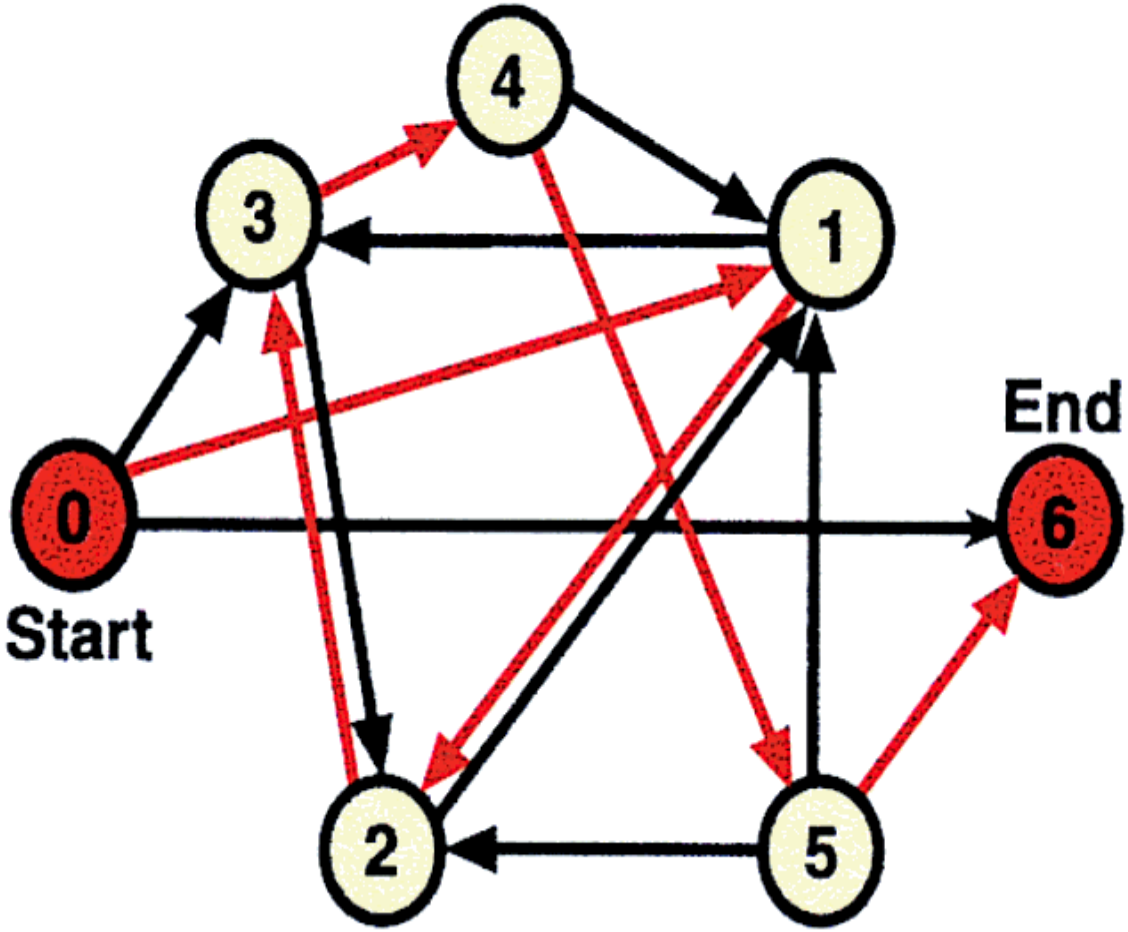
Combinatorial Optimization + Fitness Landscapes

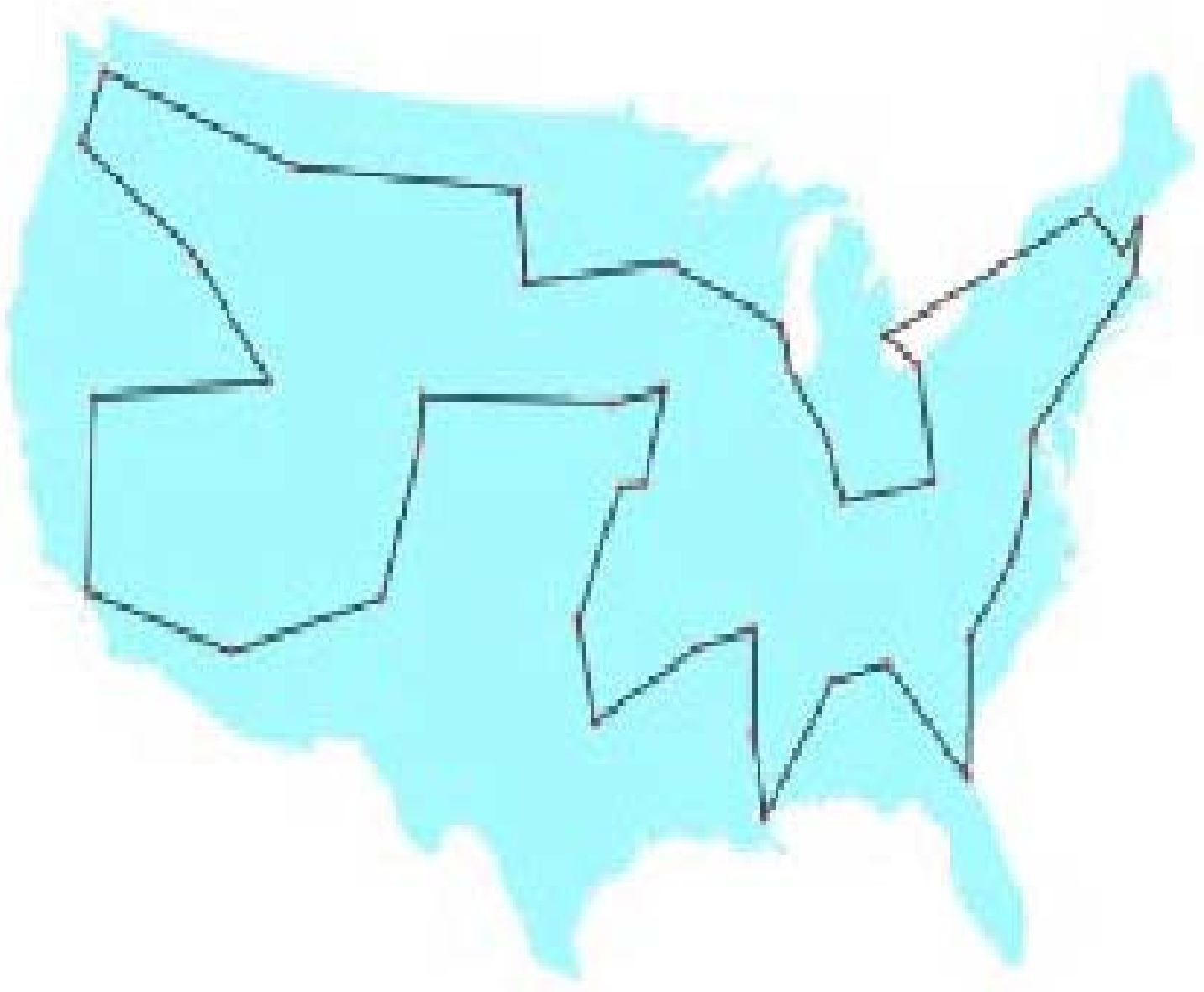
Dr. Stephan Steigele

Vorlesung im SS 2008

Bioinf / Universität Leipzig

Hamiltonian Path





The traveling salesperson problem: find the shortest tour connecting n cities?

Decision Problems

To keep things simple, we will mainly concern ourselves with decision problems. These problems only require a single bit output: ``**yes**'' and ``**no**''.

How would you solve the following decision problems?

* Is this directed graph acyclic?
* Is there a spanning tree of this undirected graph with total weight less than w?
* Does the pattern p appear as a substring in text t?

Algorithm Running Time

Given a size n problem, an algorithm runs $O(f(n))$ time:

1. $O(f(n))$: upper bound. (Ω :lower θ: equal)

2. Polynomial: $f(n)=1$(constant), n(linear), n^2(quadratic), n^k.

3. Exponential: $f(n)=2^n$, $n!$, n^n.

Time	$n=1$	$n=10$	$n=100$	$n=1000$
n	1	10	10^2	10^3
n^2	1	10^2	10^4	10^6
n^{10}	1	10^{10}	10^{20}	10^{30}
2^n	2	$>10^3$	$>10^{30}$	$>10^{300}$
$n!$	1	$>10^6$	$>10^{150}$	$>10^{2500}$

Class *P*

P is the set of **decision problems** that can be solved in worst-case polynomial time:

If the input is of size n, the running time must be $O(n^k)$.
Note that k can depend on the problem class, but not the particular instance.

All the decision problems mentioned above are in *P*.

Nice Puzzle

The **class *NP*** (meaning non-deterministic polynomial time) is the set of problems that might appear in a puzzle magazine: ``Nice puzzle."

What makes these problems special is that they might be hard to solve, but a short answer can always be printed in the back, and it is easy to see that the answer is
correct once you see it.

Class *NP*

Technically speaking:

A problem is in *NP* if it has a short accepting certificate.
An accepting certificate is something that we can use to quickly show that the answer is ``yes'' (if it is yes).
Quickly means in polynomial time.
Short means polynomial size.

This means that all problems in *P* are in *NP* (since we don't even need a certificate to quickly show the answer is ``yes'').

But other problems in *NP* may not be in *P*. Given an integer x, is it composite? How do we know this is in *NP*?

Exponential Upperbound

Another useful property of the class *NP* is that all *NP* problems can be solved in exponential time (*EXP*).

This is because we can always list out all short certificates in exponential time and check all $O(2^{nk})$ of them.

Thus, *P* is in *NP*, and *NP* is in *EXP*. Although we know that *P* is not equal to *EXP*, it is possible that *NP* = *P*, or *EXP*, or neither. Frustrating!

NP-hardness

As we will see, some problems are at least as hard to solve as any problem in NP. We call such problems NP-hard.

How might we argue that problem X is at least as hard (to within a polynomial factor) as problem Y?

If X is at least as hard as Y, how would we expect an algorithm that is able to solve X to behave?

Tractable problems: the classes P and NP

Classes of decision problems.

The class P consists of those problems that are solvable in polynomial time ($O(n^k)$ for k =constant, where n is the size of the input).

Examples:

- Given $G = (V, E)$ and $w : E \to \mathbb{R}^+$ and $k \in \mathbb{N}$, is theree a minimum spanning tree of weight $\leq k$. Jarník-Prim greedy gives a $O(n^2)$ solution ($n = |V|$).
- Max Flow Given a network G and a $k \in \int$, decide if there is a flow $|f^*| \geq k$. Edmonds-Karp gives a $O(nm)$ ($n = |V|, m = |E|$)
- The decision version of Linear Programming is in P (Karmarkar's algorithm)
- Formally sorting is not function problem, so not in P.

Algorithm Complexity

- P = solutions in polynomial deterministic time.
  - e.g. dynamic programming
- NP = (non-deterministic polynomial time)
  solutions checkable in deterministic polynomial time.
  - e.g. RSA code breaking by factoring
- NP-complete = most complex subset of NP
  - e.g. traveling all vertices with mileage < x
- NP-hard = optimization versions of above
  - e.g. Minimum mileage for traveling all vertices
- Undecidable = no way even with unlimited time & space
  - e.g. program halting problem

P and NP problems

- Assume we have a "conventional" deterministic computer.
  - The class of problems which can be solved on such a computer in polynomial time is called P (for Polynomial).
- Suppose we have a (theoretical) non-deterministic computer that can "guess" the right option when faced with choices.
  - The class of problems which can be solved on a non-deterministic computer in polynomial time is called NP (for Nondeterministic Polynomial).

Richard Feynman

"There's Plenty of Room at the Bottom"
(1959)

NP-complete problems

- The boolean satisfiability problem has been proved (by S.A. Cook (1971)) to be NP-complete.
- That is, Cook proved that if the boolean satisfiability problem (which was discussed above) is in P, then so are all problems in NP.
- Researchers have since shown that many other problems are NP-complete.
  - Proof of NP-completeness consists of mapping a given problem to the boolean satisfiability problem (or some other proven NP-complete problem).

Some NP-complete problems

- Many practical problems are NP-complete.
  - Given a linear program (a set of linear inequalities) is there an integer solution to the variables?
  - Given a set of integers, can they be divided into two sets whose sum is equal?
  - Given two identical processors, a set of tasks of varying length, and a deadline, can the tasks be scheduled so that they finish before the deadline?
  - If there is an efficient solution to any of these, then all NP problems have efficient solutions! This would have a major impact.

P=NP or P≠NP?

- Proving whether P=NP or P≠NP is one of the most important open problems in computer science.
- If someone showed that P=NP, then many "hard" problems (i.e. The NP-complete problems) would be tractable.
- However most computer scientists believe that P≠NP, largely because there are many problems which are in NP but for which no one has found an efficient solution.
  - That is, absence of evidence that P=NP counts as evidence that P≠NP.

Summary: P and NP

- Some problems seem to be intrinsically very complex (NP). The only "efficient" known solutions require a non-deterministic computer.
- At present we have no proof that such problems do not have efficient solutions (they could be in P).
- Some NP problems are significant in the sense that if they are in P, then so are all NP problems.

NP-Complete Problems

A formal-language framework:

- Alphabet Σ
- String (empty string ϵ)
- Language L over Σ (empty language $\emptyset$, Σ^*)
- Decision problem as a language $L = \{x \in \Sigma^* : Q(x) = 1\}$
- Accept, Decide (in polynomial time, by an algorithm)
  - common points
  - distinctions
- Define class P:

$$\mathrm{P} = \{L \subseteq \{0,1\}^* : L \text{ is decided by a polynomial time algorithm}\}$$

Example Proof of NP-completeness:

- Hamiltonian Cycle (HC) problem:

  Given a simple connected graph $G = (V, E)$, is G Hamiltonian?

- Traveling Salesman Problem (TSP):

  Given a simple edge-weighted complete graph $H = (U, F)$, where weights are positive integers, and an integer k, is there a Hamiltonian cycle in G such that the total weight of edges on the cycle is at most k?

Evolutionary Computation ("Genetic Algorithms")

What is Evolutionary Computation?

Example: A Genetic Algorithm:

- Works from a definition of a set ("space") of designs so that specifying a string (vector) of values (often numbers, or "yes or no" values) can completely define one design
- Starts from random "population" of solutions (designs or "chromosomes")
- Mutates some designs each generation
- Recombines some pairs of designs each generation
- Uses some analysis or simulation tool to evaluate each new design, keeps the better ones
- Quits when out of time or when no longer making progress

Genetic Algorithms:

- **Are a method of search, often applied to optimization or learning**
- **Are stochastic – but are *not* random search**
- **Use an evolutionary analogy, “survival of fittest”**
- **Not *fast* in some sense; but sometimes more robust; scale relatively well, so can be useful**

The Canonical or Classical GA

- **Maintains a set or “population” of strings at each stage**
- **Each string is called a chromosome, and encodes a “candidate solution”– CLASSICALLY, encodes as a *binary string* (but today, can be string of real numbers or almost any conceivable representation)**

Criterion for Search

- **Goodness (“fitness”) or optimality of a string’s solution determines its FUTURE influence on search process -- survival of the fittest**
- **Solutions which are good are used to generate other, similar solutions which may also be good (even better)**
- **The POPULATION at any time stores ALL we have learned about the solution, at any point**
- **Robustness (efficiency in finding good solutions in difficult searches) is key to GA success**

Classical GA: The Representation

`1011101010` – a possible 10-bit string ("CHROMOSOME") representing a possible solution to a problem

Bits or subsets of bits might represent choice of some feature, for example. Let's represent choice of shipping container for some object:

bit position	meaning
1-2	steel, aluminum, wood or cardboard
3-5	thickness (1mm-8mm)
6-7	fastening (tape, glue, rope, hinges/latches)
8	stuffing (paper or plastic "peanuts")
9	corner reinforcement (yes, no)
10	handle material (steel, plastic)

Terminology

Each position (or each set of positions that encodes some feature) is called a LOCUS (plural LOCI)

Each possible value at a locus is called an ALLELE

We need a simulator, or evaluator program, that can tell us the (probable) outcome of shipping a given object in any particular type of container

- **may be a COST (including losses from damage) (for example, maybe 1.4 means very low cost, 8.3 is very high cost on a scale of 0-10.0), or**
- **may be a FITNESS, or a number that is larger if the result is BETTER (expected net profit, for example)**

How Does a GA Operate?

