Greetings from Maui to China

Evolved only in Hawai`i, endangered, and the source behind the term "wiki".

Evolutionary Computation

Lee Altenberg

University of Hawai`i at Manoa
altenber@hawaii.edu
http://dynamics.org/Altenberg/

Goals of my lectures:

- Introduce you to core concepts in **evolutionary computation**, a form of *heuristic search*:
  - **WHAT**: Representations & operators
  - **HOW**: No Free Lunch theorems
    - Knowledge incorporation
  - **WHY**: Spectral analysis
    - Rapid mixing and rapid first hitting time
    - Higher order phenomena

Evolutionary Computation

A deep "reverse engineering" of nature

- Living things possess functionality that people have sought to reverse-engineer:
  - Intelligence, flight, immunity, mechanical force, defenses, locomotion, vision, etc.
- EC: Reverse-engineer *the process* that produced complex adaptations in nature: Darwinian Selection.

When is Evolutionary Computation useful?

- When there is a problem where *you know what you want,* but
- you just don't know *how to go about getting it.* *(Hendrix, 1967)*

- "We know what we want", i.e.
  - **We have a space S = {x} of things to search;**
  - **and an objective function y=F(x) that says how good each thing x ∈ S is;**
- "But we just don't know how to go about getting it." i.e.
  - **We don't know the optimal x***
  - **We don't have $x=F^{-1}(y)$ to compute x's from maximal y's.**
    - **Known as the "inverse problem".**

Great source of inverse problems: complex systems!

- There can be complex mappings from *system components* to *emergent behaviors*
- Inverse problems exist in
  - physical,
  - mathematical, and
  - symbolic systems

For example:

Seismic waveform inversion

- Goal: determine subsurface geological structure.
- Data: Seismic waves traveling through strata are transformed by the strata into a signal.
- Inversion problem:
  - Given the waves and the structure, we *can* compute the signal.
  - Given the waves and the signal, we cannot compute the *structure.*

For example:

Neural Network Behavior

- Goal: create a neural network to execute a given mapping.
- Inversion problem:
  - Given a neural network, we *can* compute the mapping.
  - Given a mapping, it may be arduous or intractable to compute the *network structure* that would produce it.

For example:

Electronic Circuit Design

- Goal: create a circuit that behaves according to a specification.
- Inversion problem:
  - Given the circuit, we *can* compute the behavior.
  - Given the behavior, there may be no way to compute a circuit that produces it.

For example:

Boeing 777 jet engine

Traveling Salesman Problem

$$\begin{pmatrix}
0 & 303 & 667 & 383 & 430 & 244 & 180 & 553 & 98 & 771 & 725 & 440 & 601 & 38 & 727 & 415 \\
303 & 0 & 618 & 343 & 243 & 513 & 676 & 330 & 236 & 766 & 744 & 175 & 406 & 425 & 499 & 735 \\
667 & 618 & 0 & 553 & 812 & 854 & 553 & 469 & 497 & 718 & 727 & 502 & 311 & 482 & 358 & 779 \\
383 & 343 & 553 & 0 & 276 & 157 & 523 & 746 & 177 & 348 & 121 & 399 & 191 & 612 & 313 & 482 \\
430 & 243 & 812 & 276 & 0 & 410 & 382 & 643 & 837 & 453 & 380 & 766 & 332 & 490 & 360 & 338 \\
244 & 513 & 854 & 157 & 410 & 0 & 523 & 728 & 423 & 614 & 432 & 173 & 420 & 42 & 523 & 639 \\
180 & 676 & 553 & 523 & 382 & 523 & 0 & 411 & 593 & 488 & 501 & 600 & 588 & 611 & 469 & 439 \\
553 & 330 & 469 & 746 & 643 & 728 & 411 & 0 & 834 & 692 & 35 & 785 & 929 & 738 & 478 & 222 \\
98 & 236 & 497 & 177 & 837 & 423 & 593 & 834 & 0 & 602 & 288 & 246 & 627 & 67 & 485 & 456 \\
771 & 766 & 718 & 348 & 453 & 614 & 488 & 692 & 602 & 0 & 728 & 215 & 480 & 832 & 443 & 585 \\
725 & 744 & 727 & 121 & 380 & 432 & 501 & 35 & 288 & 728 & 0 & 609 & 671 & 142 & 318 & 544 \\
440 & 175 & 502 & 399 & 766 & 173 & 600 & 785 & 246 & 215 & 609 & 0 & 453 & 585 & 348 & 203 \\
601 & 406 & 311 & 191 & 332 & 420 & 588 & 929 & 627 & 480 & 671 & 453 & 0 & 631 & 557 & 494 \\
38 & 425 & 482 & 612 & 490 & 42 & 611 & 738 & 67 & 832 & 142 & 585 & 631 & 0 & 598 & 565 \\
727 & 499 & 358 & 313 & 360 & 523 & 469 & 478 & 485 & 443 & 318 & 348 & 557 & 598 & 0 & 586 \\
415 & 735 & 779 & 482 & 338 & 639 & 439 & 222 & 456 & 585 & 544 & 203 & 494 & 565 & 586 & 0
\end{pmatrix}$$

- **16 cities.**
- **120 distances.**
- **653837184000 circuits.**
- **Which circuits are shortest?**

