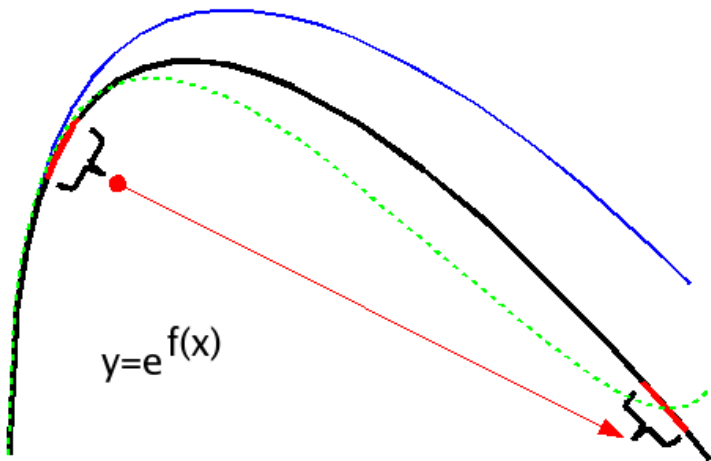
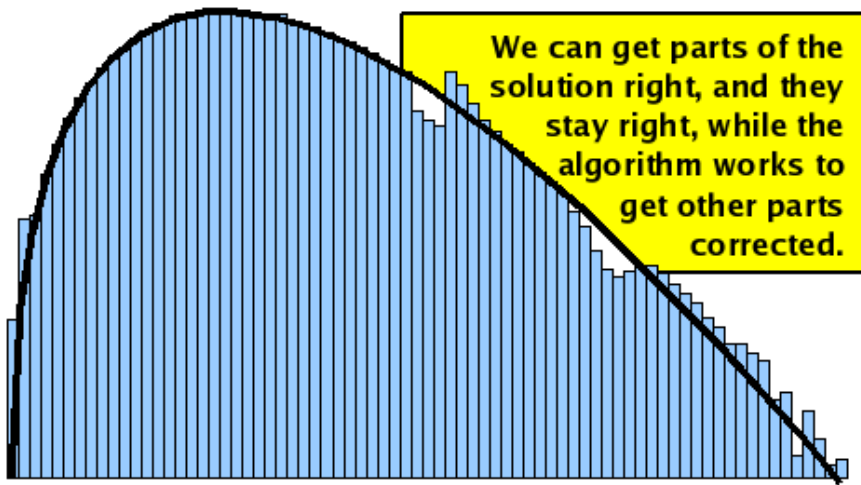


- **Nature's Strategy: If it is worth doing — It is worth doing poorly, in 4 ways:**
 - ① This is the only type of problem GA solves well.
 - ② A 0.1% advantage can crush a competitor in 100 generations.
 - ③ Your implementation can be flawed and buggy and still work!
 - ④ Ridiculously innacurate but fast can beat precise but slow.
- **Nature's Strategy: Winner takes — A lot.**
 - More territory, more food, more mates. More offspring.
- **Nature's Strategy: Independent Organs.**
 - Eyes, ears, liver, teeth, lungs, claws, beaks:
 - Divide and Conquer so expertise evolves *independently*.
- **Nature's Strategy: Patience**
- **Nature's Strategy: Strength in Numbers**

- *Organisms* are data structures capable of representing the solution to a problem. It contains *Genes* which are:
 - Thresholds for decisions to be made,
 - Parameters for a function (such as a line or curve),
 - Estimates of constants (mass, charge, concentration, etc),
 - Functions, Categories, Worth, etc.
- A *Fitness Function* is a heuristic “distance” telling how close an Organism is to solving the problem.
- *Mate*: A child is produced that inherits characteristics from each parent, with the parent for each characteristic chosen at random.
- EVOLVE!
 - Sort the population by the fitness function.
 - Mate the most successful organisms to create children.
 - The children replace the least successful organisms.
 - Repeat until solution found, time runs out, or “fit enough”.