- **For ANY chromosome, must be able to determine a FITNESS (measure of performance toward an objective) using a simulator or analysis tool, etc.**
- **Objective may be maximized or minimized; usually say *fitness* is to be maximized, and if objective is to be minimized, define fitness from it as something to maximize**
- **Can have one or many objectives, and possibly constraints**

GA Operators: Classical Mutation

- Operates on ONE “parent” chromosome
- Produces an “offspring” with changes.
- Classically, toggles one bit in a binary representation
- So, for example: `1101000110` could mutate to: `1111000110`
- Each bit has same probability of mutating

Classical Crossover

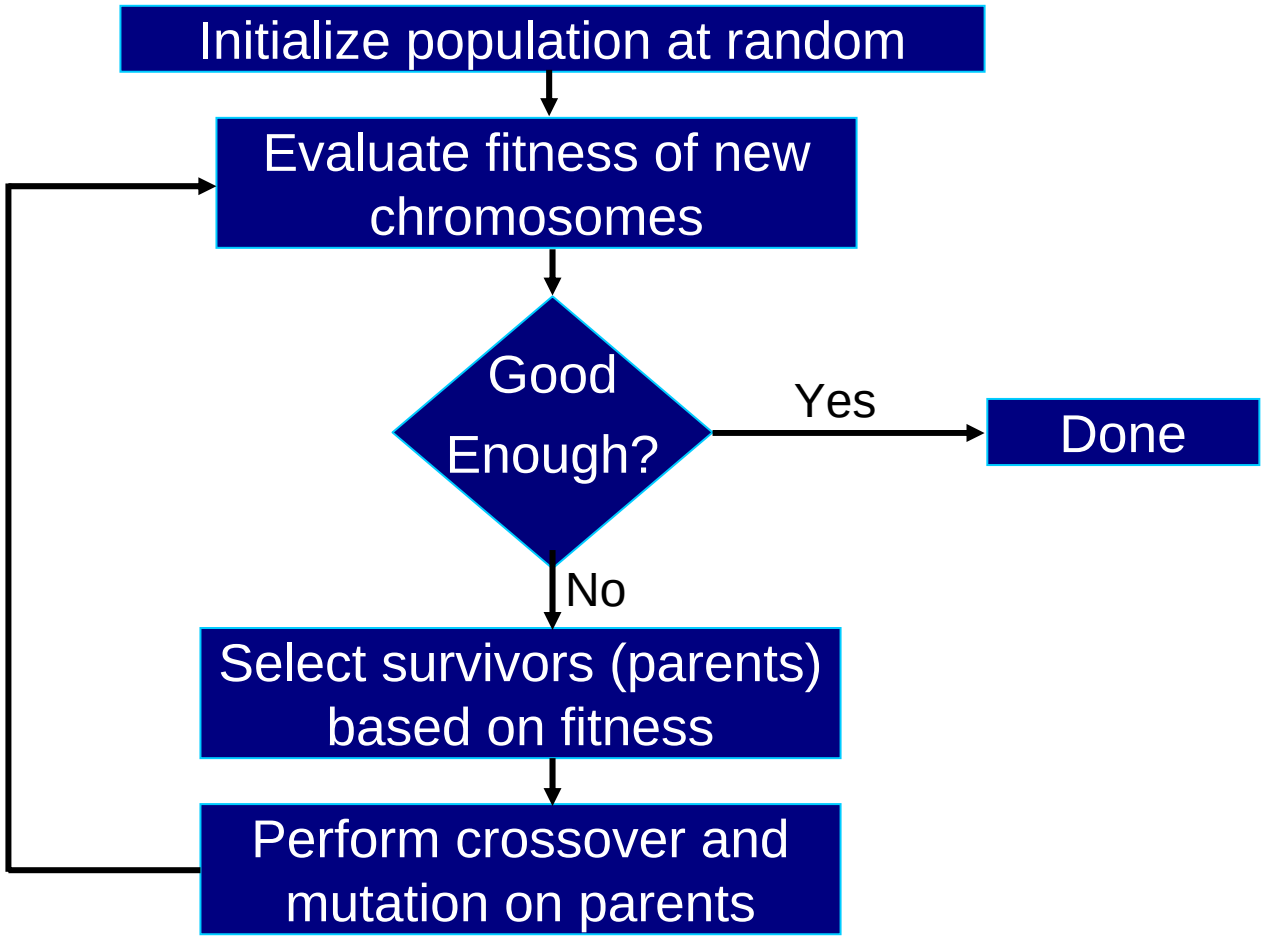
- Operates on two parent chromosomes
- Produces one or two children or offspring
- Classical crossover occurs at 1 or 2 points:
- For example: (1-point) (2-point)

```
          1111111111 or   1111111111
      X   0000000000      0000000000
          ----------      ----------
          1110000000      1110000011
and       0001111111      0001111100
```

Selection

- ***Traditionally*, parents are chosen to mate with probability proportional to their fitness: *proportional selection***
- **Traditionally, children replace their parents**
- **Many other variations now more commonly used (we'll come back to this)**
- **Overall principle: survival of the fittest**

Typical GA Operation -- Overview



Synergy – the KEY

Clearly, selection alone is no good …

Clearly, mutation alone is no good …

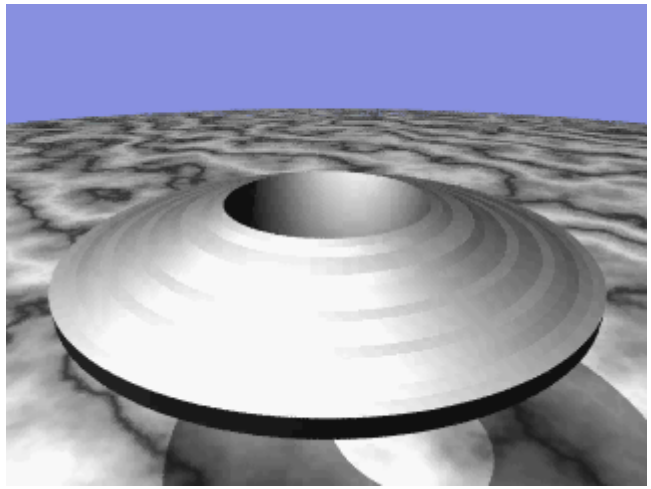
Clearly, crossover alone is no good …

Fortunately, using all three simultaneously is sometimes spectacular!

EXAMPLE!!!
Let's Design a Flywheel

GOAL: To store as much energy as possible (for a given size of flywheel) without breaking apart (think about spinning a weight at the end of a string):

- On the chromosome, a number specifies the thickness (height) of the "ring" at each given radius
- Center "hole" for a bearing is fixed
- To evaluate: simulate spinning it faster and faster until it breaks; calculate how much energy is stored just before it breaks



Flywheel Example

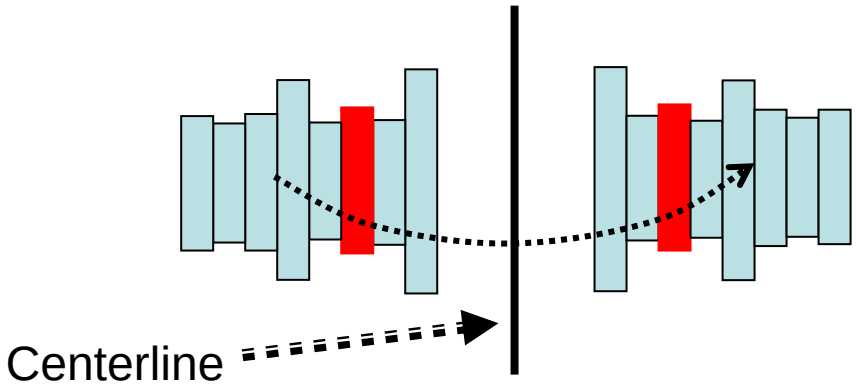
So if we use 8 rings, the chromosome might look like:

6.3 3.7 2.5 3.5 5.6 4.5 3.6 4.1

If we mutate HERE, we might get:

6.3 3.7 4.1 3.5 5.6 4.5 3.6 4.1

And that might look like (from the side):



Recombination

If we recombine two designs, we might get:

6.3	3.7	2.5	3.5	5.6	4.5	3.6	4.1
				x			
3.6	5.1	3.2	4.3	4.4	6.2	2.3	3.4
				↓			
3.6	5.1	3.2	3.5	5.6	4.5	3.6	4.1

This new design might be BETTER or WORSE!

When Might a GA Be Any Good?

- **Highly multimodal functions**
- **Discrete or discontinuous functions**
- **High-dimensionality functions, including many combinatorial ones**
- **Nonlinear dependencies on parameters (interactions among parameters) -- “epistasis” makes it hard for others**
- **Often used for approximating solutions to NP-complete combinatorial problems**
- **DON’T USE if a hill-climber, etc., will work well**

The Limits to Search

- **Efficient search must be able to EXPLOIT correlations in the search space, or it's no better than random search or enumeration**
- **Must balance with EXPLORATION, so don't just find nearest local optimum**

Representation Terminology
Genotype vs. Phenotype mapping

- **Classically, binary string: individual or chromosome**
- **What's on the chromosome is GENOTYPE**
- **What it *means* in the problem context is the PHENOTYPE (e.g., binary sequence may map to integers or reals, or order of execution, or inputs to a simulator, etc.)**
- **Genotype determines phenotype, but phenotype may *look* very different**

Discretization – Representation Meets Mutation!

- **If problem is binary decisions, bit-flip mutation is fine**
- **BUT if using binary numbers to encode integers, as in [0,15] → [0000, 1111], problem with Hamming cliffs:**
  - **One mutation can change 6 to 7: 0110 → 0111, BUT**
  - **Need 4 bit-flips to change 7 to 8: 0111 → 1000**
  - **That's called a "Hamming cliff"**
- **May use Gray (or other distance-one) codes to improve properties of operators: for example: 000, 001, 011, 010, 110, 111, 101, 100**

Defining Objective/Fitness Functions

- **Problem-specific, of course**
- **Many involve using a simulator**
- **Don’t need to know (or even HAVE) derivatives**
- **May be stochastic**
- **Need to evaluate thousands of times, so can’t be TOO COSTLY**
- **For real-world, evaluation time is typical bottleneck**

Selection

In a classical, "generational" GA:

- **Based on fitness, choose the set of individuals (the *"intermediate"* population) that will soon:**
  - **survive untouched, or**
  - **be mutated, replaced, or**
  - **in pairs, be crossed over and possibly mutated, with offspring replacing parents**

  One individual may appear several times in the intermediate population (or the next population)

Types of Selection

- **“roulette wheel” -- classical Holland -- chunk of wheel ~ *relative* fitness**
- **stochastic uniform sampling -- better sampling -- integer parts GUARANTEED; still proportional**

Explaining Why a GA Works – Intro to GA Theory

- **Just touching the surface with two classical results:**
  - **Schema theorem – how search effort is allocated**
  - **Implicit parallelism – each evaluation provides information on many possible candidate solutions**

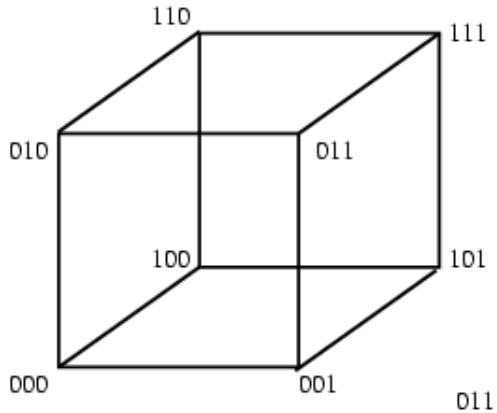
What is a GA DOING? (Schemata and Hyperstuff)

- **Schema -- adds "*", means "don't care"**
- **One schema, two schemata**
- **Definition: *ORDER* of schema H = o(H): # of non-*'s**
- **Def.: *Defining Length* of schema, $\Delta(H)$: distance *between* first and last non-* in a schema; for example:**

  Δ (1*01*0**) = 5 (= number of positions where 1-pt crossover can disrupt it).**

  (NOTE: diff. xover → diff. relationship to defining length)
- **Strings or chromosomes are order L schemata, where L is length of chromosome (in bits or loci). Chromosomes are *INSTANCES* (or members) of lower-order schemata**

Cube and Hypercube



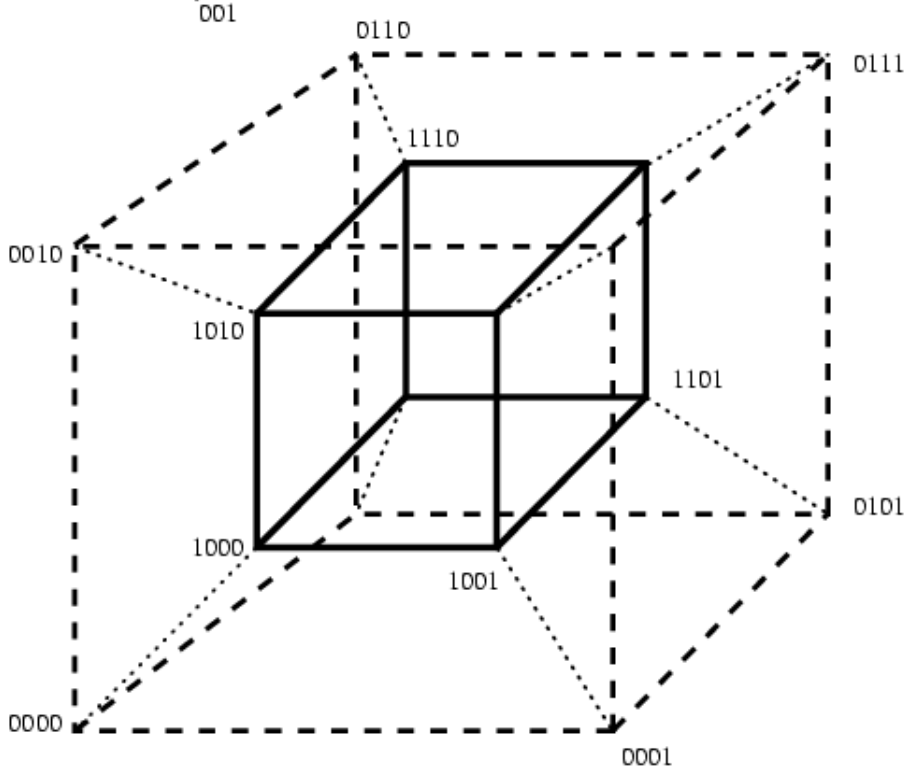
Vertices are order ? schemata

Edges are order ? schemata

Planes are order ? schemata

Cubes (a type of hyperplane) are order ? schemata

8 different order-1 schemata (cubes): 0***, 1***, *0**, *1**, **0*, **1*, ***0, ***1



Hypercubes, Hyperplanes, Etc.

- **A *string* is an instance of how many schemata (a member of how many hyperplane partitions)? (not counting the "all *'s," per Holland)**
- **If L=3, then, for example, 111 is an instance of how many (and which) schemata: 7 schemata**
- **2^3-1**

GA Sampling of Hyperplanes

So, in general, string of length L is an instance of 2^L-1 schemata

But how many schemata are there in the whole search space?

(how many choices each locus?)

Since one string instances 2^L-1 schemata, how much does a population tell us about schemata of various orders?

