



- **Fitness Function:**
  - **Normalized distance** to the target:

the structure

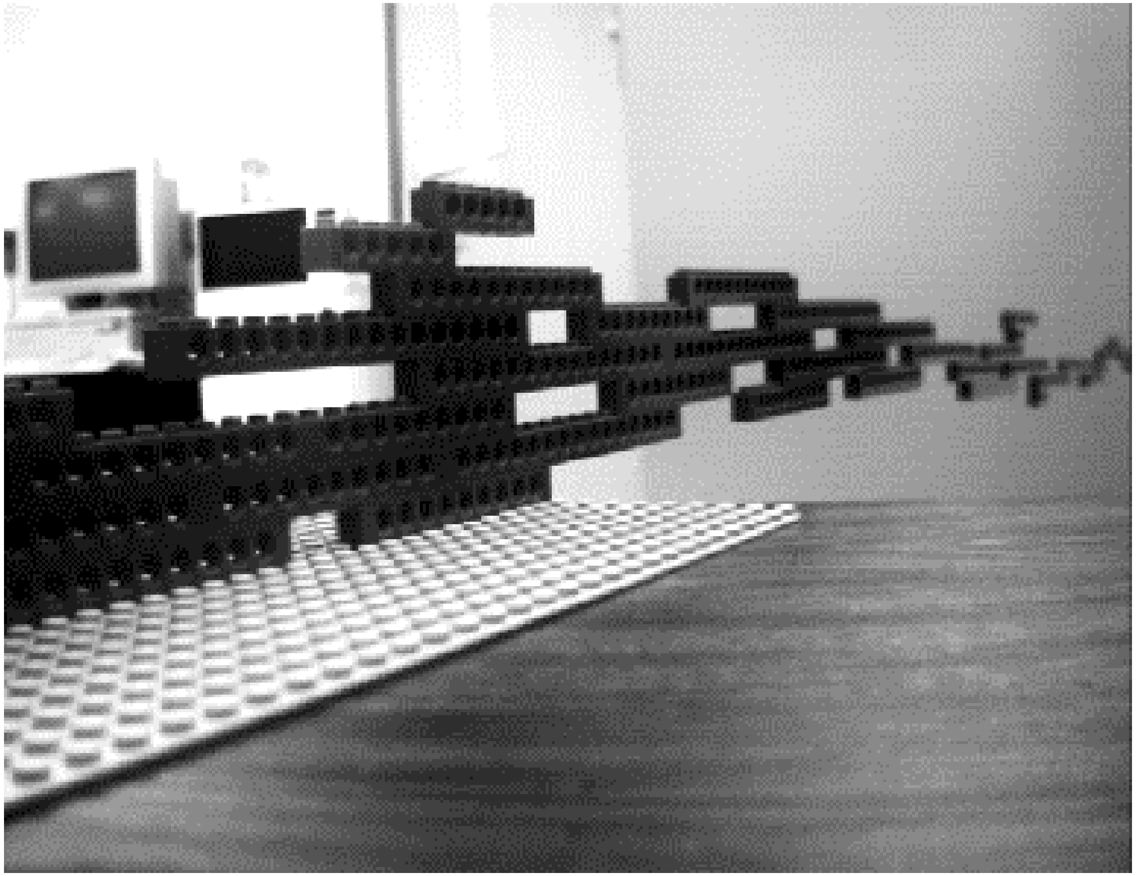
the target point

$$Nd(S,T) = 1 - \frac{d(S,T)}{d(0,T)}$$

d is the Euclidean distance

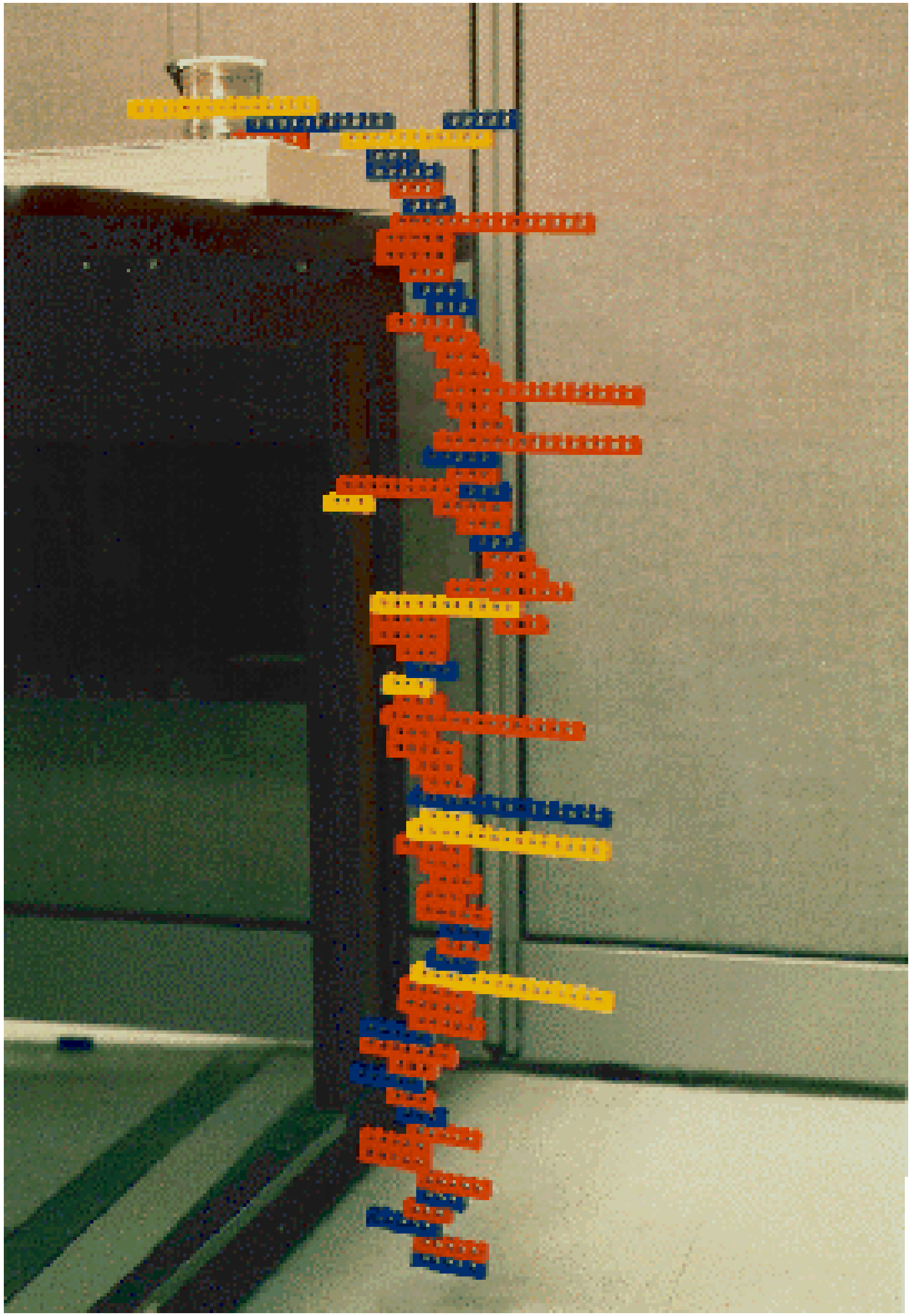
Bridge

- An example successful run
- The target fitness was reached after 133,000 generations