Every segment value is determined by every other, so small changes due to noise on the left may mean large changes on the right. You must get the entire answer right at once. This is *overly dependent*.



- Find what a solution might look like.
 - Do you know how to use it?
 - Can it be just *partially correct*?
- We want genes to be independent; so specify
 - Thresholds for decisions,
 - Constants for concentration or mass,
 - Parameters for functions *you already know are useful in your particular problem*, such as decay curves, parabolas and lines.
 - Function choices *you already know are useful*, Category choices *you already know are useful*, etc.
- The framework is the *environment* for the organism, and the environment provides *constraints*. With no constraints, the space is too large to search.
- Match your *fitness function* to your framework.

Consider matrix multiply; an $M \times N$ matrix is multiplied by an $N \times P$ matrix, producing an $M \times P$ result:

$$C_{M \times P} = A_{M \times N} \cdot B_{N \times P}$$

But sometimes, a linear algebra program can finish the transpose problem faster, despite the overhead of transposing:

$$C_{P \times M}^T = B_{P \times N}^T \cdot A_{N \times M}^T$$

So, the question is: To Transpose, or not To Transpose?

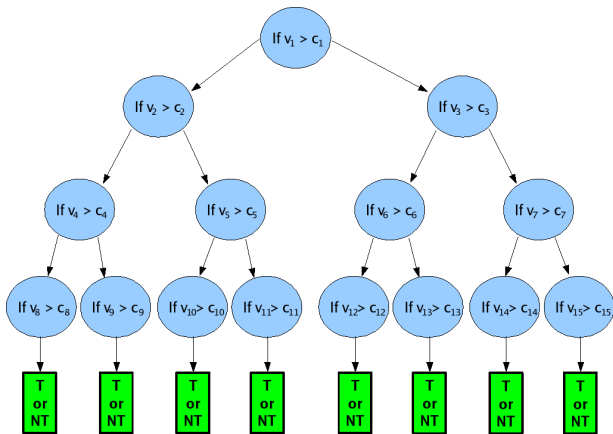
The answer is *non-linear*, and depends on several factors:

- $M \times P$, $M \times N$ and $N \times P$ determine copy overhead,
- M and P and N determine blocking and cleanup costs,
- $M \times N \times P$ is proportional to actual flops required.

Given execution times for both Transposed and Non-Transposed operations, try to find two linear functions, f_T and f_N , that best approximate run times if transposed first or not. We will give each function the same 16 pieces of info about the problem:

$c_1 \cdot M$	$c_2 \cdot c_{14} \cdot M \bmod (M_b)$	$c_3 \cdot \lfloor M/M_b \rfloor$
$c_4 \cdot N$	$c_5 \cdot c_{15} \cdot N \bmod (N_b)$	$c_6 \cdot \lfloor N/N_b \rfloor$
$c_7 \cdot P$	$c_8 \cdot c_{16} \cdot P \bmod (P_b)$	$c_9 \cdot \lfloor P/P_b \rfloor$
$c_{10} \cdot M \times N$	$c_{11} \cdot N \times P$	$c_{12} \cdot M \times P$
$c_{13} \cdot M \times N \times P$	$\text{if}(\lfloor M/M_b \rfloor > c_{14}) : (1, 0)$	
$\text{if}(\lfloor N/N_b \rfloor > c_{15}) : (1, 0)$	$\text{if}(\lfloor P/P_b \rfloor > c_{16}) : (1, 0)$	

So the gene consists of 32 constants; c_1, c_2, \dots, c_{16} are the constants for f_N , and $c_1', c_2', \dots, c_{16}'$ are the constants for f_T . Then we can just compute an estimate for f_N , and an estimate for f_T , and see which estimate of execution time is lower.

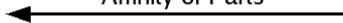


So the gene consists of 15 variable indices, 15 threshold constants, and 8 T/F decisions (transpose or do not transpose).