Implicit parallelism: **one string's fitness tells us something about relative fitnesses of more than one schema.**

Fitness and Schema/ Hyperplane Sampling

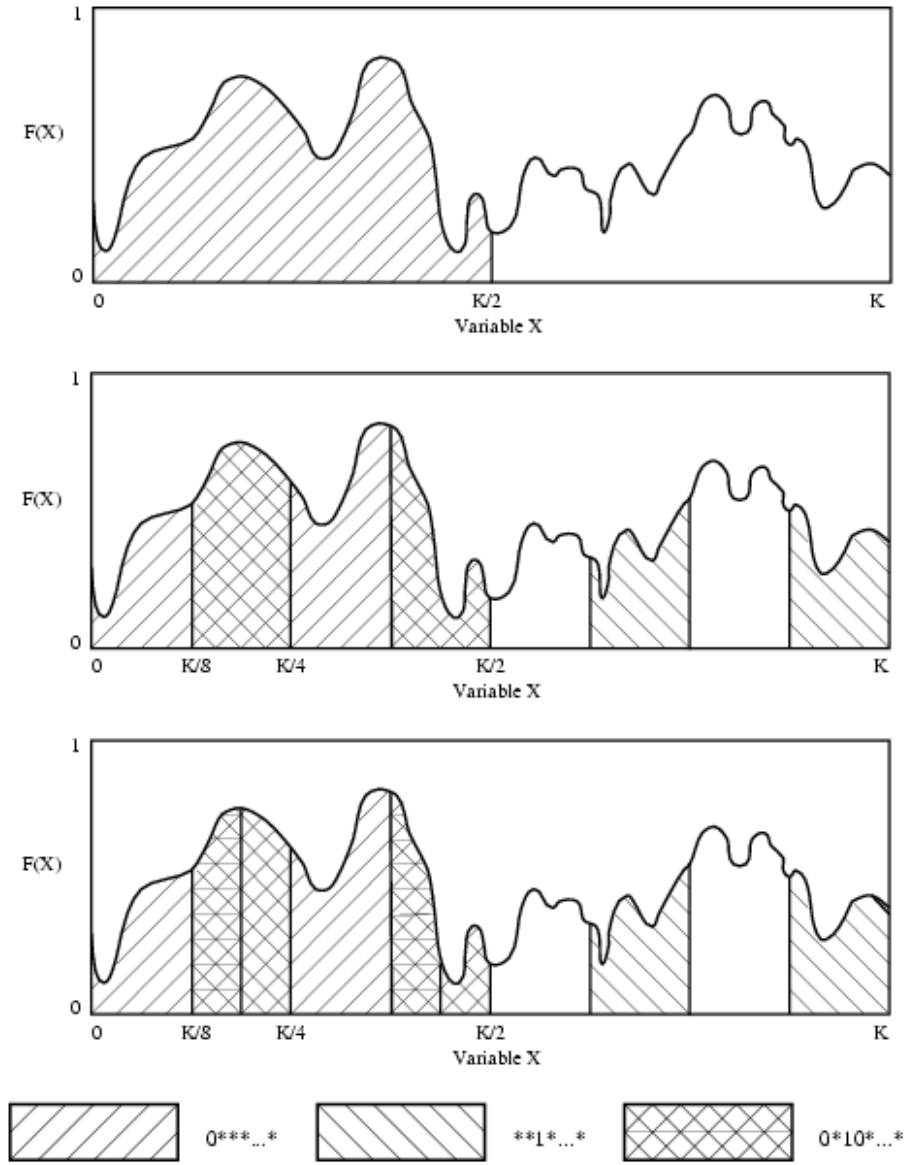
Whitley's illustration of various partitions of fitness hyperspace

Plot fitness versus *one variable* discretized as a K = 4-bit binary number: then get ➔

First graph shades 0*

Second superimposes **1*, so crosshatches are ?

Third superimposes 0*10



How Do Schemata Propagate?

- **Via *instances* -- only STRINGS appear in pop – you'll never actually see a schema**
- **But, in general, want schemata whose instances have higher average fitnesses (even just in the current population in which they're instanced) to get more chance to reproduce. That's how we make the fittest survive!**

Proportional Selection Favors "Better" Schemata

- **Select the INTERMEDIATE population, the "parents" of the next generation, via fitness-proportional selection**
- **Let *M(H,t)* be number of instances (samples) of schema H in population at time t. Then fitness-proportional selection yields an expectation of:**

$$M(H, t + intermed) = M(H, t) \frac{f(H, t)}{\overline{f}}$$

- **In an example, actual number of instances of schemata (next page) in intermediate generation tracked expected number pretty well, in spite of small pop size**

Schemata and Fitness Values									
Schema	Mean	Count	Expect	Obs		Schema	Mean	Count	Expect	Obs
101*...*	1.70	2	3.4	3		*0**...*	0.991	11	10.9	9
111*...*	1.70	2	3.4	4		00**...*	0.967	6	5.8	4
1*1*...*	1.70	4	6.8	7		0***...*	0.933	12	11.2	10
01...*	1.38	5	6.9	6		011*...*	0.900	3	2.7	4
**1*...*	1.30	10	13.0	14		010*...*	0.900	3	2.7	2
11...*	1.22	5	6.1	8		01**...*	0.900	6	5.4	6
11**...*	1.175	4	4.7	6		0*0*...*	0.833	6	5.0	3
001*...*	1.166	3	3.5	3		*10*...*	0.800	5	4.0	4
1***...*	1.089	9	9.8	11		000*...*	0.767	3	2.3	1
0*1*...*	1.033	6	6.2	7		**0*...*	0.727	11	8.0	7
10**...*	1.020	5	5.1	5		*00*...*	0.667	6	4.0	3
*1**...*	1.010	10	10.1	12		110*...*	0.650	2	1.3	2
****...*	1.000	21	21.0	21		1*0*...*	0.600	5	3.0	4
						100*...*	0.566	3	1.70	2

Results of example run (Whitley) showing that observed numbers of instances of schemata track expected numbers pretty well

Now, What Does CROSSOVER Do to Schemata

- **One-point Crossover Examples (blackboard)**
  11****** and 1********1**
- **Two-point Crossover Examples (blackboard)**
  (rings)
- **Closer together loci are, less likely to be disrupted by crossover. A “compact representation” tends to keep alleles together under a given form of crossover (minimizes probability of disruption).**

Linkage and Defining Length

- **Linkage -- “coadapted alleles” (generalization of a *compact representation* with respect to schemata)**
- **Example, convincing you that probability of disruption by 1-point crossover of schema H of length Δ(H) is Δ(H)/(L-1):**

  1**01**1**

The Fundamental Theorem of Genetic Algorithms -- "The" Schema Theorem

Holland published in ANAS in 1975, had taught it much earlier

It provides *lower bound* on change in sampling rate of a single schema from generation t to t+1. We'll consider it in several steps, starting from the change caused by selection alone:

$$M(H, t + intermed) = M(H, t)\frac{f(H,t)}{\bar{f}}$$

Schema Theorem Derivation (cont.)

Now we want to add effect of crossover:

Conservative assumption: crossover within the defining length of H is always disruptive to H, and will ignore gains (we're after a LOWER bound -- won't be as tight, but simpler). Then:

$$M(H,t+1) \geq (1-p_c)M(H,t)\frac{f(H,t)}{\overline{f}} + p_c[M(H,t)\frac{f(H,t)}{\overline{f}}(1-disruptions)]$$

Schema Theorem Derivation (cont.)

Whitley adds a *non*-disruption case that Holland ignored:

prob. of disruption by x-over is:

$$\frac{\Delta(H)}{L-1}(1-P(H,t))$$

Then can simplify the inequality, dividing by popsize and rearranging re p_c:

$$P(H,t+1) \geq P(H,t)\frac{f(H,t)}{\bar{f}}[1-p_c\frac{\Delta(H)}{L-1}(1-P(H,t))]$$

So far, we have ignored mutation and assumed second parent is chosen at random. But it's interesting, already.

Schema Theorem Derivation (cont.)

Now, we'll choose the second parent based on fitness, too:

$$P(H,t+1) \geq P(H,t)\frac{f(H,t)}{\overline{f}}[1-p_c\frac{\Delta(H)}{L-1}(1-P(H,t)\frac{f(H,t)}{\overline{f}})]$$

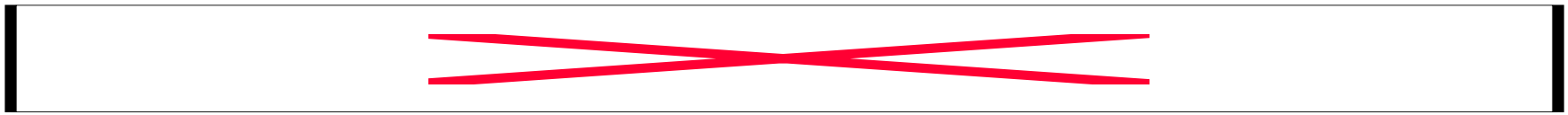
Now, add effect of mutation. What is probability that a mutation affects schema H? (Assuming mutation always flips bit or changes allele):

Each fixed bit of schema (o(H) of them) changes with probability p_m, so they ALL stay UNCHANGED with probability:

$$(1-p_m)^{o(H)}$$

Schema Theorem Derivation (cont.)

Now we have a more comprehensive schema theorem:



People often use Holland's earlier, simpler, but less accurate bound, first approximating the mutation loss factor as (1-*o*(H)p_m), assuming p_m<<1.

Schema Theorem Derivation (cont.)

That yields:

$$P(H,t+1) \geq P(H,t)\frac{f(H,t)}{\overline{f}}[1-p_c\frac{\Delta(H)}{L-1}][1-o(H)p_m]$$

But, since p_m<<1, we can ignore small cross-product terms and get:

$$P(H,t+1) \geq P(H,t)\frac{f(H,t)}{\overline{f}}[1-p_c\frac{\Delta(H)}{L-1}-o(H)p_m]$$

That is what many people recognize as the "classical" form of the schema theorem.

What does it tell us?

Using the Schema Theorem

Even a simple form helps balance initial selection pressure, crossover & mutation rates, etc.:

$$P(H,t+1) \geq P(H,t)\frac{f(H,t)}{\overline{f}}[1-p_c\frac{\Delta(H)}{L-1}-o(H)p_m]$$

Say relative fitness of H is 1.2, p_c = .5, p_m = .05 and L = 20: What happens to H, if H is long? Short? High order? Low order?

Building Block Hypothesis

Define a *Building block* as: a short, low-order, high-fitness schema

BB Hypothesis: "Short, low-order, and highly fit schemata are sampled, recombined, and resampled to form strings of potentially higher fitness… we construct better and better strings from the best partial solutions of the past samplings."

-- David Goldberg, 1989

(GA's can be good at assembling BB's, but GA's are also useful for many problems for which BB's are not available)

Traditional Ways to Do GA Search…

- **Population “large”**
- **Mutation rate (per locus) ~ 1/L**
- **Crossover rate moderate (<0.3) or high (per DeJong, .7, or up to 1.0)**
- **Selection scaled (or rank/tournament, etc.) such that Schema Theorem allows new BB’s to grow in number, but not lead to premature convergence**

The N^3 Argument (Implicit or Intrinsic Parallelism)

Assertion: A GA with pop size N can usefully process on the order of N^3 hyperplanes (schemata) in a generation.

(WOW! If N=100, N^3 = 1 million)

***To elaborate, assume*:**

- **Random population of size N.**
- **Need ϕ instances of a schema to claim we are "processing" it in a statistically significant way in one generation.**

The N^3 Argument (cont.)

Instead of general case, Fitzpatrick & Grefenstette argued:

- **Assume** $L \geq 64 \text{ and } 2^6 \leq N \leq 2^{20}$
- **Pick ϕ=8, which implies** $3 \leq \theta \leq 17$
- **By inspection (plug in N's, get θ's, etc.), the number of schemata processed is greater than N^3. For example, N=64, # schemata order 3 or less is > 2**61 > 64**3 = 2**18 = 256K.**
- **So, as long as our population size is REASONABLE (64 to a million) and L is large enough (problem hard enough), the argument holds.**
- **But this deals with the initial population, and it does not necessarily hold for the latter stages of evolution. Still, it may help to explain why GA's can work so well…**

Exponentially Increasing Sampling and the K-Armed Bandit Problem

Question: How much sampling should above-average schemata get?

Holland showed, subject to some conditions, using analysis of problem of allocating choices to maximize reward returned from slot machines ("K-Armed Bandit Problem") that:

- **Should allocate an exponentially increasing fraction of trials to above-average schemata**
- **The schema theorem says that, with careful choice of population size, fitness measure, crossover and mutation rates, a GA can do that:**
- **(Schema Theorem says M(H,t+1) >= k M(H,t))**

  That is, H's instances in population grow exponentially, as long as small relative to pop size and k>1 (H is a "building block").

Crowding

Crowding (DeJong) helps form “niches” and reduce premature takeover by fit individuals

For each child:

- **Pick K candidates for replacement, at random, from intermediate population**
- **Calculate pseudo-Hamming distance from child to each**
- **Replace individual most similar to child**

Effect?

Crossover Operators for Permutation Problems

What properties do we want:

- **1) Want each child to combine building blocks from both parents in a way that preserves high-order *schemata* in as meaningful a way as possible, and**
- **2) Want all solutions generated to be feasible solutions.**

Operators for Permutation-Based Representations, Using TSP Problem:

Example: PMX -- Partially Matched Crossover

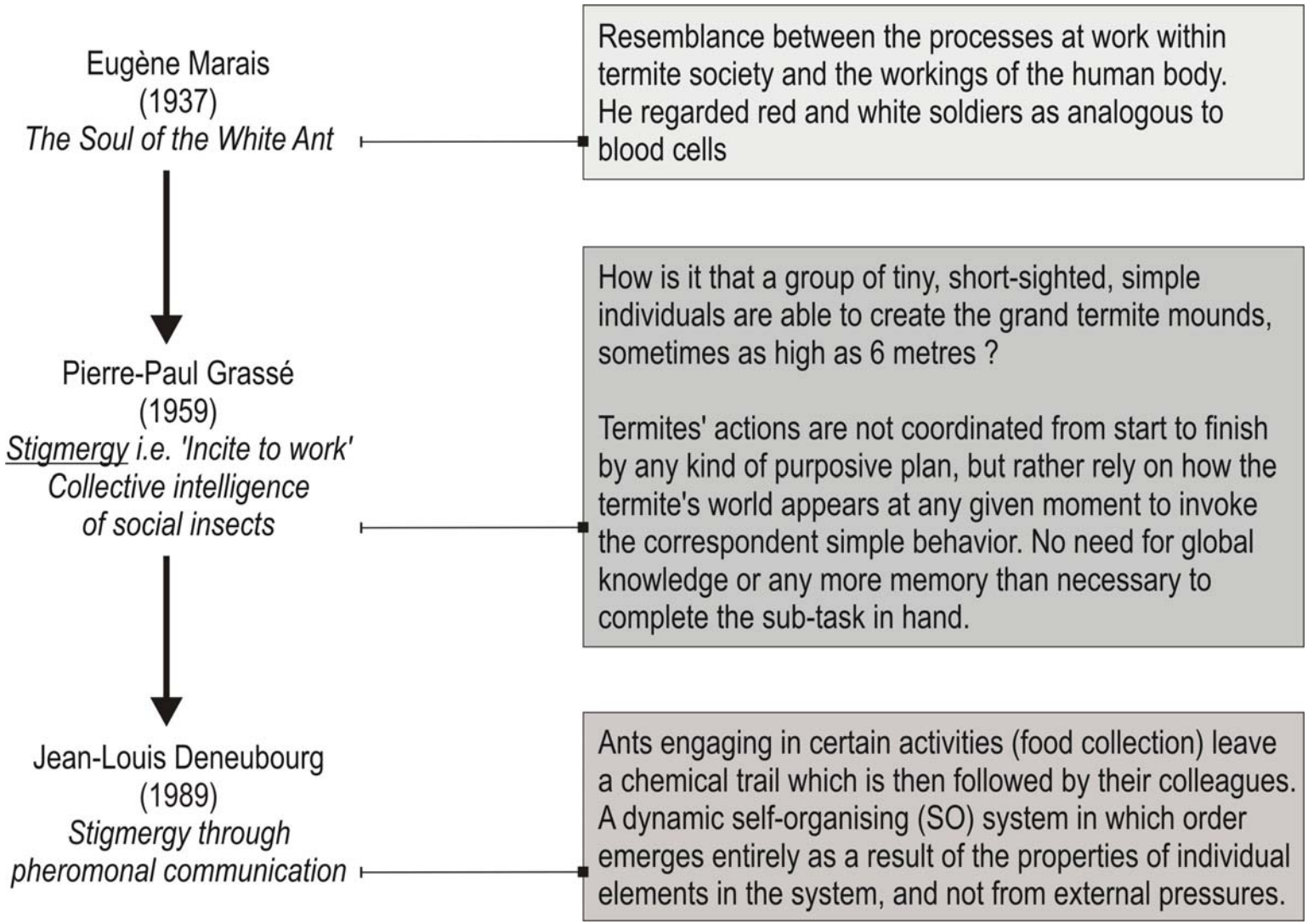
- 2 sites picked, intervening section specifies "cities" to interchange between parents:
- A = 9 8 4 | 5 6 7 | 1 3 2 10
- B = 8 7 1 | 2 3 10 | 9 5 4 6
- A' = 9 8 4 | 2 3 10 | 1 6 5 7
- B' = 8 10 1 | 5 6 7 | 9 2 4 3