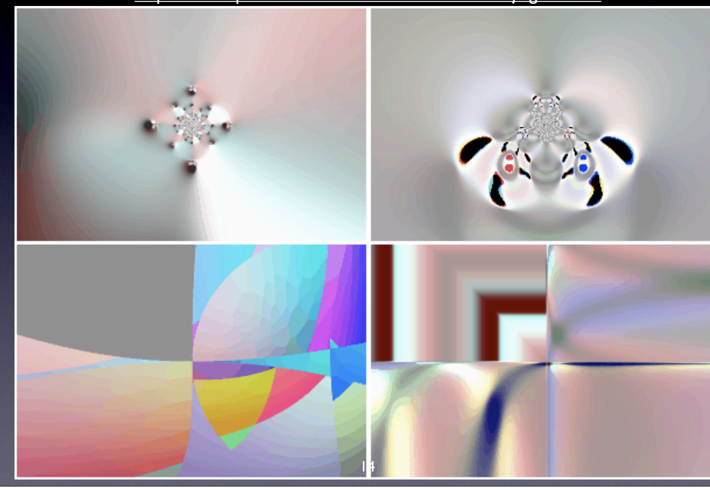
- Circuit length is an emergent property.
  - It is not a property of cities, but of the *relationship* between cities.
- The *number of possible relationships* between objects increases vastly faster than the *number of objects*.
- $|\{\text{cities}\}| = N$
- $|\{\text{circuits}\}| = (N-1)*(N-2)*(N-3)*\ldots* 4 * 3$

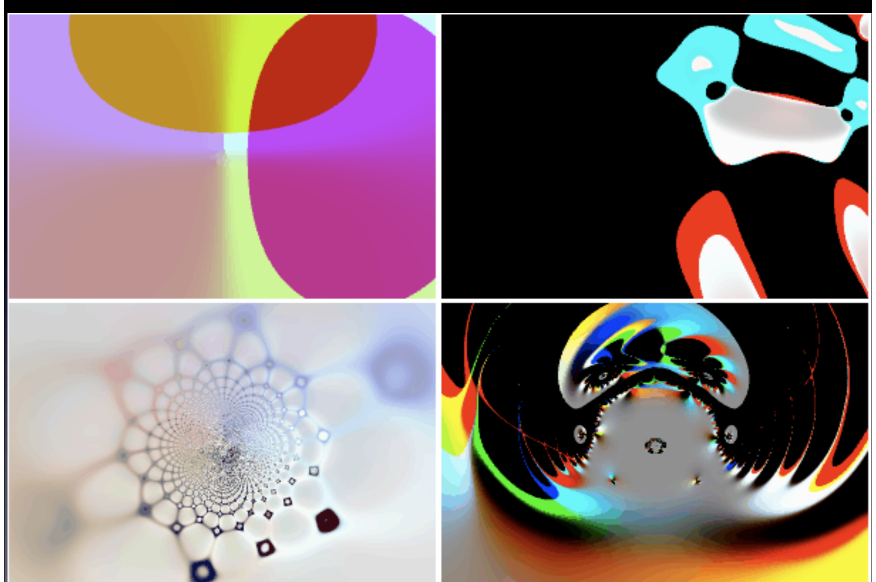
For example:

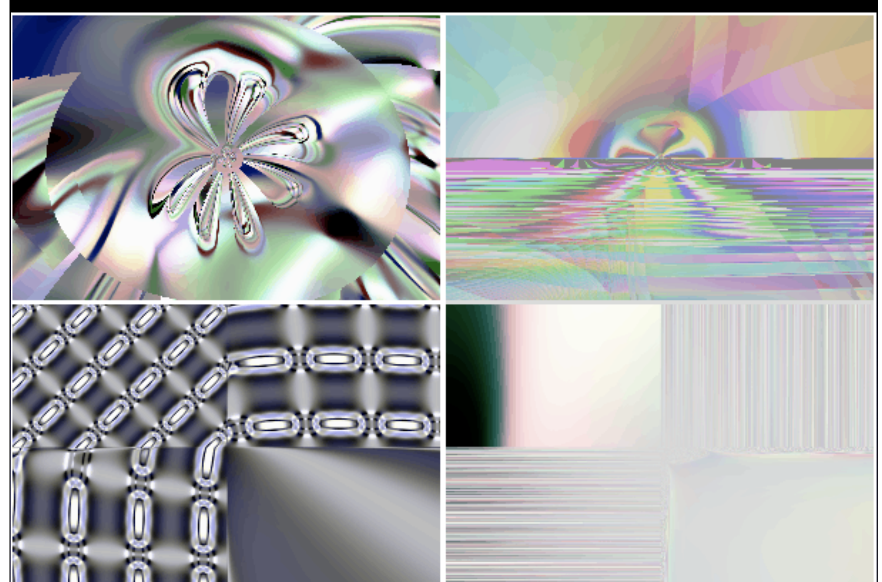
Evolved Art

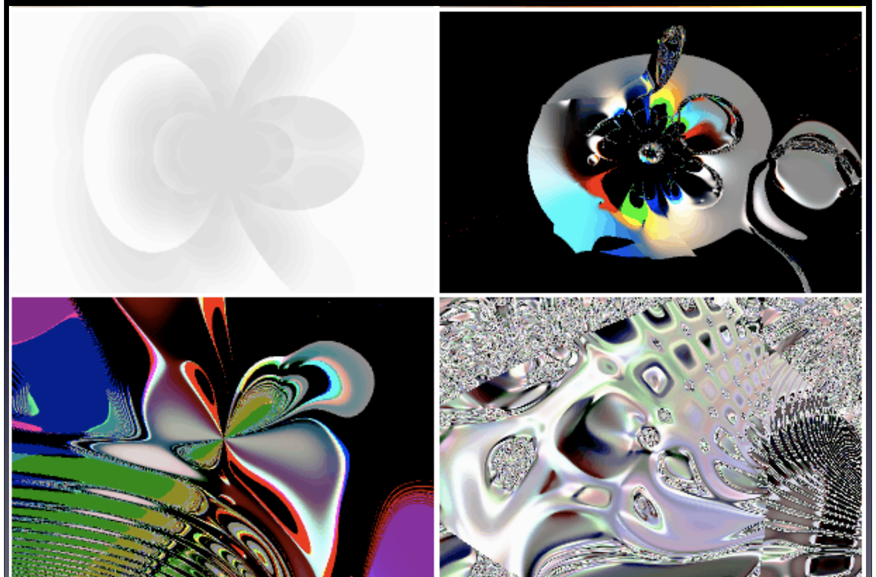
From a long-gone interactive Web site

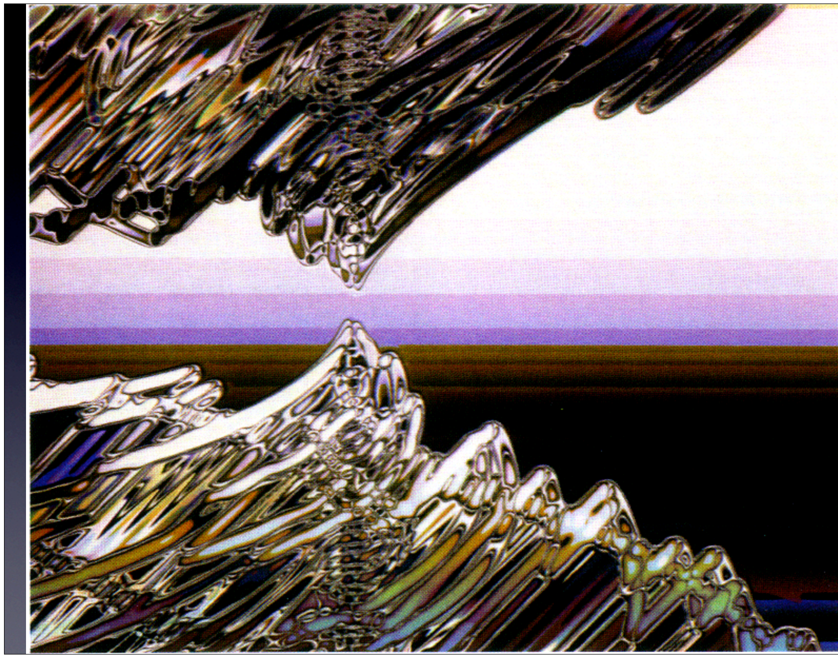
http://robocop.modmath.cs.cmu.edu:8001/htbin/mjwgenformll



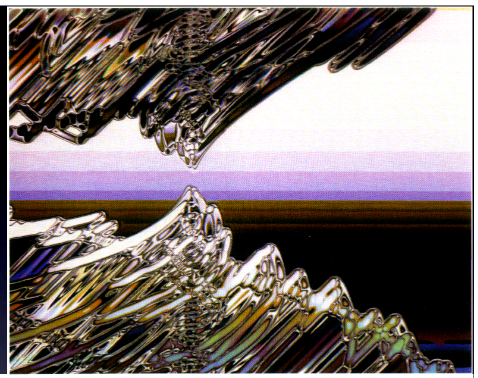






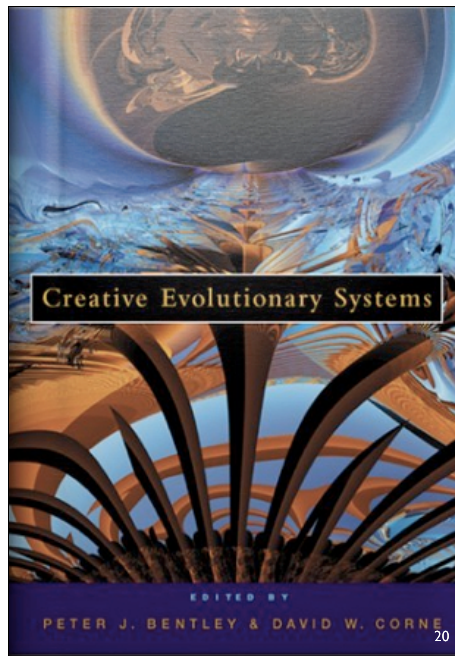


- The evolved code that generates the image:



```
(cos (round (atan (log (invert y) (+ (bump (+ (round x y) y) #(0.46 0.82 0.65) 0.02 #(0.1
0.06 0.1) #(0.99 0.06 0.41) 1.47 8.7 3.7) (color-grad (round (+ y y) (log (invert x) (+
(invert y) (round (+ y x) (bump (warped-ifs (round y y) y 0.08 0.06 7.4 1.65 6.1 0.54
3.1 0.26 0.73 15.8 5.7 8.9 0.49 7.2 15.6 0.98) #(0.46 0.82 0.65) 0.02 #(0.1 0.06 0.1)
#(0.99 0.06 0.41) 0.83 8.7 2.6))))) 3.1 6.8 #(0.95 0.7 0.59) 0.57))) #(0.17 0.08 0.75)
0.37) (vector y 0.09 (cos (round y y)))))
```

Artificial Evolution for Computer Graphics. Karl Sims. *Computer Graphics*, 25(4), July 1991, pp. 319-328.



Creative Evolutionary Systems . David W. Corne Peter J. Bentley (Morgan Kaufmann). 2001.

- contemporary melodies,
- photo-realistic faces,
- jazz music
- architectural designs,
- electronic circuits,
- novel aircraft maneuvers,
- 2- and 3-dimensional art,
- original proteins.

For example:

Virtual Creatures

- Goal: create a virtual physical structure controlled by a program that can locomote in a virtual physics world of water or land.
- Inversion problem:
  - Given the structure and the program, we *can* compute the locomotion behavior.
  - Given the desired locomotion, we have no idea how to compute a structure and program that generates it.

Evolved Creatures by Karl Sims

- Sims simulated a virtual physical world and the behavior of block structures run by programs, “virtual creatures”.
- Starting with randomly generated block structures and controllers, he produced offspring using genetic operators, and selected for desired movement.
- Iteration in the Connection Machine evolved successful (and suprising!) locomotion behaviors
  - http://www.archive.org/details/sims_evolved_virtual_creatures_1994

- Representation of the creatures

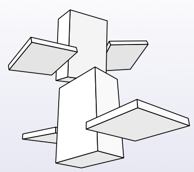


Figure 4a: The phenotype morphology generated from the evolved genotype shown in figure 3.