Parts List

ID / Weight / Concentration
1 13 27
2 15 21
3 2 50
4 101 13
5 57 2
6 11 18
7 14 63
8 68 7

Feedback Adjusts Utility
of Parts, and Perhaps
Affinity of Parts



Assemblies

1, 3, 6
5, 7, 8
1, 6, 8
2, 3, 4, 7, 8
1, 6

- A pointer plus a score ($P+S$) is 8 bytes.
- An organism record is typically 1024 bytes.
- For identical scores, Quicksort moves exactly *as many* items either way — But *128 times less memory* with ($P+S$).
- You must visit every organism anyway to score it. Build the ($P+S$) as you go.
- Bonus: ($P+S$) may fit in L_2 or even L_1 and make the sort an extra 10 to 50 times faster.
- Bonus: Fixed position organisms mean easy reference.
 - Children can have reference to their parents.
 - Children can combine partial solutions of other organisms.
 - Tracking parents lets us see if an ancestral line is improving.

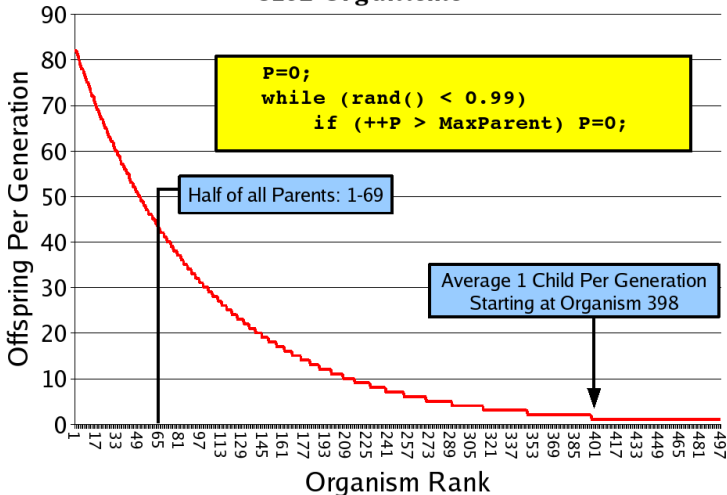
It's Worth Doing Poorly!

- The fitness function is going to be invoked *millions* of times.
- Minimize or eliminate *decisions* in the fitness function; decisions slow us down.
- Minimize computation in the fitness function. Parallelize what computation you can (Modern SSE on any Intel or AMD chip allows up to 8 floating point operations in parallel, per cpu. This alone can make a summation or product 7-8 times faster).
- Use a cheap heuristic early, and the expensive fitness function later. e.g. use taxicab distance to start (N adds) and Euclidean distance later (N Multiplies and N adds).
- Slight selective pressure is often enough if your generations are fast.

BIG. But not TOO Big.

- The bigger the population, the easier it is to hit a useful mutation. Nature's favorite strategy is *overwhelming numbers*, a hundred thousand eggs and the death penalty for the slightest error, or just bad luck. Every fish in the sea is a lottery winner.
- Fit the population in 90% of main memory.
- Fit the *breeding population* in 65% of L_2 .
- Use small populations only if the fitness function is unavoidably costly.
- We want *thousands* of generations. Trim the population size and fitness function to permit this.