- (i.e., swap 5 with 2, 6 with 3, and 7 with 10 in both children.)
- Thus, some ordering information from each parent is preserved, and no infeasible solutions are generated
- Only one of many specialized operators developed

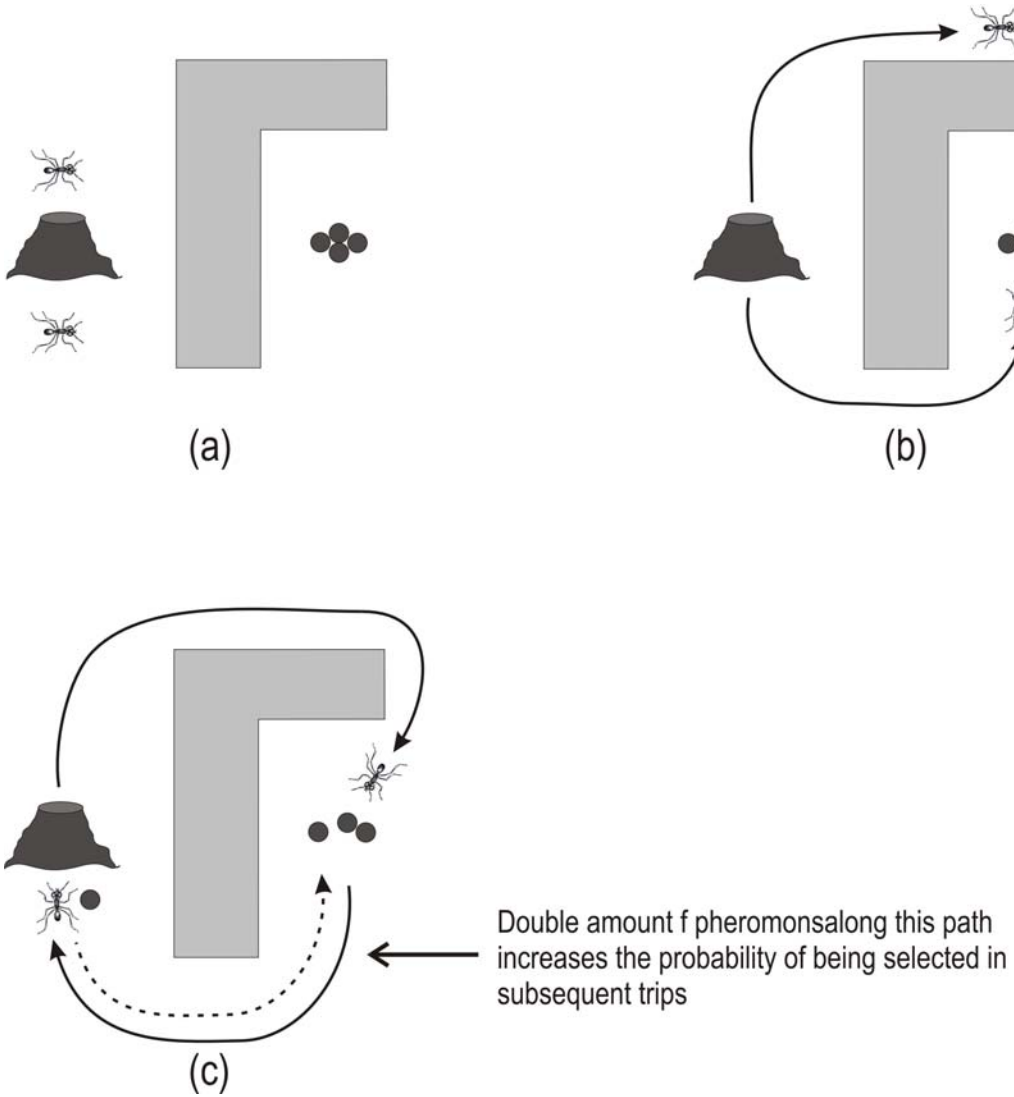
What is Ant Algorithms

- Ant optimisation algorithms [Dorigo 1996] are multi-agent systems, which consist of agents with the collective behavior (stigmergy) of ants for finding shortest paths
  - Alternative to applying complex algorithms to static datasets
  - A set of artificial ants implement a simple algorithm collectively to solve a combinatorial problem by a cooperative effort
  - Originated from Alife research

A Brief History - Timeline

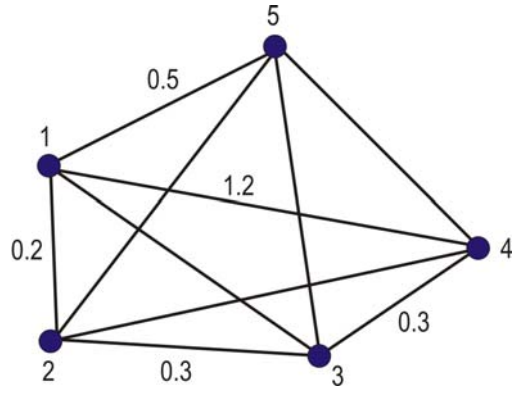


Deneubourg's Simple Experiment



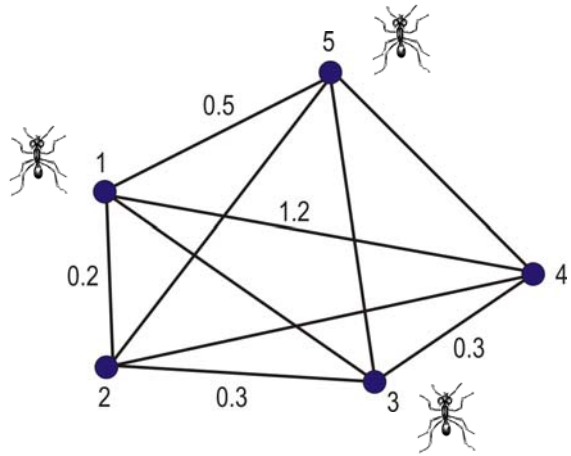
Terminology Index

- Node: a vertice
- Edge: a line connecting 2 vertices
- Graph: Diagram connecting nodes with edges
- Weighted graph: Edges have weights
- Path: Sequence of nodes-edges from between two nodes
- Hamiltonian path: A path without any node revisited
- Sigma: a sum of terms
- Delta: a difference between terms



Computational Model

- The environment
  - Weighted graph representing distances between nodes
- The ants
  - Init population randomly distributed
  - Travel across the graph
  - Ordered tabu list of visited nodes
  - Follow a Hamiltonian path



$$P = \frac{\tau(r,u)^{\alpha} * \eta(r,u)^{\beta}}{\sum_{K} \tau(r,u)^{\alpha} * \eta(r,u)^{\beta}}$$

Computational Model

- Ant Tour
  - Hamiltonian path across all nodes in graph
  - Path length L computed based on distances η
  - Pheromone left on each node in the path
    - Short tour → High pheromone
    - Long Tour → Low Pheromone

$$\Delta\tau_{ij}^{k}(t) = \frac{Q}{L^{k}(t)}$$

  - Pheromone increase at the end of tour

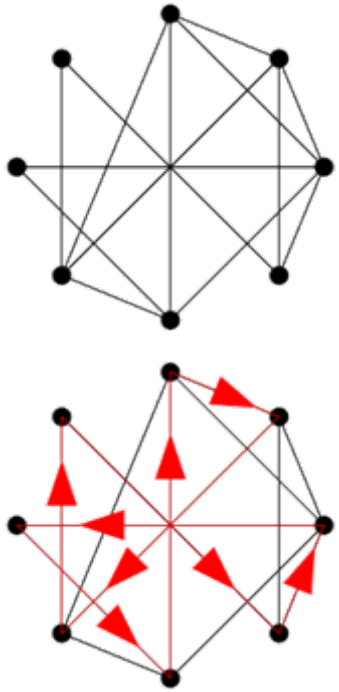
$$\tau_{ij}(t) = \tau_{ij}(t) + (\Delta\tau_{ij}^{k}(t) * \rho)$$

- Evaporation
  - Remove edges of poor paths

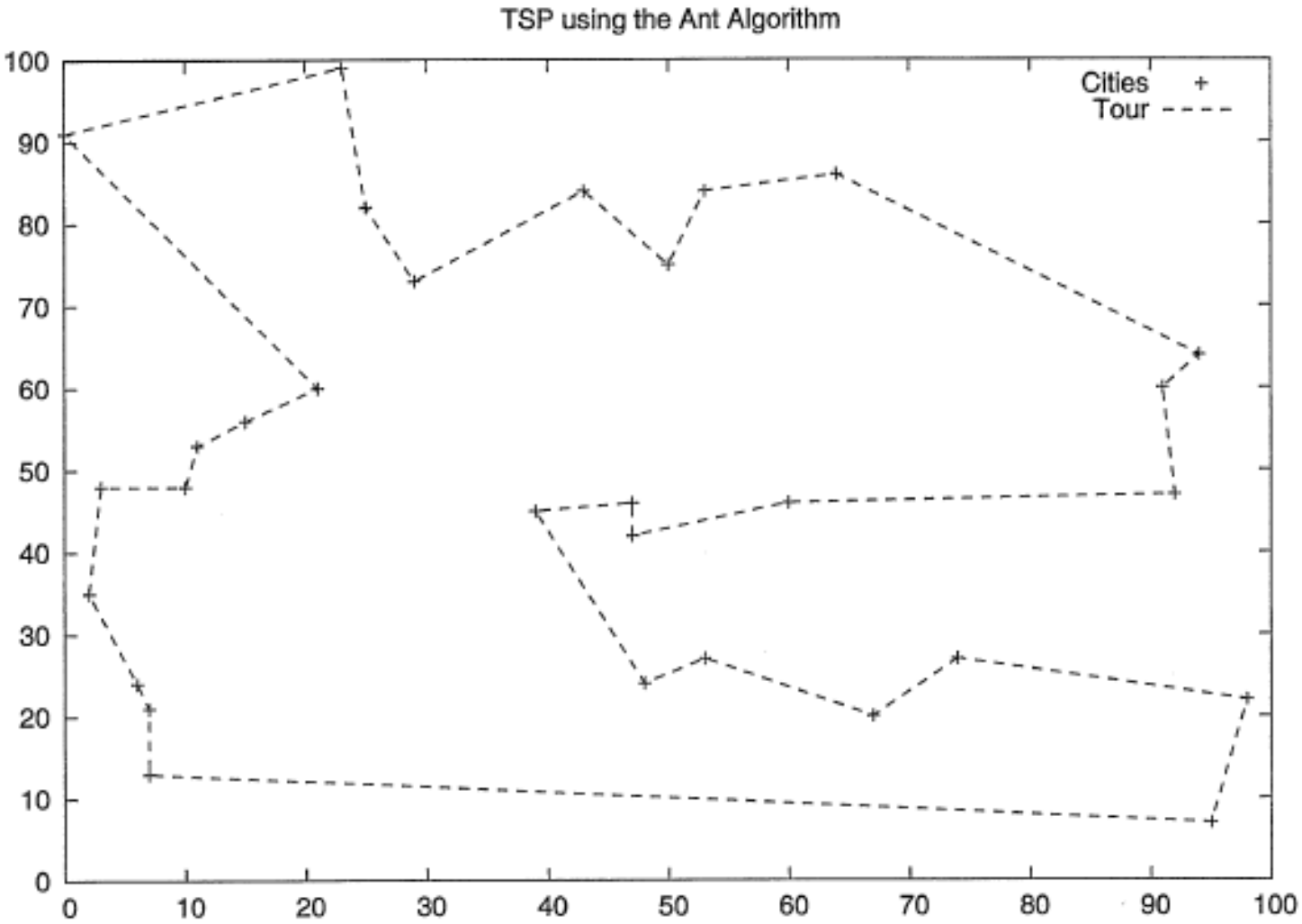
$$\tau_{ij}(t) = \tau_{ij}(t) * (1 - \rho)$$

Application: Travelling Salesman Problem

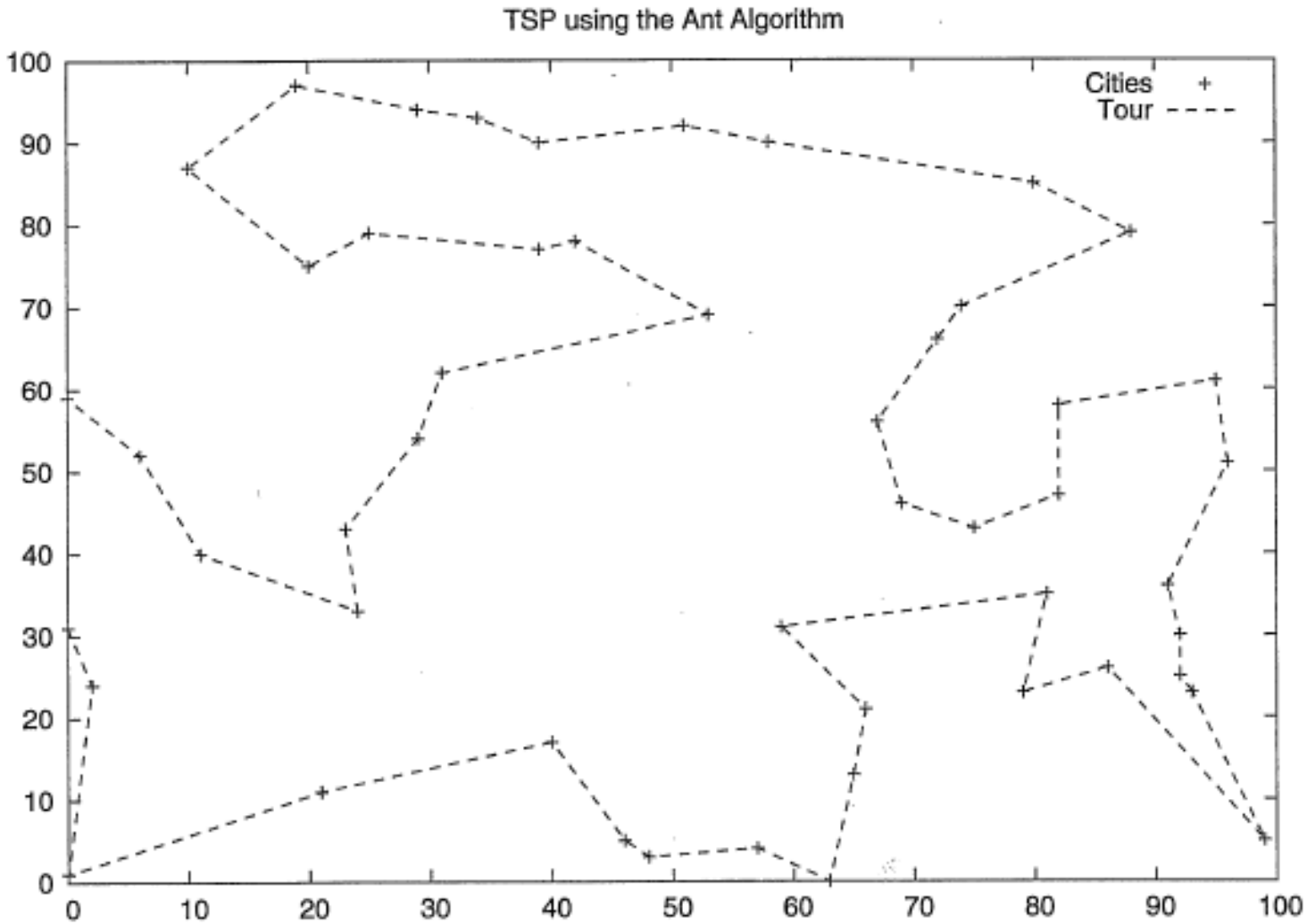
- A set of cities. A salesman needs to travel through all the cities following an optimal route (Hamiltonian path) that minimises the distance travelled.
  - Fist studied in 1930s
  - NP-hard: Not been found an algorithm that solves the general problem (for any number of cities at any arrangement), in polynomial time.
  - (Sub-)optimal solutions are possible for specific instances of the problem



TSP: Sample run (30 cities)



TSP: Sample run (50 cities)



Hillclimbing, Simulated annealing, and GA

A nice (and very interesting) comparison

Notice that in all [hill-climbing] methods discussed so far, the kangaroo can hope at best to find the top of a mountain close to where he starts. There's no guarantee that this mountain will be Everest, or even a very high mountain. Various methods are used to try to find the actual global optimum.