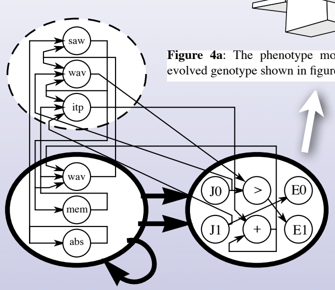


Figure 3: Example evolved nested graph genotype. The outer graph in bold describes a creature’s morphology. The inner graph describes its neural circuitry. J0 and J1 are joint angle sensors, and E0 and E1 are effector outputs. The dashed node contains centralized neurons that are not associated with any part.

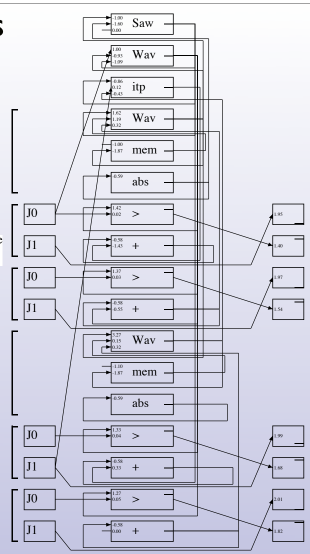


Figure 4b: The phenotype “brain” generated from the evolved genotype shown in figure 3. The effector outputs of this control system cause paddling motions in the four flippers of the morphology above.

"Evolving Virtual Creatures". K.Sims, Computer Graphics (Siggraph '94 Proceedings), July 1994, pp.15-22.

Elements of Darwinian Systems:

- As stated in the classical literature:
  - Heritable
  - Variation
  - In Fitness
- Stated as *forces*:
  - Selection and
  - Transformation

Heritable:

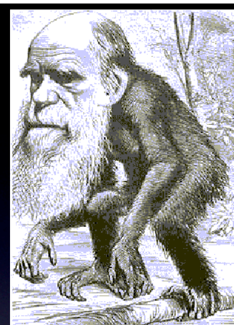
- 龙生龙,
- 凤生凤,
- 老鼠的儿子
- 会打洞.

Variation:

- 龙生凤,
- 凤生龙,
- 老鼠的儿子...

In Fitness:

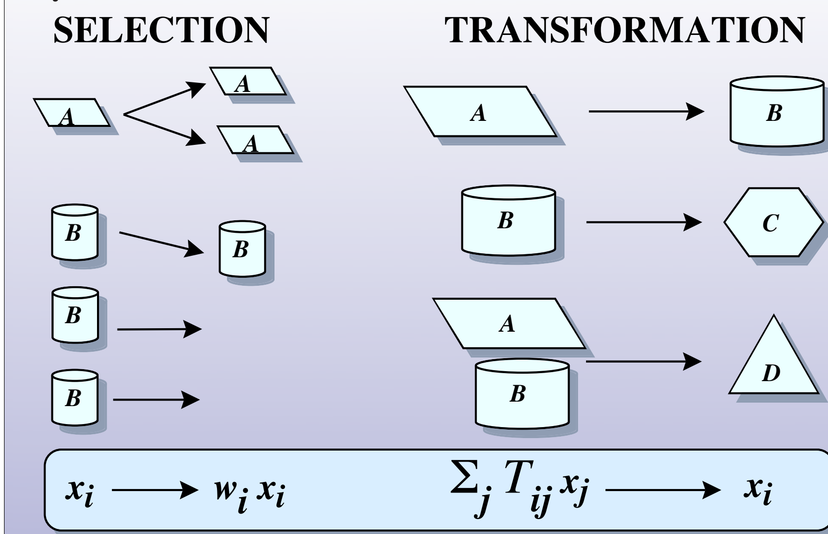
- 跑得比风还快.



Dragon begets dragon,
Phoenix begets phoenix,
and the son of the rat
digs a hole in the ground.

Dragon begets phoenix,
Phoenix begets dragon,
and the son of the rat
flies through the air!

The Two Elements of Darwinian Dynamics:

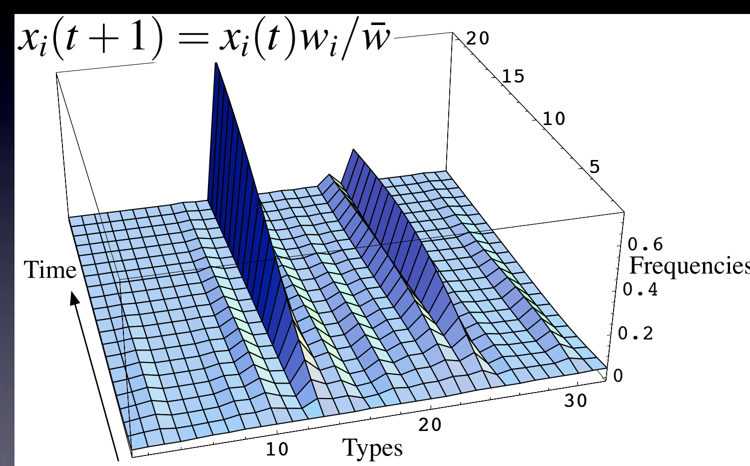


Selection & Transformation

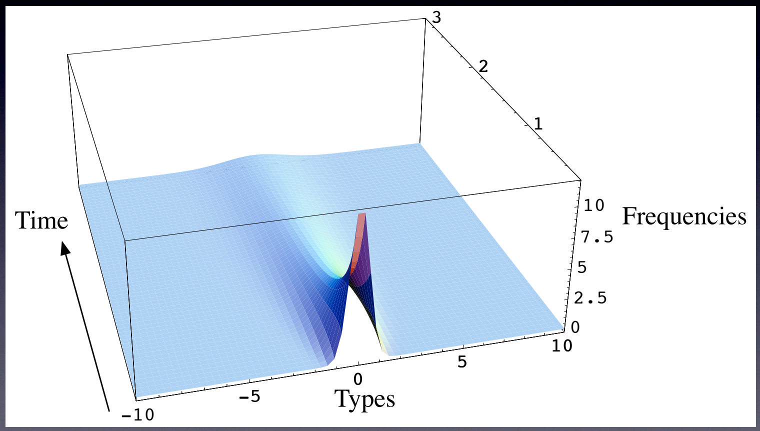
- **SELECTION** is a *concentrating operator*
- **TRANSFORMATION** is a *diffusing operator*

Operating alone, neither selection nor transformation produce signficant adaptation. Together, they can combine to create very powerful search dynamics.