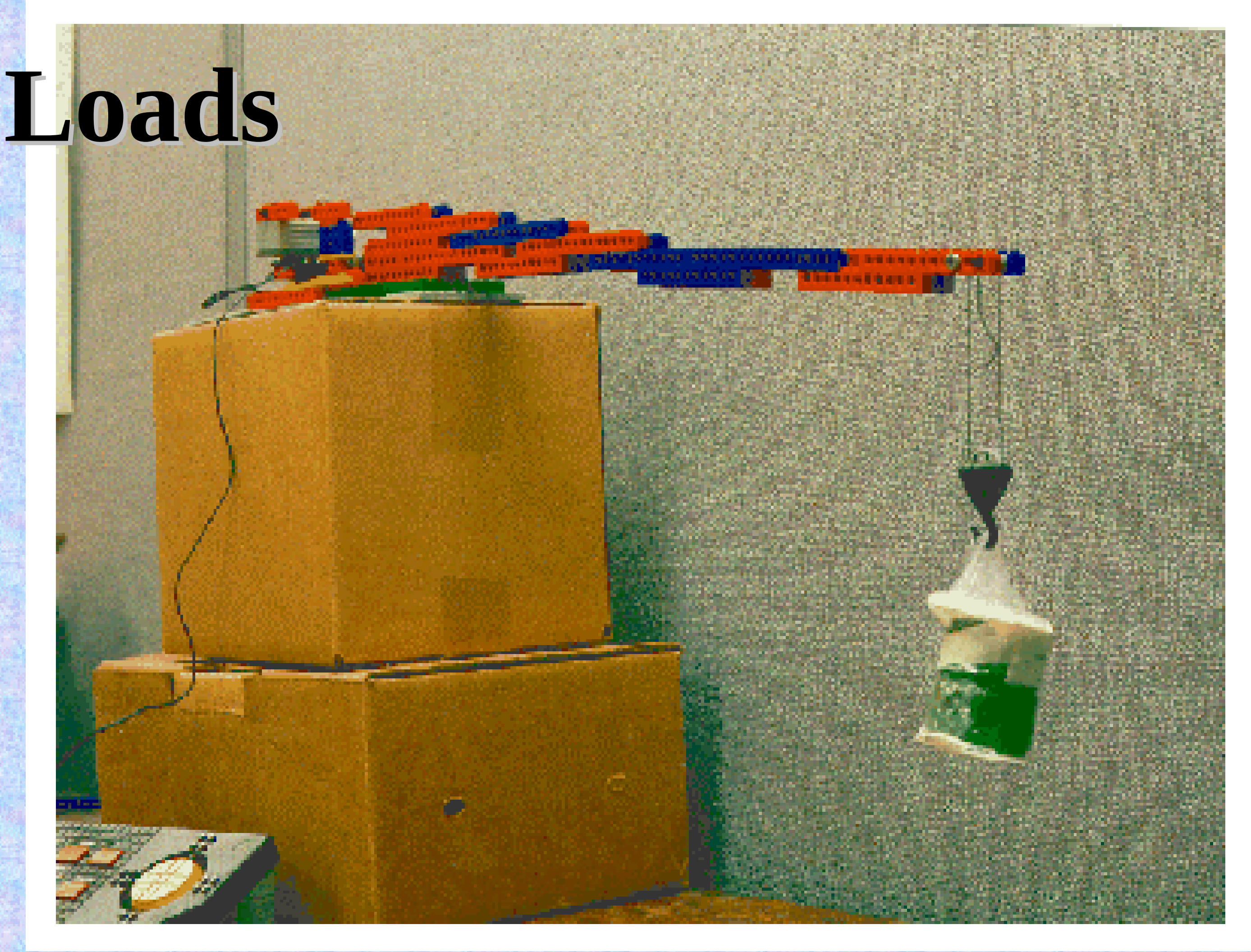


Scaffold

- Evolved in 40,000 generations



External Loads



- Uses a **two-step fitness** function
- **Weight is added** in