In simulated annealing, the kangaroo is drunk and hops around randomly for a long time. However, he gradually sobers up and tends to hop up hill.

In genetic algorithms, there are lots of kangaroos that are parachuted into the Himalayas (if the pilot didn't get lost) at random places. These kangaroos do not know that they are supposed to be looking for the top of Mt. Everest. However, every few years, you shoot the kangaroos at low altitudes and hope the ones that are left will be fruitful and multiply".

Adaptation

- Feedback process in which external changes in an environment are mirrored by compensatory internal changes in a system.
- Simple case: thermostat
- Interesting case : adaptation in complex systems
  - Actions of the adaptive unit can affect the environment, which, in turn, feeds the information back into the adaptive system.
  - Adaptation can be seen as a computation of the most complex form that emerges through the multiplicity and recursion of simple subunits.

Genetics and Evolution

- The predictive and explanatory power of evolution and natural selection has shed light on every facet of biology
  - Microscopic scale of how bacteria quickly adapt and become resistant to new drugs
  - Macroscopic scale of the distribution and interrelatedness of whole species.
- There are many details behind evolution that are still a mystery.

Biological adaptation

- Adaptation = variation + heredity + selection

Variation

- Refers to how individual can differ from each other.
- Evolution operates on no single individual but on entire species.
- Can be expressed only in terms of multiple individuals
- Parallelism and multiplicity are essential ingredients in the algorithm of evolution.

Heredity

- A form of temporal persistence
- Traits are inherited in discrete chunks of information
- Traits are iteratively passed down a time line.
- Parallelism and iteration as fundamental pieces of the biological equation for adaptation.

Selection (1/2)

- With limitation on available resources, reproduction is far from a sure thing.
- You and I can proudly make the claim that every one of our ancestors, without exception, survived long enough to reproduce.
- If we consider the number of organisms that did not survive long enough to reproduce, we can be seen as members of a truly exclusive club.

Selection (2/2)

- Our ancestors may have been strong, fast, clever, or even sexy.
- But what really counts in natural selection is an organism's ability to reproduce.
- In fact, fitness can often be associated with a trait that may actually decrease the functionality of an organism.
  - Peacock's gaudy tail feathers may decrease a peacock's ability in the daily business of survival, but are selective for survival solely because peahens find them sexy.

Coevolution

- Two species mutually adapt to one another in such a way to have a circular relationship, with one species' influence on another ultimately returning to the first species in a feedback loop.
- Coevolution of predator and prey
  - Lions and gazelles
  - Lions become victims of their own success.
- Coevolution of bats and moths

Bats and moths

- Bats have evolved a technique to filter out the most powerful sounds so that they can concentrate on the faint return signals.
- Moths have evolved a soft covering on their bodies and wings.
- Bats have evolved new frequencies that can identify the moths' fuzzy coating.
- Moths have come up with a jamming technique that emits their own sounds.
- Bats have evolved an elaborate flight pattern that can overwhelm a moth's senses and periodically turn off their echolocation.

Adaptation

- Every species partially molds its own environment, which makes the boundary between the selector and selected somewhat blurry.
- Earth as a whole may be best understood and appreciated as one enormous complex adaptive system.

Heredity

- Neo-Darwinist view of biological adaptation differs from Darwin's original formulation in that it includes a method of heredity.
- Gregor Mendel
  - Austrian monk and amateur botanist
  - Showed that traits are inherited in discrete chunks of information.
  - Refuted the idea that traits are merged or blended.
    - Offspring from the paired tall and short pea plants are either tall or short, not medium.

Molecular biology

- In the 1950s, Mendelian genetics was further solidified by work in molecular biology.
  - Nucleic acid
  - DNA triplets code amino acids that form proteins, the building blocks for everything from digestive acids, skin, bone, eyes, blood, and brain.
  - DNA consists of extremely long chains of chemical bases, denoted by A, C, G, T.
  - Each base has a complement, double helix, etc..
- The crucial part of all of this is that the language of life has a discrete alphabet.

Why does evolution produce increasingly complex structures and organisms?

- Stephen J. Gould has proposed that evolution tends to fill niches.
  - With a world consisting entirely of some simple organisms, say bacteria, where are only so few innovation that can be made that improve fitness yet maintain the defining features of what it means to be a bacterium.
  - As the number of bacteria increases, it becomes easier to make a living as something that bacteria have not already mastered.
  - The major innovations in nature are the consequences of species trying to find new room to grow.

Unit of Selection

- Richard Dawkins has persuasively argued that a surprisingly large number of biological phenomena can be accounted for as artifacts of the gene being the sole unit of selection.
  - Altruistic animal behavior
    - Bees that sacrifice themselves for the benefit of the hive
    - Birds that warn of predators
    - Gazelles that do an eccentric dance to distract lions
  - A gene actually improves its long-term survival rate if it encourages self-sacrifice for the benefit of other relatives that share the same gene.

Natural selection as an explanation of evolution in non-biological systems

- Richard Dawkins has coined the term ***meme*** to stand for a unit of cultural information.
  - Examples of memes are tunes, ideas, catch-phrases, clothes fashions, ways of making pots, or building arches. Just as genes propagate themselves in the gene pool by leaping from body to body via sperms or eggs, so memes propagate themselves in the meme pool by leaping from brain to brain via a process which can be called imitation.

Lamarckism (1/2)

- Central dogma of biology
  - Genetic information flows in one direction, that is, DNA codes protein but protein does not code DNA.
- Lamarckism would suggest that a giraffe can endow its offspring with a longer neck solely through its efforts to stretch.

Lamarckism (2/2)

- Lamarckism may now sound ridiculous for biological adaptation, it is a perfectly reasonable method of heredity for non-biological systems.
- In the next chapter we will combine ideas from evolution and cultural adaptation into a single adaptive mechanism.

Fitness Landscapes

- A visualization of the relationship between genotype and reproductive success
- Fitness Landscape Models: generate the state space of possible solutions and use heuristic methods to efficiently find best (most fit) solutions
- Adaptive Landscapes

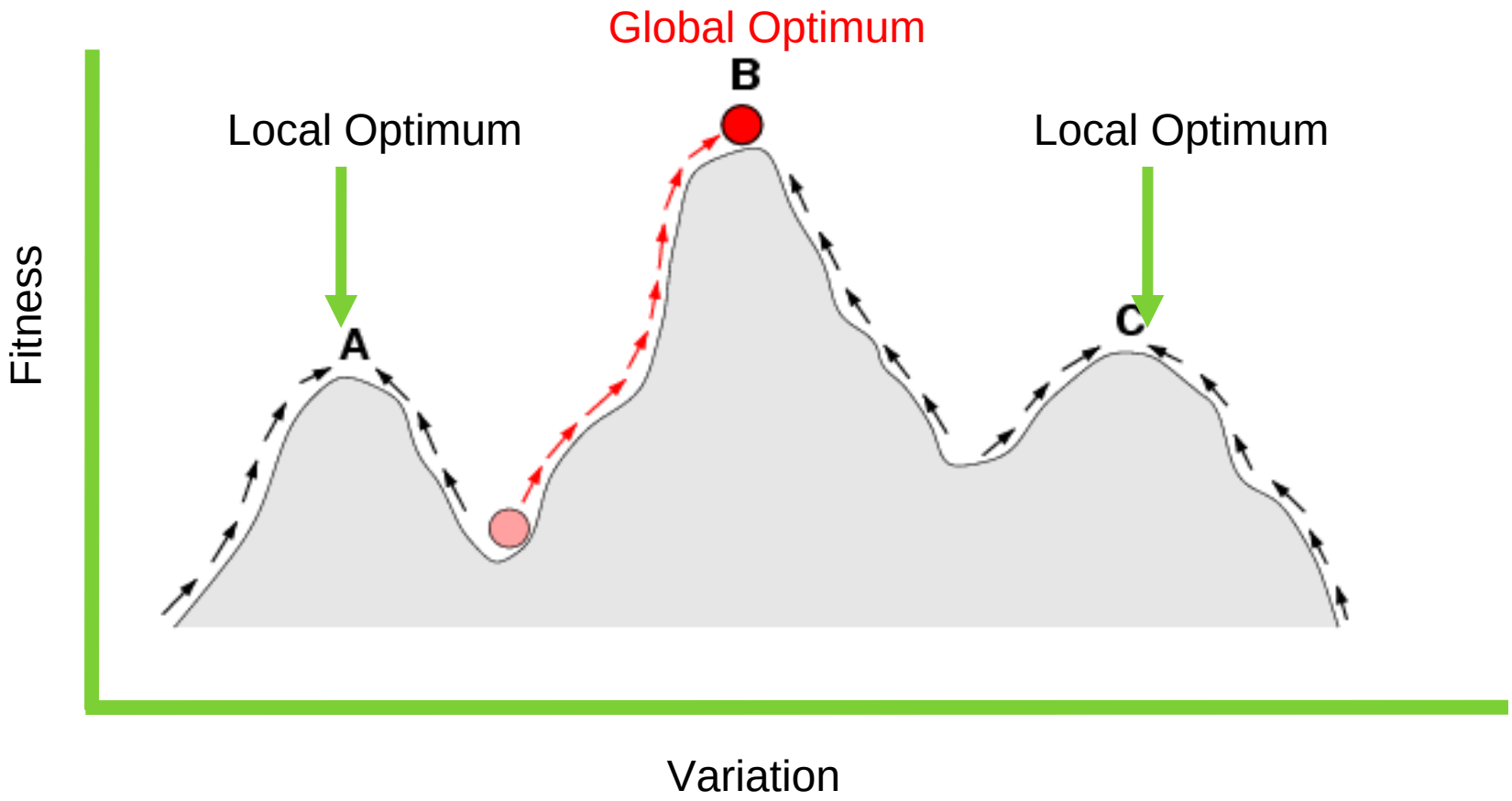
Fitness

- An individual's capability to reproduce
- A genotype's (or variation's) capability to reproduce
  - Proportion of individual's genes in all the genes of the next generation
- A measure of likelihood of survival and reproductive potential

Fitness Landscapes

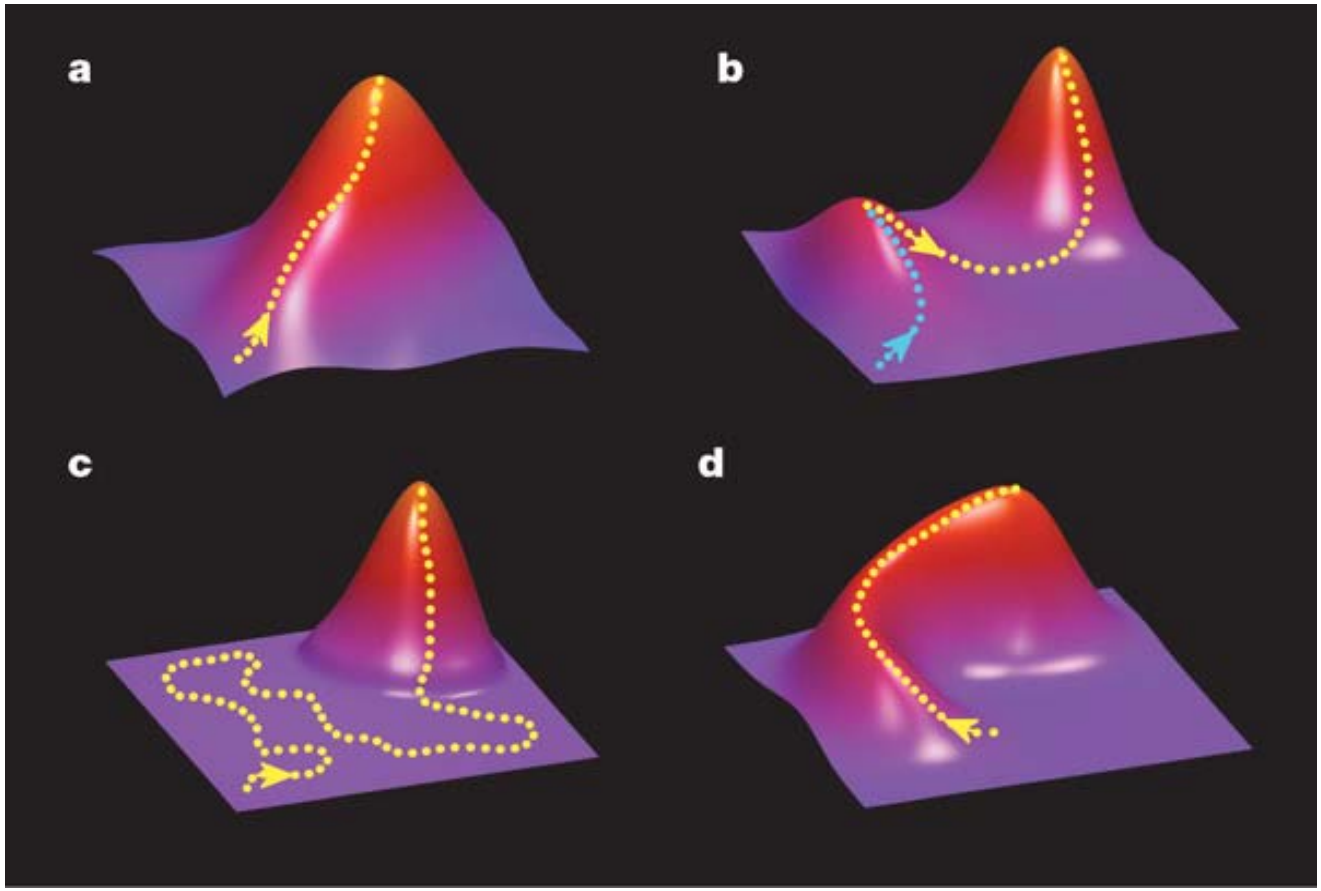
- Evolution is an uphill struggle across a fitness landscape
- Mountain Peaks: high fitness, ability to survive
- Valleys: low fitness
- As a population evolves, it takes an adaptive walk across the fitness landscape

Understanding Landscapes



Modified from http://en.wikipedia.org/wiki/Image:Fitness-landscape-cartoon.png

Understanding Landscapes



From Poelwijk et al. 2007

NK Fitness Landscapes

- Stuart Kaufmann (1993): Origins of Order
- A model of genetic interactions
- Developed to explain and explore the effects of local features on the ruggedness of a fitness landscape

- Why do we care about ruggedness?

NK Fitness Landscape

- A landscape has *N* sites (a site is an amino acid sequence that codes for a specific protein or peptide)
  - Each site contributes to overall fitness of landscape
  - Each of the *N* sites has one of *A* possible states
  - The total number of possible landscape states is A^N.

NK Fitness Landscape

- Calculate fitness of each peptide
- Map out adaptive walk toward uphill values
  - Begin at any of the 16 corners,
  - A series of uphill moves from one corner to its neighbor along one edge of the hypercube.
  - Each move leads to a change at exactly 1 of the 4 amino acid sites,
  - Because the walk is adaptive, each move results in an improved fitness.
  - The adaptive walk ends when a corner is reached which has no immediate neighbors with better fitness.