Simple 1% Exponential Decay 8192 Organisms

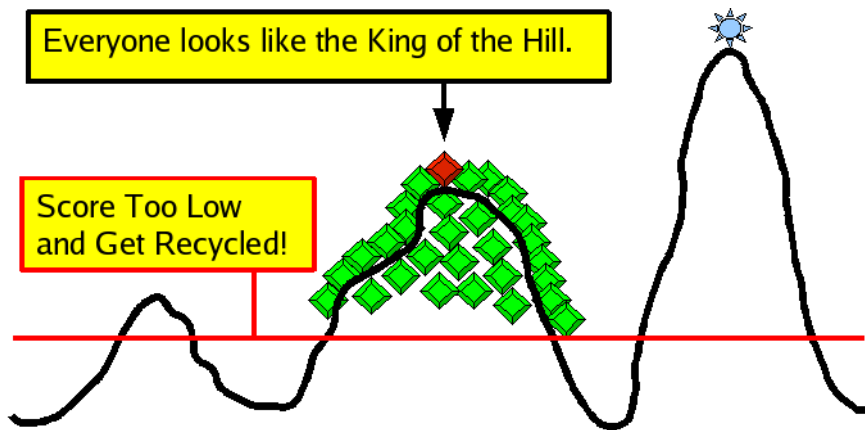


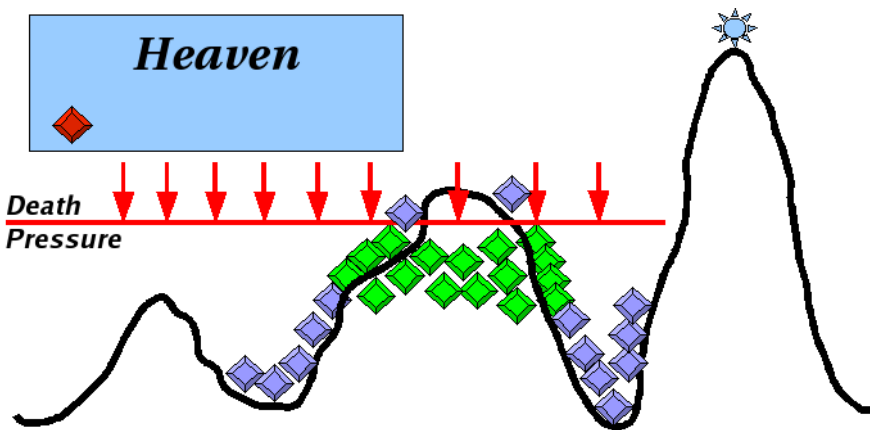
- **Mating does not have to be random.**

- If you can measure compatibility of mates, you can choose *one* parent at random, then test a random selection of ten potential mates and choose the most suitable.
- If the problem can be broken into parts (like puzzle pieces), we can try to mate by synergy – Mate the organisms that produce a bigger chunk of the final solution, or mate organisms that have the most parts in common.
- OR BOTH. Nothing says a mating must produce one and only one child. If different mating strategies have different benefits at different times, select one at random or just do them all.

- **Mating does not have to be equitable.**

- Early mating might draw equally from both parents.
- Later mating might “fine tune” organisms, drawing 90% from first parent, 10% from second. Or even transfer just one gene.





- **MATLAB!** *Hundreds* of times slower than straight C.
- **Interpreted Languages**, even partly compiled (e.g. Java, C#).
- **Object Oriented Languages**. OO Languages makes heavy use of pointers and indirection, which defeat the architecture's prediction mechanisms and cause frequent stalls.
- OO Languages may get less than 1% of the peak performance.
- **Virtual Memory**. Disk access = $10,000 \times$ Memory access.
- **Database Accesses**. *Disk access = $10,000 \times$ Memory access!*
- **Crossover**. Remember this as exactly the wrong approach; it forces dependence where none is necessary (e.g. It is impossible to get just middle genes from Dad and rest from Mom. Cannot get just one key gene from Dad unless it happens to be in first position).

- Frame the problem to maximize the *independence* of genes.
- When possible, make your solution the composition of independently evolved partial solutions.
- Fitness must be relative, not a binary choice. We must be able to distinguish the relative fitness of two *awful* solutions.
- Try early (fast, rough) and late (slow, precise) fitness functions. The axe that gets us to “serviceable” can be replaced in generation 500 by the scalpel that gets us to “perfect”.
- Consider early (rough) and late (fine) mate/mutate strategies.
- Notice Nature works problems that have *many* valid solutions.
- Remember the beauty of Nature’s Maxim:

If It Is Worth Doing, It Is Worth Doing Poorly!