Selection alone:



Transformation alone:



The Darwinian Heuristic:

- When g_1 produces a variant g_2, there is a *significant chance* that
  1. g_2 is better than g_1, and
  2. g_2 can produce a still-better variant, g_3.
- The better the parent, g_1, the better the chance of producing an even better g_2.
- i.e. there is **evolvability.**
- Thus iteration can continue the process.

Elements of Evolutionary Computation

1. Search Space – candidate solutions
2. Representation of the search space
3. Variation operators: transform elements of the search space into other elements
4. Objective function on elements of the search space
5. Selection operators: use the objective function to choose which elements to transform
6. Population: N individuals to be selected among and used as parents

1. Search spaces:

- Neural networks
- Function arguments
- Combinatorial spaces
- DNA sequence reconstructions
- Protein & DNA structure alignments
- Peptide structures
- Image recognition systems
- Cellular automata
- Seismic wave forms
- Process parameters
- Circuit designs
- Jet engine designs
- Scheduling sequences
- Computer programs
- Routing
- etc. etc.

2. Representations:

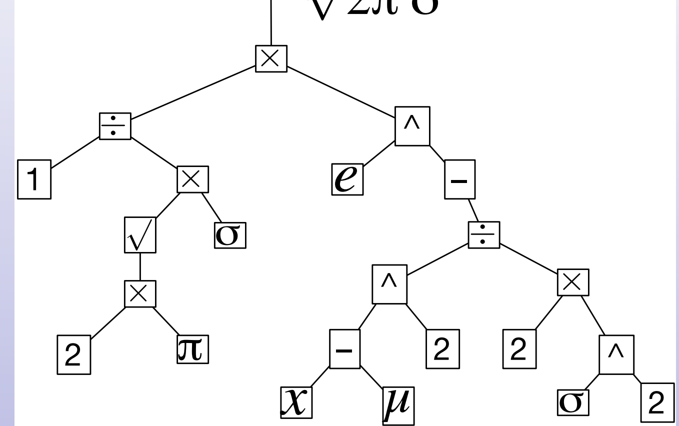
The actual data structures that represent the search space

Examples:

- The real numbers, R, or vectors in R^L
- Binary strings, $\{0,1\}^L$
- Permutations, combinations
- Trees and weights (for neural networks)
- Generative grammars or algorithms
- Parse trees

Algorithms: Genetic Programming

$$N(x,\mu,\sigma) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

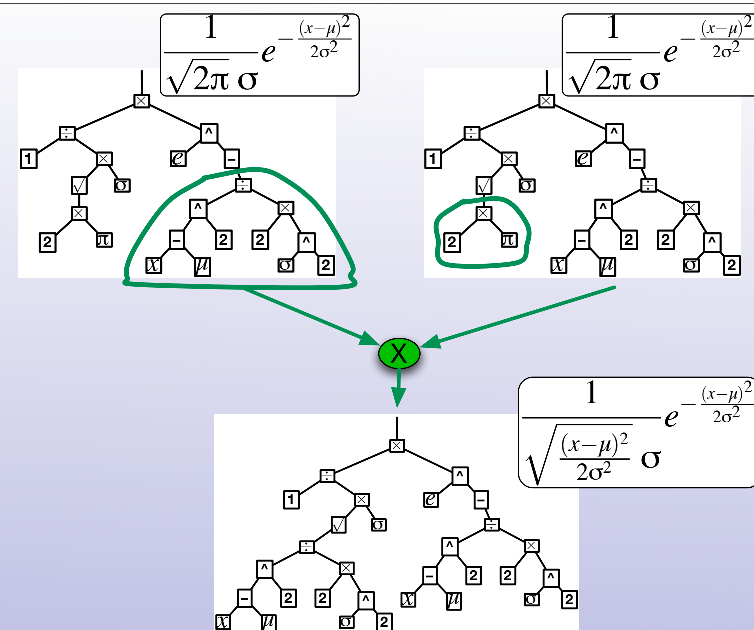


3. Variation Operators:

i.e." Genetic Operators"

Examples:

- Bit-flip mutation
- Gaussian, Cauchy, Scale-free, etc. random perturbations of real numbers
- Recombination
- Subtree exchange
- Permutations



4. Objective functions

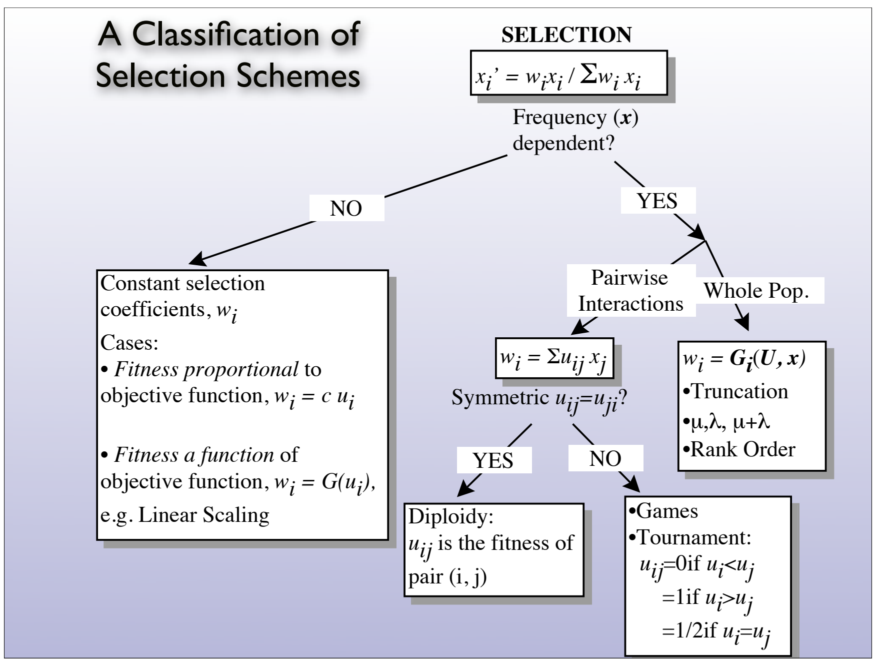
Examples:

- Length of a Hamiltonian path in a graph
- Distance between a function and a desired function
- Energy efficiency of an engine
- Folding energy of an RNA strand configuration
- Score from playing a game with other individuals

5. Selection

Examples:

- Fitness proportional to objective function
- Tournament matches
- Ranking by objective function
- Truncation below a certain objective function value
- Fertility differences for pairs of parents
- Frequency-dependent selection based on population composition



6. Populations

- Collection of multiple individuals:
  - Between which **selection** acts
  - From which **pairs** are chosen if there is bi-parental **transmission**
  - That may **interact** with **each other** or other populations if there are **game** interactions
  - Which may be subdivided with migration between subpopulations

Basic Evolutionary Algorithm:

```
population := RandomTypes();
while (stopping_criteria == unmet)
  { population_xo := Recombine(population);
  population_mut := Mutate(population_xo);
  population := Select(population_mut);
  Update(stopping_criteria);}
```


Recombination

One-point crossover:

Recombination rate = r

- For every individual:
  - with probability 1-r, offspring = parent;
  - with probability r, offspring is recombinant.
- If recombinant, pick a mate, pick a crossover point
- Offspring alleles are drawn from the parent up to crossover point, otherwise drawn from the mate.

Recombination code

One-point crossover:

```
• Recombine(population){
    for (i=1 to PopSize)
      if (RandomReal(0,1) < recombination_rate)
        offspring = population[i];
      else {
        parent1 = population[i];
        parent2 = population[RandomInt(1,PopSize)];
        XO_pt = RandomInt(0,L);
          for (locus=1 to L)
            offspring[locus] = If(locus <= XO_pt,
            parent1[locus], parent2[locus]); }
      population_xo[i] = offspring; }
```


Mutation

Mutation rate = m

- For every individual:
  - with probability 1-m, offspring = parent;
  - with probability m, offspring is mutant.
- If mutant, pick a locus, randomly choose a replacement allele at this locus

Mutation code

One way to do it:

```
• Mutate(population_xo){
    for (i=1 to PopSize)
      if (RandomReal(0,1) < mutation_rate)
        offspring = population_xo[i];
      else {
        locus = RandomInt(1,L);
        offspring[locus] = alphabet[Random(1,AlphabetSize)];
        }
      population_mut[i] = offspring; }
```


Selection

For example, Proportional Selection:

- Compute mean objective function for the population
$$\bar{w} = \sum_{i=1}^{N} w_i$$
- Sample N individuals from the weighted distribution:
$$\text{Prob}(i) = w_i/\bar{w}$$

Selection code:

```
• Select(population_sel){
    for (i=1 to PopSize) {
      fitness[i] = objective_function(population[i]);
      fitness_sum += fitness[i]; }
    for (i=1 to PopSize) {
      sampleProb[i] = fitness[i] /fitness_sum;
      CDF[i] += sampleProb[i]; }
    for (i=1 to PopSize)
      for (k=1 to PopSize)
        if (RandomReal(0,1) > CDF[k]) {
                population_sel[i] = population[k];
                break; }
    }
```

Example: “Hill Climbing”

- the “(1+1)” Evolution Strategy:
- Population size = 1.

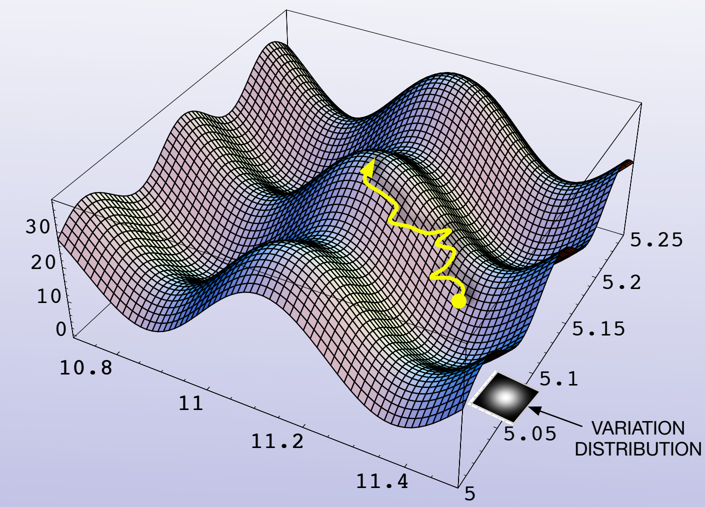
1. Generate a variant from the parent
2. Evaluate the objective function of the variant.
3. If it is better than the parent, it replaces the parent. If not, return to 1.

Why is this called "hill climbing"?

- The variation generating operator defines the types that are "neighbors"–a topology*.
- Objective function values on this *topology* define a *topography*.
- The Hill Climber climbs a hill in this topography.