small increments to see how much the structure can hold

Constraining the Space

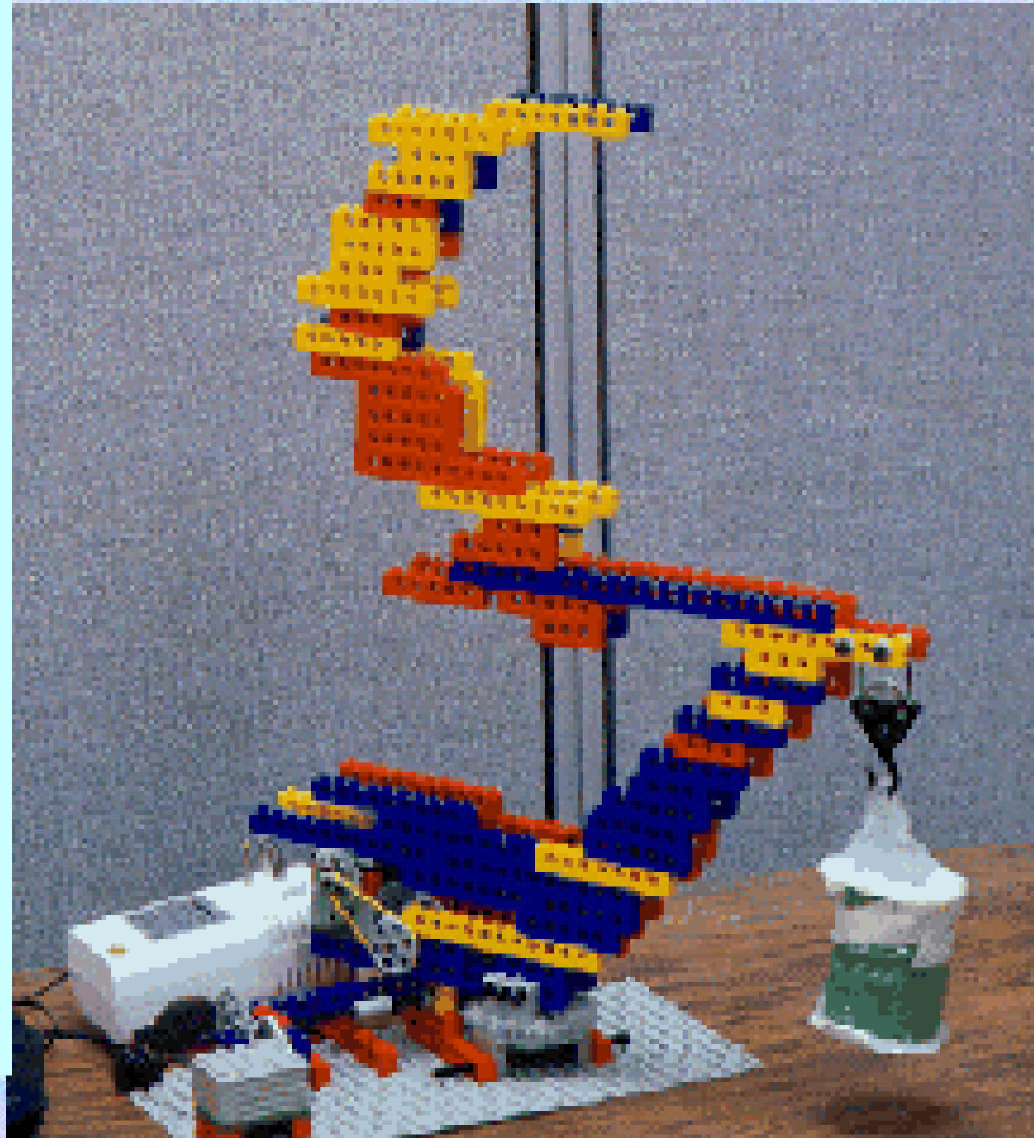
- Bricks could only be placed above the diagonal