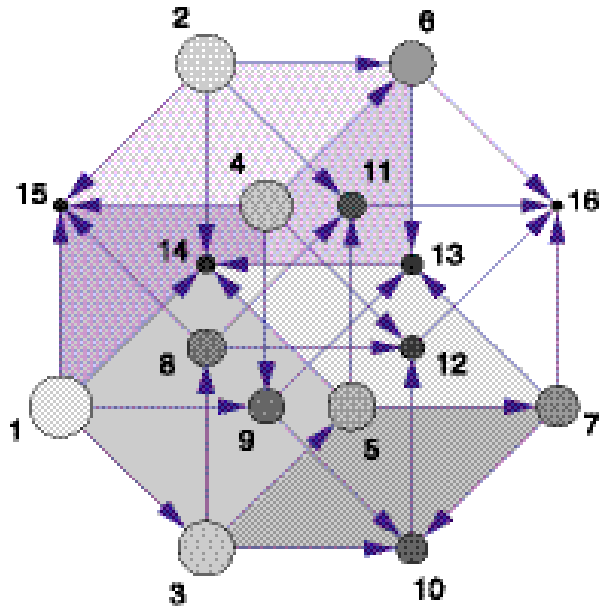


Exhibit 8 Each peptide has been assigned, at random, a rank-order fitness, ranging from 1 (least fit) to 16 (most fit). Moves allowed in an adaptive walk are shown by arrows, all of which are pointing "uphill" toward higher values, represented here by smaller, darker circles.

From http://gemini.tntech.edu/~mwmcrae/esre95.html

NK Fitness Landscape

- In a rugged landscape, some adaptive walks will result in suboptimal fitness
- Because a local, non-global maximum is reached
- This ruggedness is quantified by the *K* parameter of the *NK* model.

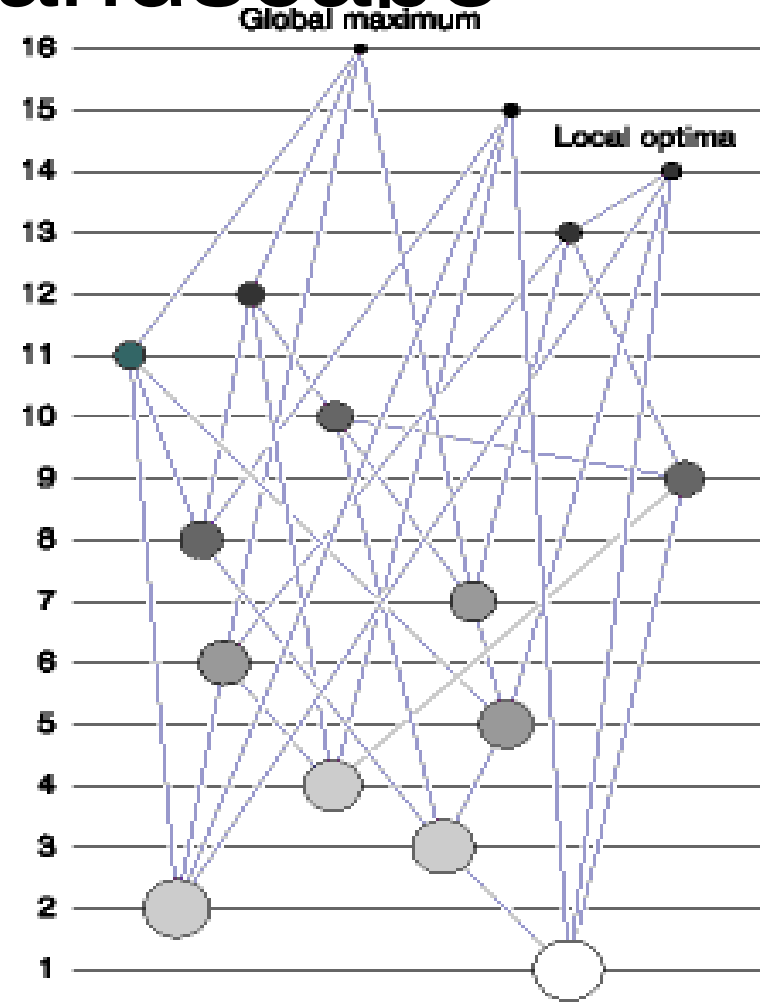


Exhibit C A flattened mapping of possible adaptive walks given in Exhibit B. Walks may only go upward, and so cannot go beyond one of the three local optima, two (those with fitnesses 14 and 15) being suboptimal.

From http://gemini.tntech.edu/~mwmcrae/esre95.html

NK Fitness Landscape

- Consider a fitness landscape for a peptide that is 4 amino acids long (N = 4)
  - Each can be one of 2 different amino acids (*A* = 2).
  - The number of possible peptides upon this fitness landscape is 16.
  - Represent each by a four-bit string (e.g., 0101).
  - Since *N* is 4, this fitness landscape can be mapped in a 4-D space, where each of the possible peptides is at one of the 16 corners of a 4-D cube, or hypercube.

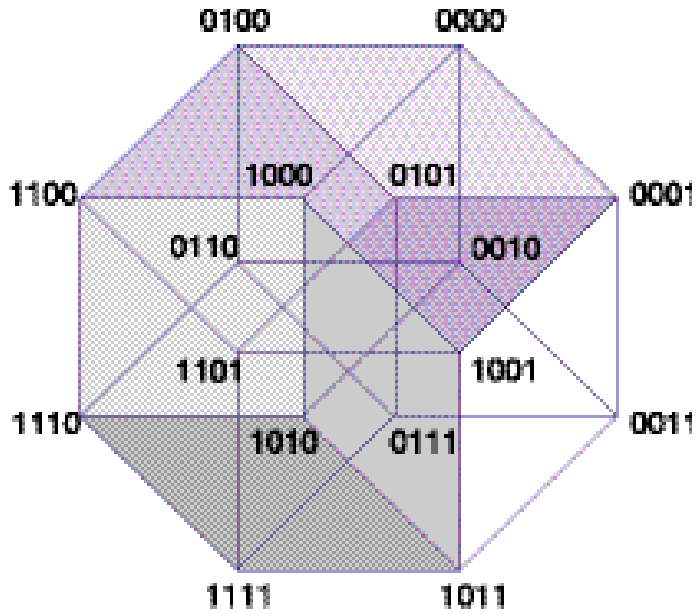


Exhibit A All 16 possible peptides of length four composed of amino acid A (0 in the diagram) and/or amino acid B (1 in the diagram) as mapped onto the corners of a hypercube. Each corner (peptide) differs from its neighbor by just one site (amino acid).

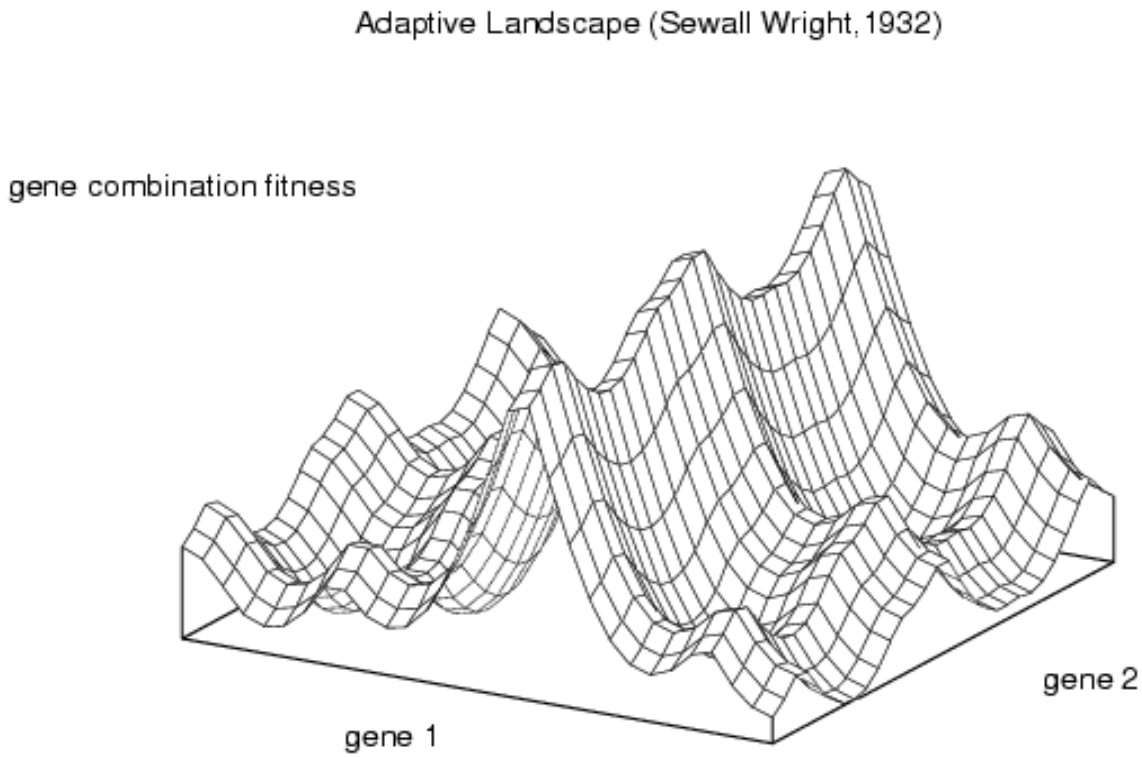
From http://gemini.tntech.edu/~mwmcrae/esre95.html

NK Fitness Model

- Each node of the solution space makes a “fitness contribution” to the landscape that depends on the relationship between itself and the state of the other K nodes
- K ~ the degree to which nodes are interconnected
  - K = 0 all nodes independent (single smooth peak)
  - K = N – 1 all nodes connected (completely random)

Types of Fitness Landscapes

NK: ruggedness due to interconnectedness of alleles
→ Internal



Problems with the NK approach

- Uncertainty of mapping of genotype to phenotype
- Reproductive success easier to judge through phenotype
- Number of phenotypes occupying a single “adaptive peak” increases in proportion to the number of biological tasks that must be simultaneously performed (Niklas 1997)

Principal of Frustration

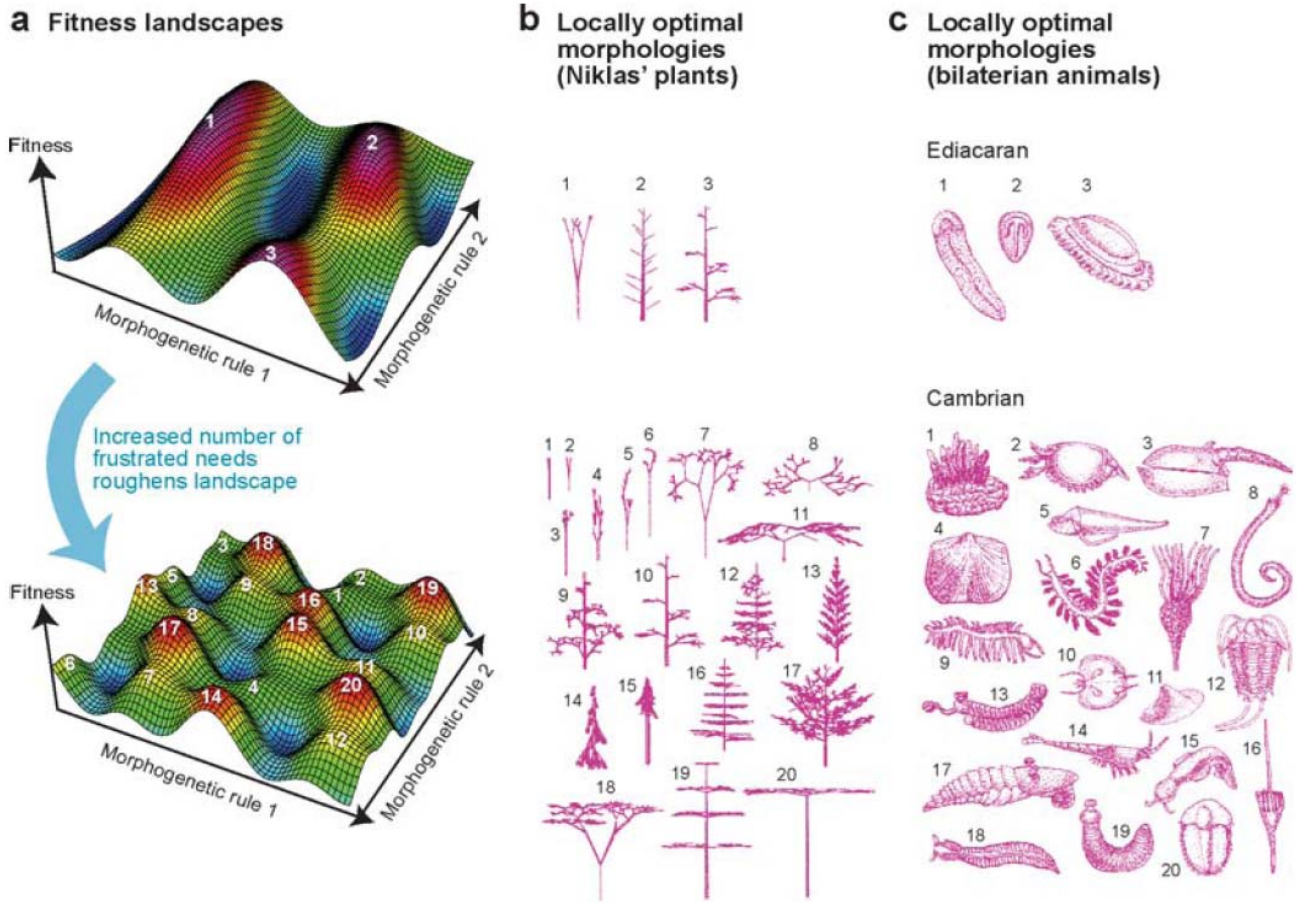
THE PRINCIPLE OF FRUSTRATION

This principle captures the notion that different needs will often have (partially) conflicting solutions, so that the overall optimal design for an organism will rarely be optimal for any of the specific tasks it needs to perform (i.e., there are trade-offs).