*or pre-topology (Stadler et al 2000)

Hill Climbing

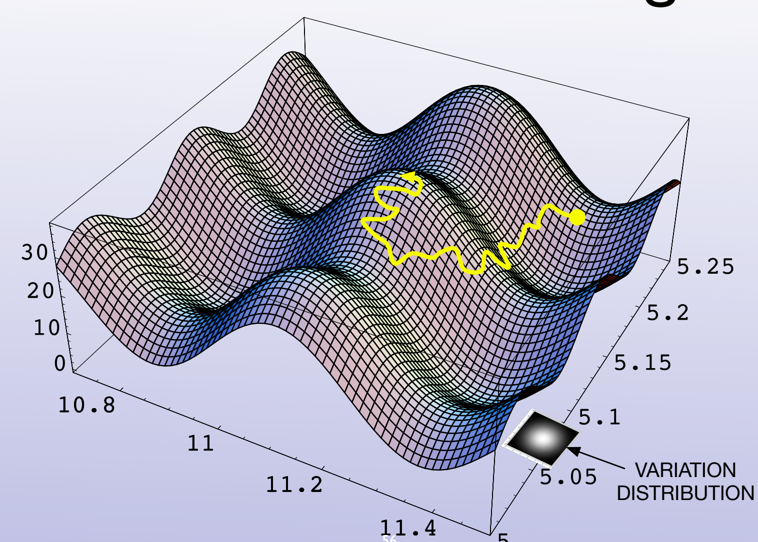


Simulated Annealing

With one modification, hill climbing becomes SIMULATED ANNEALING

1. Generate a variant x' from the parent x.
2. Evaluate the objective function of the variant, F(x').
3. If F(x') > F(x), x' replaces the parent x. If not, x' replaces the parent with probability: $e^{(F(x')-F(x))/T}$
4. Gradually lower T, the "temperature" (annealing).

Simulated Annealing



"Local Search"

- Evolutionary algorithms are often described as examples of "local" search.
- But what does "local" really mean?
  - It means that new points are sampled "nearby" points that have already been sampled.
  - But what defines "nearby"?

"Nearby"

- Extrinsic definition:
  - "nearby" means close with respect to some **metric**, like Euclidean distance or Hamming distance.
- Intrinsic definition:
  - "nearby" is whatever the variation operator picks as a variant.
- Q. What matters to the search algorithm?
  - A. The intrinsic definition.

The Adaptive Landscape metaphor

- There is often a "natural" topology with which to geometrize the search space.
- Addition of the objective function to the topology generates a "topography" – the adaptive landscape
- Rugged landscapes are seen to be difficult for "local" search.
- But what if the search operator generates a different topology from the "natural" topology?

"Massive Multimodality"

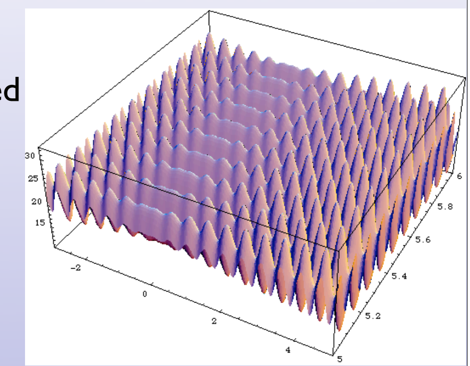
$$F(x,y) = 21.5 + x\sin(4\pi x) + y\sin(20\pi y)$$

Mutation:

$$(x,y) \rightarrow (x+\varepsilon, y+\xi)$$

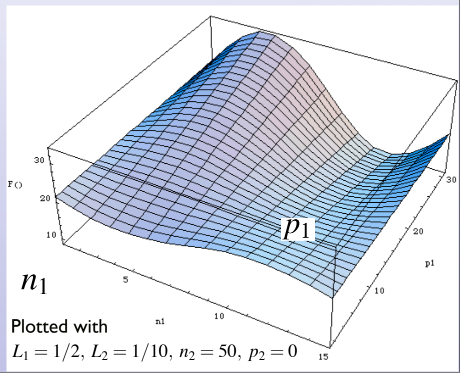
where mutation is produced by random variables distributed as:

$$f(\varepsilon) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{\varepsilon^2}{2\sigma^2}}$$
$$f(\xi) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{\xi^2}{2\sigma^2}}$$



- There is no reason the representation has to be the "natural" one.
- Rewrite the representation in terms of **phase** and **wave number**: $x = n_1 L_1 + p_1$, $y = n_2 L_2 + p_2$,
  - where $L_1 = 1/2,\ L_2 = 1/10,\ n_1, n_2 \in Z,\ p_1, p_2 \in \Re$ mod 1

The adaptive landscape becomes smooth.



Plotted with $L_1 = 1/2,\ L_2 = 1/10,\ n_2 = 50,\ p_2 = 0$

...which is equivalent to a change in the mutation operator:

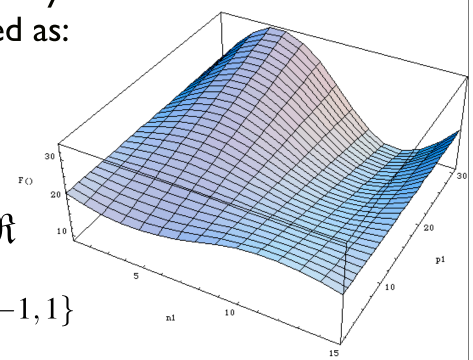
- From: $(x, y) \to (x + \varepsilon, y + \xi)$
- To: $(x, y) \to (x + \varepsilon + \nu,\ y + \xi + \mu)$

where mutation is produced by random variables distributed as:

$$f(\varepsilon) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{\varepsilon^2}{2\sigma^2}}$$

$$f(\xi) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{\xi^2}{2\sigma^2}} \quad \varepsilon, \xi \in \Re$$

$$f(\nu) = f(\mu) = 1/2 \ \text{ for } \nu, \mu \in \{-1, 1\}$$



Operator/Representation Duality:

- Changes in **representations** may be equivalent to
- changes in **genetic operators**
- in producing the same new **transmission function**

Do Evolutionary Algorithms Work?

- Often, quite well.