- ***Fitness Function:***

fraction of weight supported

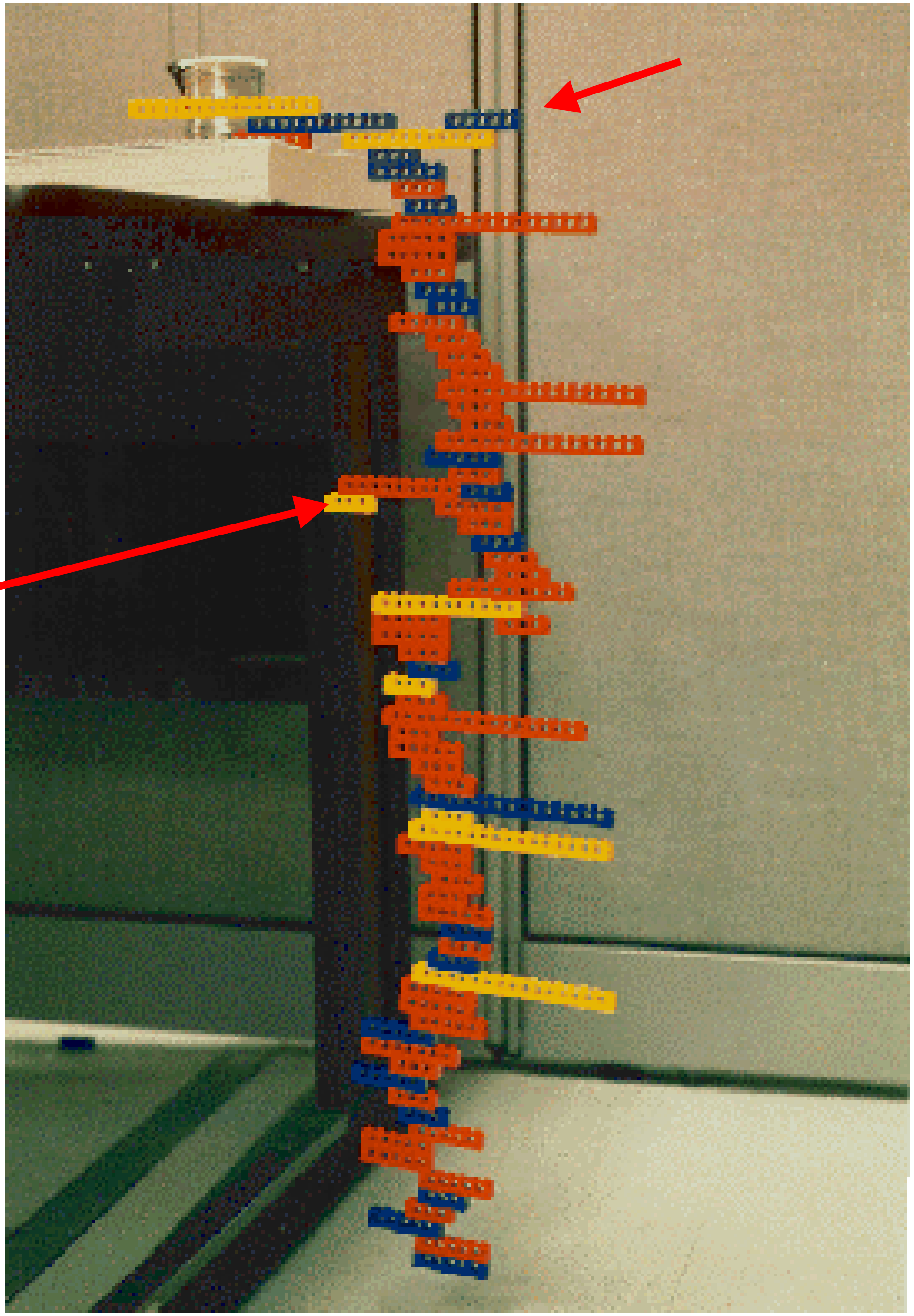
*****

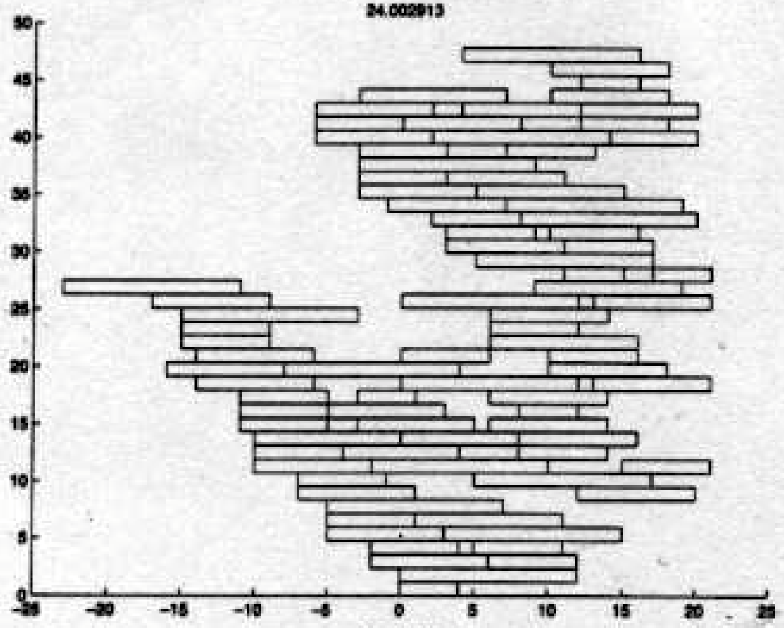
length of the arm along the x axis



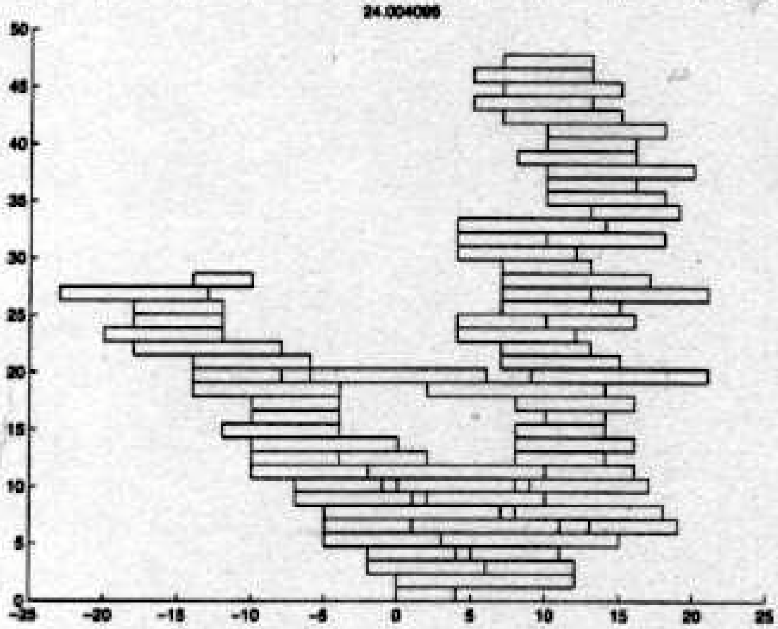
Optimization

- Usually don't reward or punish for the *number of bricks* used
- Leads to unused bricks
- Can add a little **reward for lightness**





Fitness:
24.002913



Fitness:
24.004095

Limitations

- Noise
- Safety concerns
- No complete algorithm has been found

→ results in a conservative model

Sources

Dianna Fox
and
Dan Morris