From Marshall 2006

Morphogenetic Fitness Landscape

Ruggedness due to trying to optimize too many problems simultaneously → External



From Marshall 2006

Morphogenesis

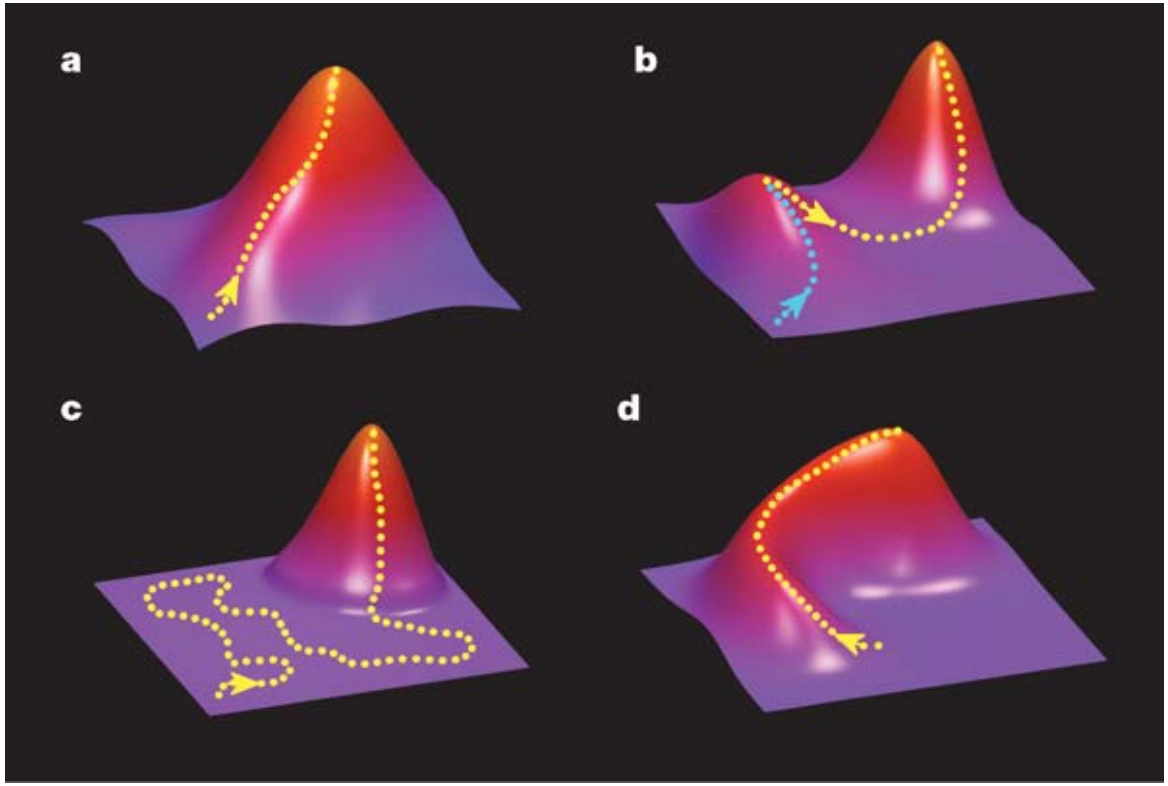
- How shape is formed
- Processes that control organized spatial distribution of cells and/or large-scale features during development
- Morphogenetic Rules: the rules that govern morphogenesis
  - Mathematical Model (Niklas)
  - L-systems (Prusinkiewicz and Lindenmayer)

Niklas 1997

- Geometric Representation
- Generated Adult Morphologies
- All morphologies are built using the same rules

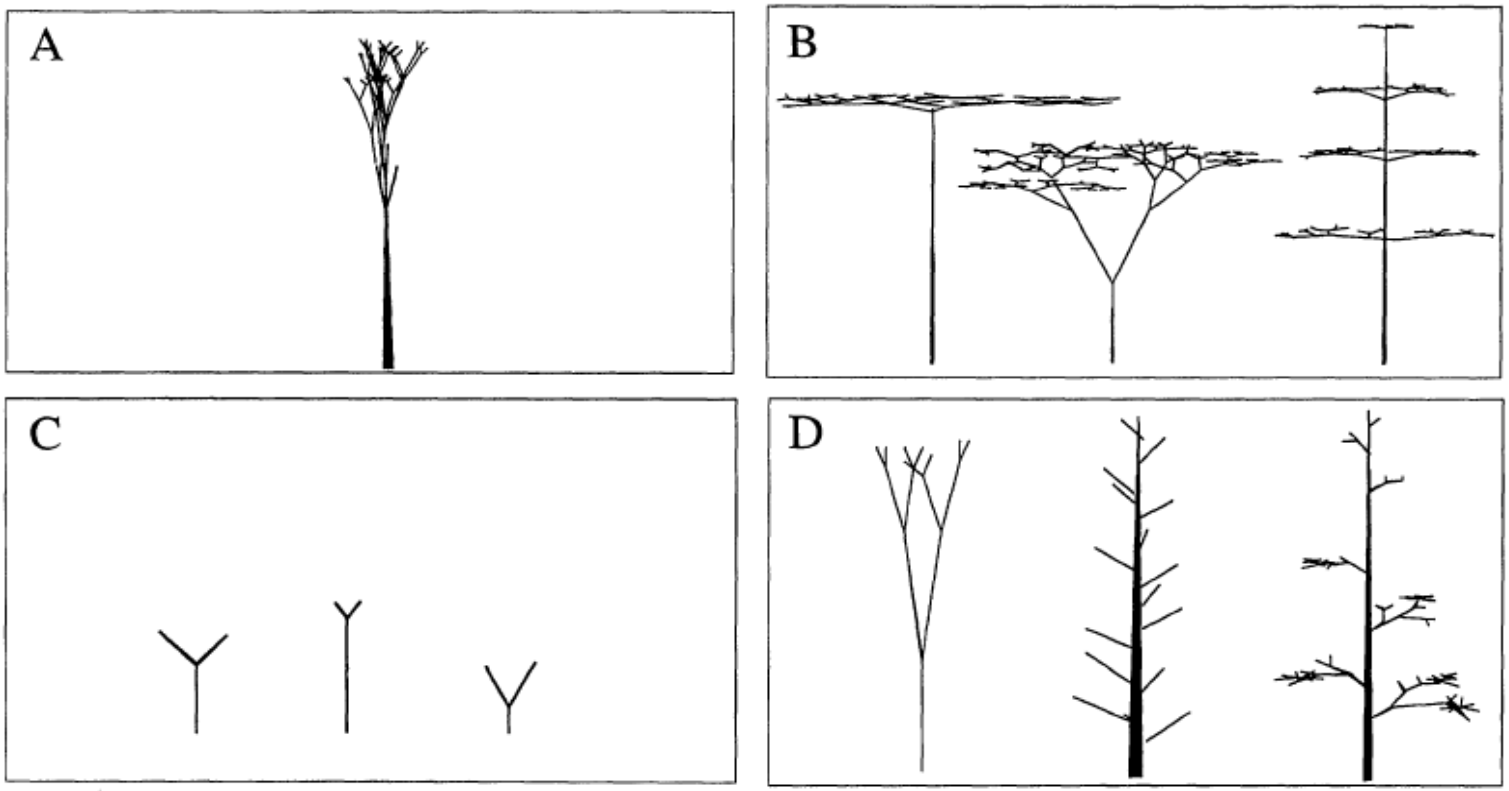
- Fitness:
  - Ability to maximize light interception
  - Mechanical stability
  - Reproductive success
  - Minimize total surface area

Search through Adaptive Walk



Principal of Frustration in Practice

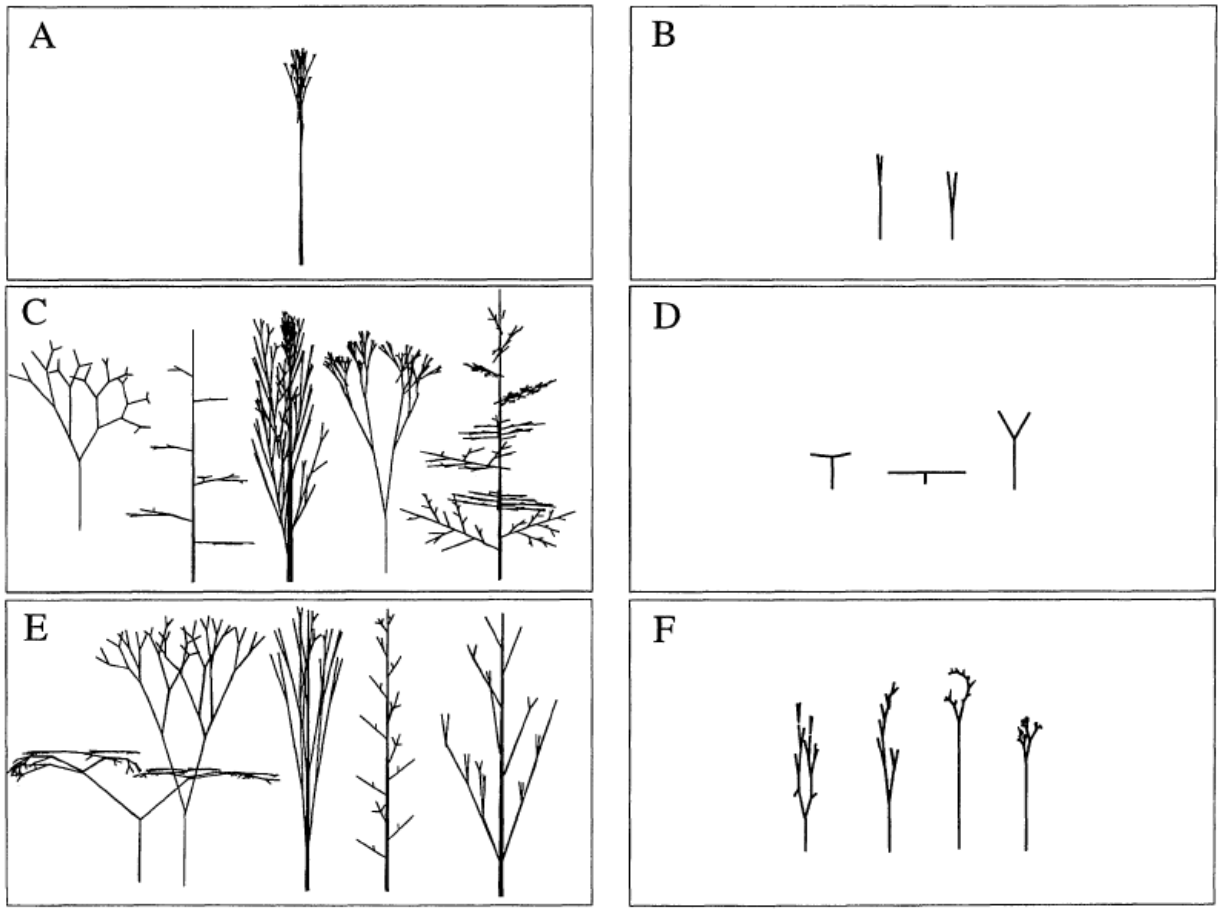
One Task:
A: reproduction
B: Light Interception
C: Minimal Area
D: Mechanical Stability



From Niklas 1997

Principal of Frustration

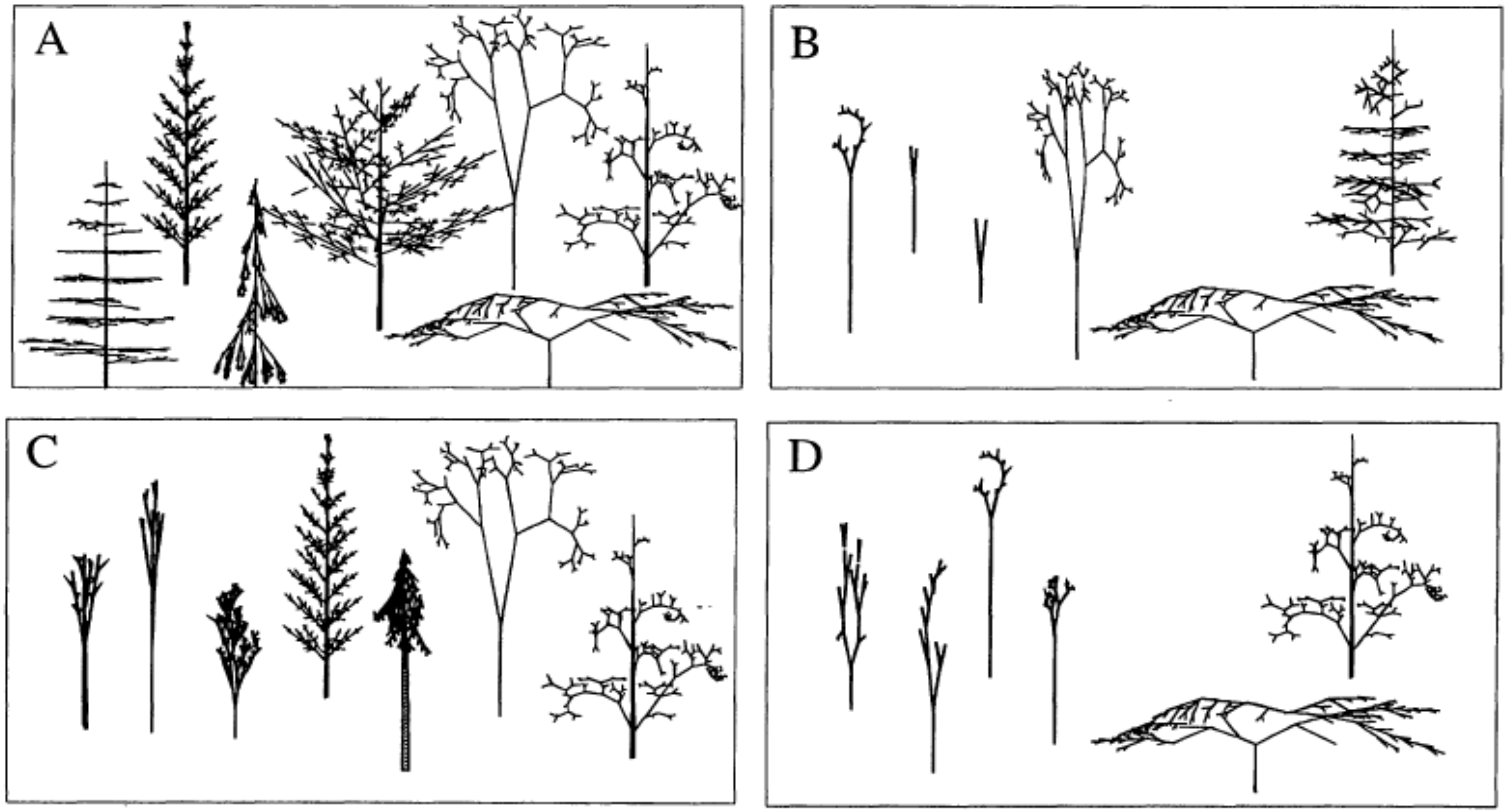
Two Tasks:
A: Stability and Reproduction
C: Light Interception and Stability
D: Light Interception and Area
F: Reproduction and Light



From Niklas 1997

Principal of Frustration

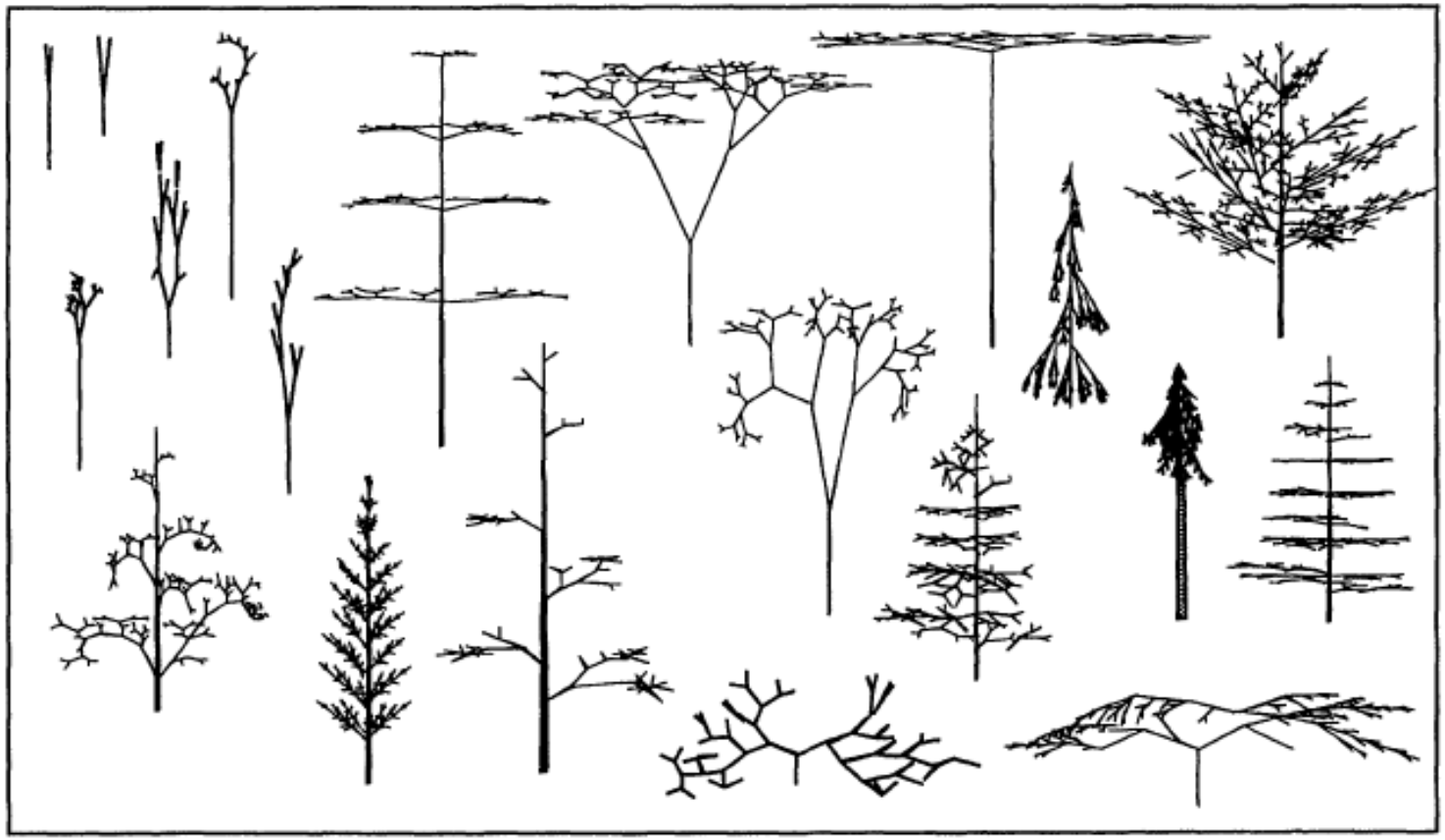
Three Tasks
A: stability, light, reproduction
B: stability, light, area
C: stability, reproduction, area
D: light, reproduction, area