EVOLVING GAS TURBINE COMBUSTOR DESIGN

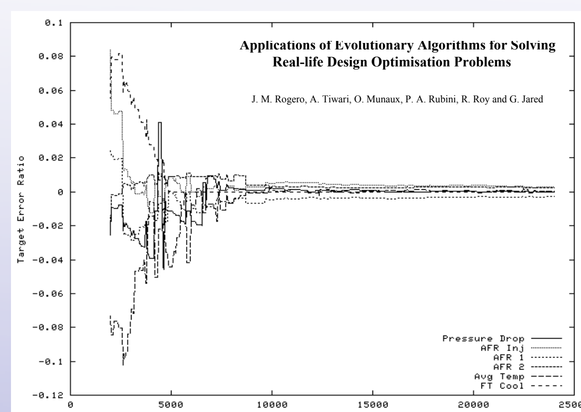


Fig1: Convergence of performance parameters toward the design targets

“As good as it gets”: the ONEMAX problem

$$F(x) = \sum_{i=1}^{L} x_i, \;\; x_i \in \{0,1\}$$

- Global optimum x=(1111111...111) found in O(L)=O(Log n) samples using bit-flip mutation

“As bad as it gets”: random search

$$F(x) = R(x)$$

- where R(x) is distributed like a random variable i.i.d.
- Global optimum found in exponential time, $O(e^L)$=O(n) samples using bit-flip mutation

For good EA performance...

Knowledge is necessary

- Wolpert and MacReady (1995), ``No Free Lunch" Theorems for search:
  - Every search algorithm has the same average performance of over all permutations of the objective function on a search space.
  - Evolutionary algorithms must *incorporate knowledge of the search space* to perform better than this average.

as our performance measure. Consequently, we are interested in the conditional probability that histogram $\vec{c}$ will be obtained under m applications of algorithm a on f. We denote this quantity $P(\vec{c}\,|\,f, m, a)$.

A major result of this work is that $P(\vec{c}\,|\,f, m, a)$ is independent of a when we average over all cost functions. In other words,

Theorem: For any pair of algorithms a_1 and a_2,

$$\sum_f P(\vec{c}\,|\,f, m, a_1) = \sum_f P(\vec{c}\,|\,f, m, a_2). \tag{1}$$

An immediate consequence of this result is that the expected histograms, $E(\vec{c}\,|\,f, m, a) = \sum_{\vec{c}} \vec{c}\, P(\vec{c}\,|\,f, m, a)$, are on average identical between any two pairs of algorithms. More generally, at the point in their search where they have both created a population of size m, the performance of any two algorithms (measured for example as the depth of the minimum found) is, on average, identical (the average being over all possible cost functions). In particular if a_1 has better performance than a_2 over some subset $\phi \subset \mathcal{F}$ of functions, then a_2 must perform better on the set of remaining functions $\mathcal{F} \setminus \phi$. So for example if simulated annealing outperforms genetic algorithms on some set ϕ, genetic algorithms must outperform simulated annealing on $\mathcal{F} \setminus \phi$.

- "How best to convert knowledge concerning *f* into an optimal *a*?

  The goal in its broadest sense is to design a system that can take in such knowledge concerning *f* and then *solve* for the optimal *a* given that knowledge."

  ---Wolpert and MacReady (1995)

How is knowledge manifest in evolutionary algorithms?

- EAs as stochastic search algorithms
  - first hitting times of optima
- EAs as dynamical systems
  - rate of convergence to attractors containing the optima
  - fraction of the space occupied by these attractors.

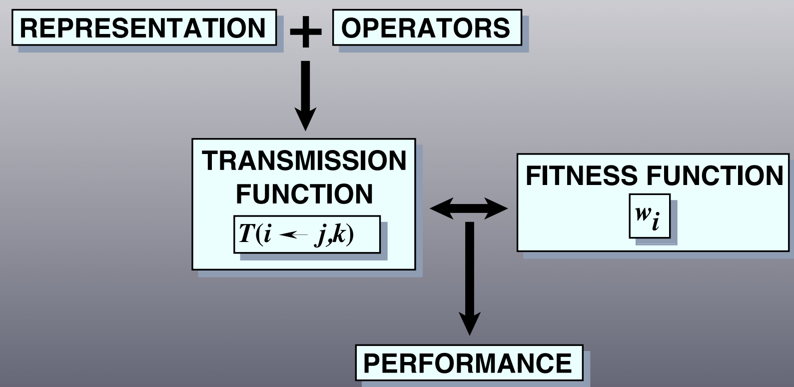
Where is the knowledge?

- In the relationship between the
  - fitness function,
  - representation, and
  - genetic operators.

No one or two of these contains the knowledge:

- The *genetic operator* acting on the *representation* produce the *transmission function*
- Knowledge is incorporated implicitly in the relation between the *transmission function* and *the fitness function.*

A Framework for Evolutionary Algorithms



- **Operators** (mutation, crossover, inversion, conversion, etc.) acting on the
- **Representations** of the search space (genotypes, strings, programs, designs, etc.), generate a
- **Transmission function**–the probability distribution of offspring types from any combination (one, two, etc., sometimes the whole population) of parent types.
- The **relationship** between the transmission function and the fitness function is
  - where *knowledge is stored* in the EA, and
  - the *main determinant* the performance of the evolutionary algorithm.

The problem of knowledge

- We usually have some knowledge about the objective function and the natural representation
- How do we convert this knowledge into representations and genetic operators that produce rapid discovery of the optima?
  - – an open question.

References

- David H. Wolpert, William G. Macready. 1995. No Free Lunch Theorems for Search.
- A. Sinclair. 1992. *Algorithms for Random Generation and Counting, A Markov Chain Approach.*
- P. Vitanyi. 2000. A discipline of evolutionary programming, *Theoret. Comp. Sci.,* 241:1-2, 3-23.
- Wright, S. 1931. Evolution in Mendelian populations. *Genetics* 16:97-159.
- Fisher, R.A. 1930. *The genetical theory of natural selection.* Oxford: Clarendon Press.