From Niklas 1997

Principal of Frustration

Four Tasks:



From Niklas 1997

Summary of Niklas's Results

A)

Number of tasks defining fitness	Number of morphologies occupying adaptive peaks (Mean ± SD)
1	2.5 ± 0.63
2	3.3 ± 0.51
3	6.5 ± 0.63
4	20

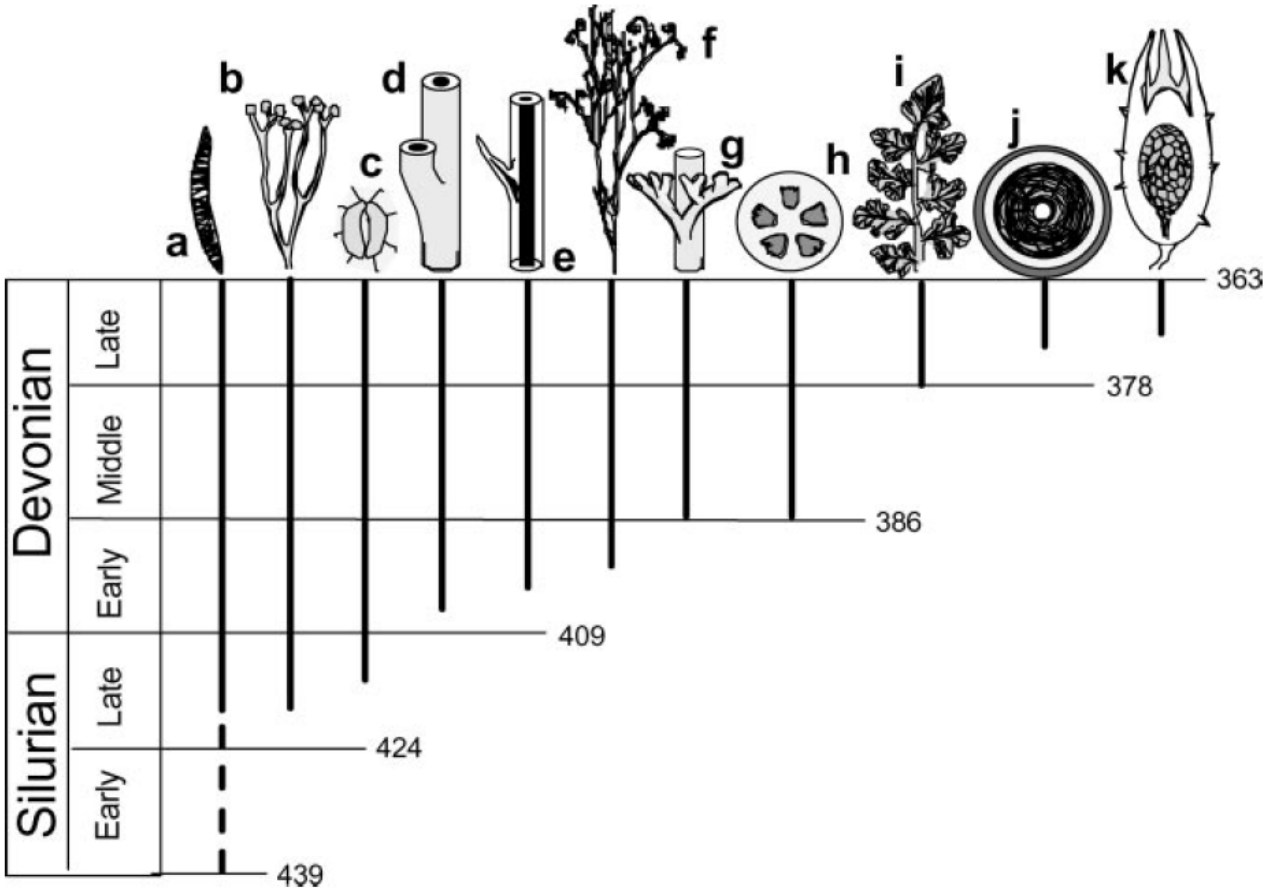
More solutions per peak

A)

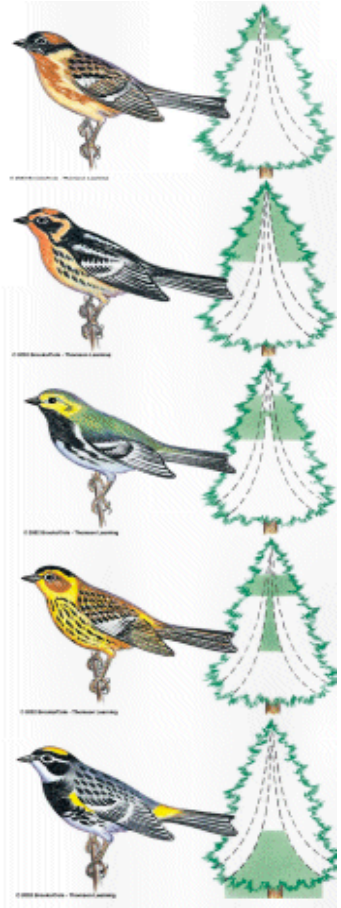
Number of tasks defining fitness	Relative height of adaptive peaks (Mean ± SD)
1	35.3 ± 1.8
2	11.6 ± 0.68
3	7.7 ± 0.14
4	2.4

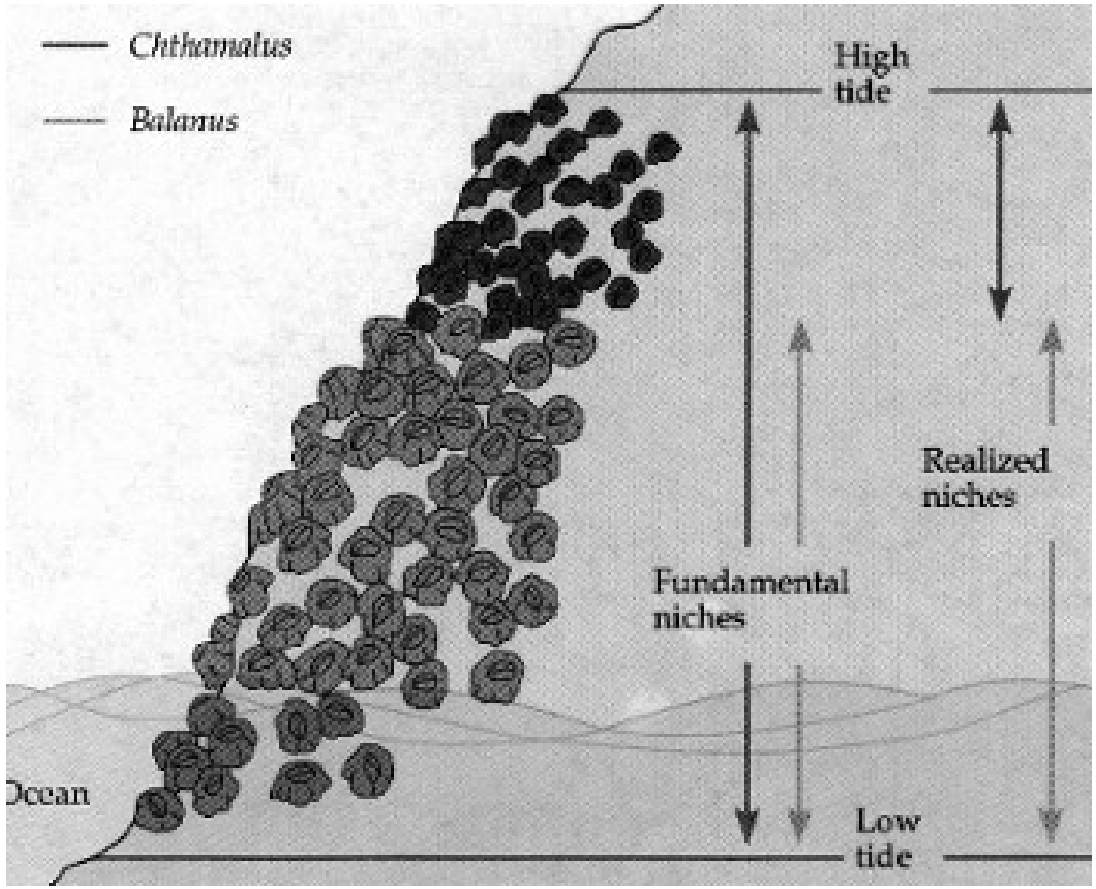
Solutions are less optimal

Niklas 2004


a
b
c
d
e
f
g
h
i
j
k
Devonian
Late
Middle
Early
Silurian
Late
Early
363
378
386
409
424
439

Niche Partitioning





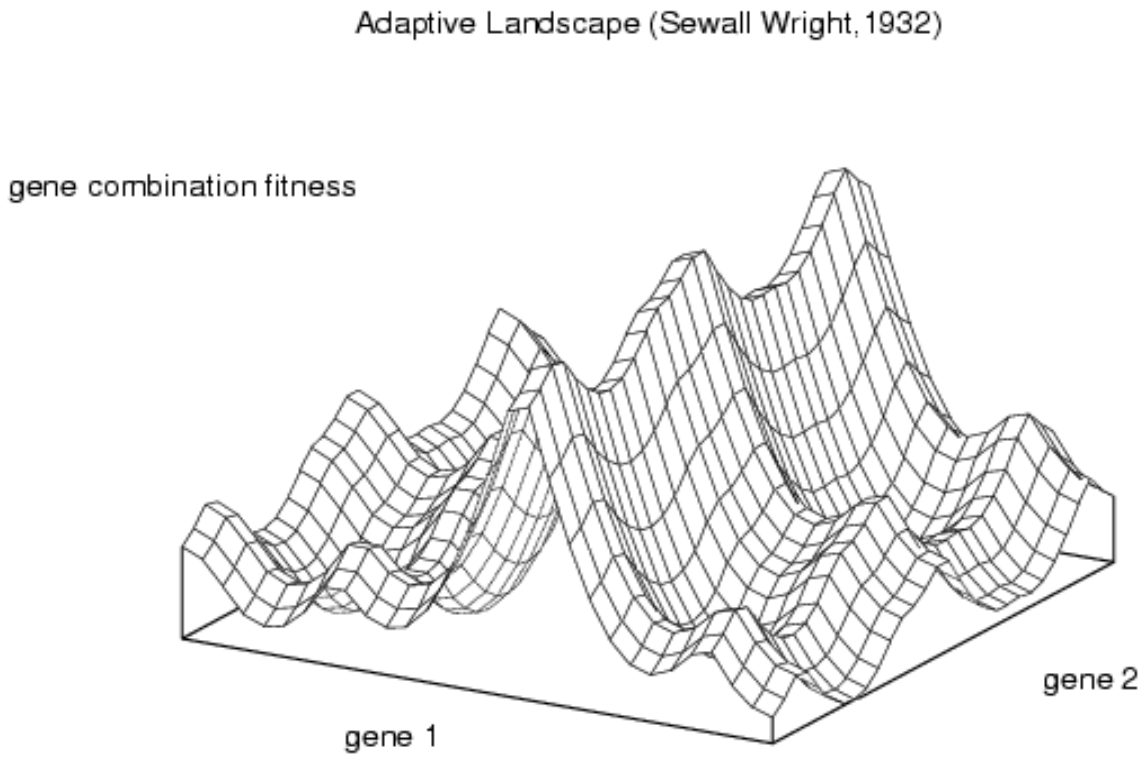
Robert MacArthur

Question

- Are adaptive walks emergent?

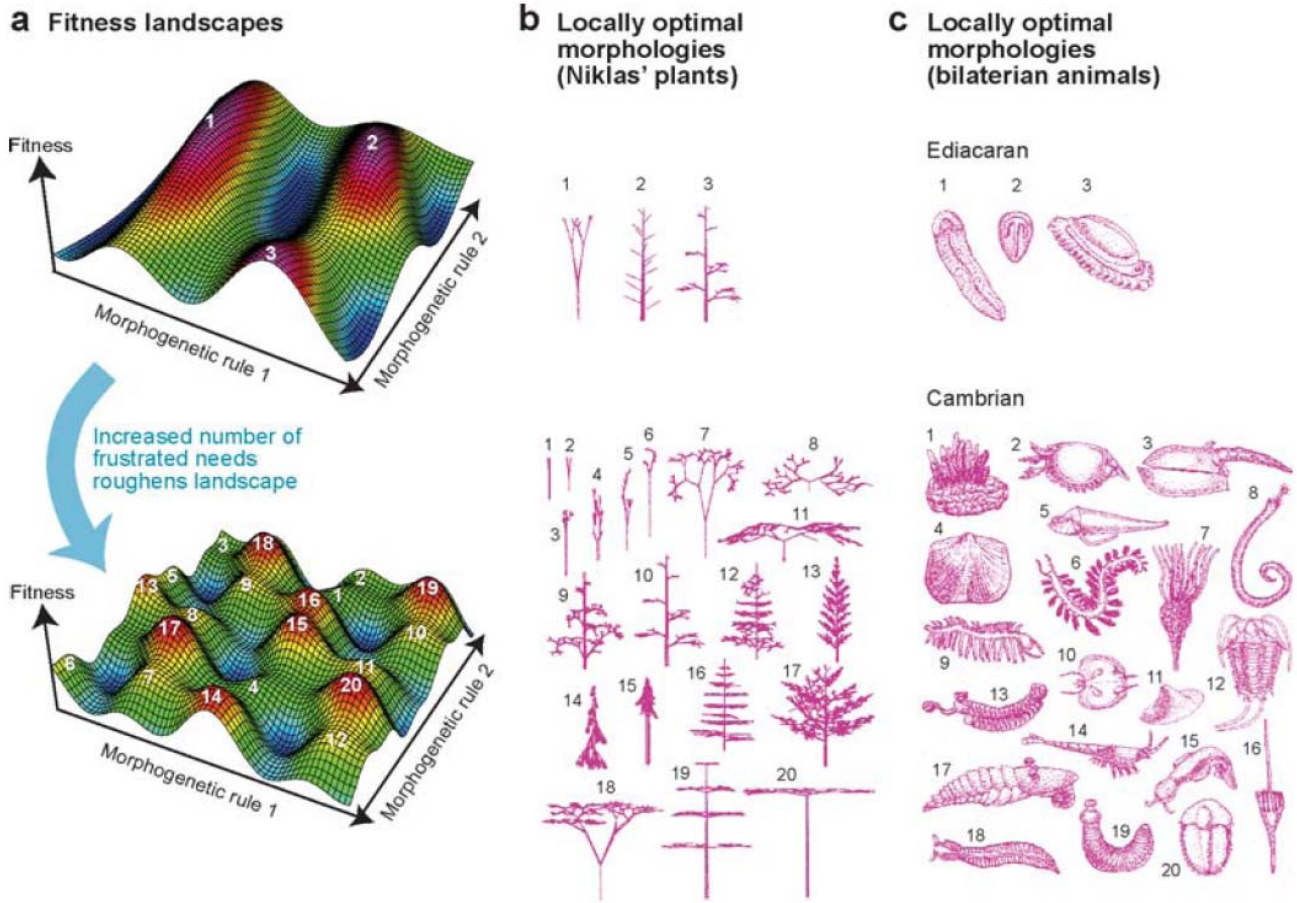
Types of Fitness Landscapes

NK: ruggedness due to interconnectedness of alleles
→ Internal



Morphogenetic Fitness Landscape

Ruggedness due to trying to optimize too many problems simultaneously → External



From Marshall 2006