SBSE Day 2
(if time allows)

Evolutionary Computation (EC)

Based on Darwin’s theory of *natural selection*

Survival of the fittest:

- Those that survive a *selection process* are the best
- Selection is brutal

Concept of *organisms*

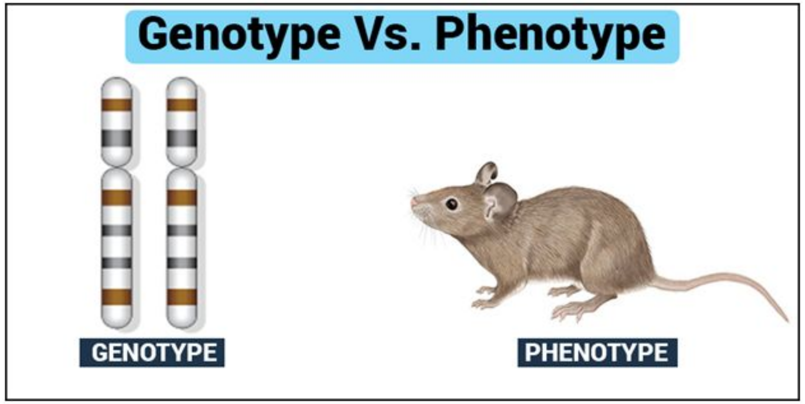
- Kind of like other approaches …
    - Each solution encoded as an organism/individual
- They live and die
    - And you’re the one killing them

EC Concepts

Individuals have *genotypes* and *phenotypes*

- Uses concept of DNA to encode solution

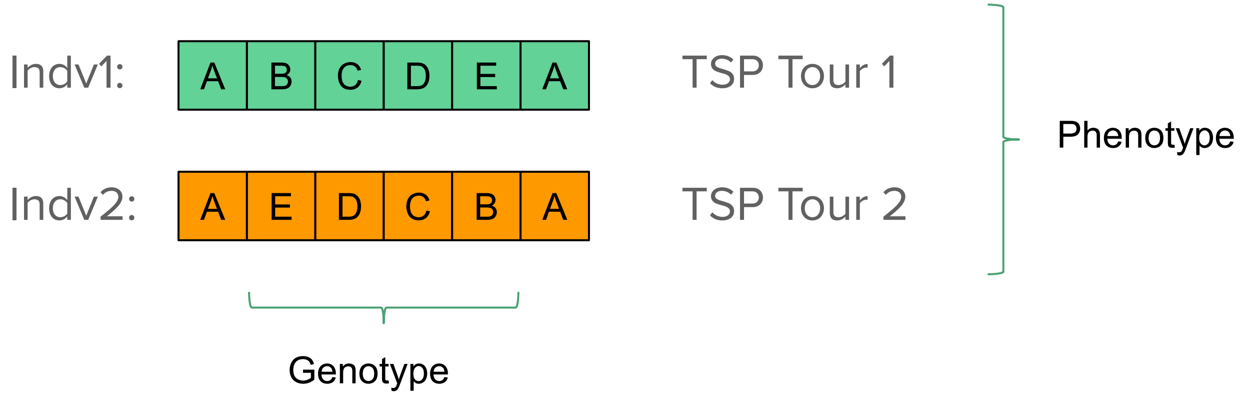
Genotype:	unique genome	(encoded solution)
Phenotype:	physical characteristics	(expressed behaviors)



EC Concepts

Evolution that is based on both *sexual* and *asexual* reproduction

In either case, everything ties to the *individual's genome*



EC Concepts

Using concept of genomes, we will **evolve** our solutions towards an optima

- Again, may not be global!
- EC family does pretty well though…

Core concepts:

- Mutation ➔ asexual reproduction
- Crossover ➔ sexual reproduction
- Selection ➔ survival of the fittest

Reproduction (gross)

Selection

Purpose:

- Retain the “elites”
  - Seeding your next generation
- Selecting candidate parents
- Selecting mutation candidates

Elite Preservation

Select the best genome(s) and seed the following generation

- Populate the remainder with crossover / mutation / random generation

Rapidly increase performance as you don’t **lose the best**

- May get you stuck in a local optima though…

The more you retain, the less diversity you introduce

Sidesteps breeding process

https://stackoverflow.com/questions/298301/roulette-wheel-selection-algorithm

Roulette Wheel Selection

Generate a random number between 0 and 1

Divide range between each individual:

0 - 0.3:	individual 1
0.3 - 0.4:	individual 2
0.4 - 0.5:	individual 3
0.5 - 0.57:	individual 4
0.57 - 0.63:	individual 5
0.63 - 0.68:	individual 6
0.68 - 0.8:	individual 7
0.8 - 0.85:	individual 8
0.85 - 0.98:	individual 9
0.98 - 1:	individual 10

Iterate for as many times as you want to select candidates

Tournament Selection

Select some *k* for tournaments

- *k* = # of individuals competing

Randomly populate tournaments

- Best individual (highest fitness) selected as winner

Number of candidates should be a factor of 2

- Need two parents

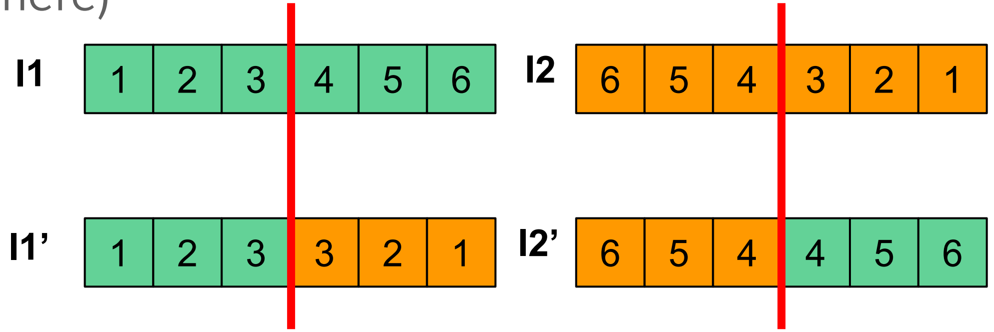
(Unless if you want to set an odd one out as a mutation candidate)

Crossover

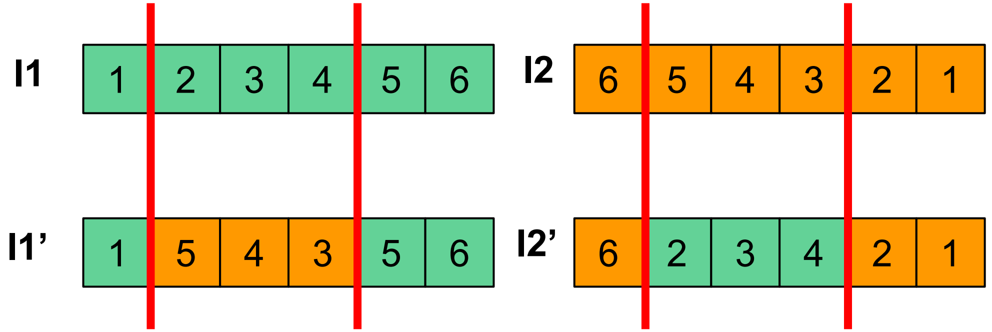
Several different types, but why do we do crossover?

(Non-TSP genome here)

Single-point

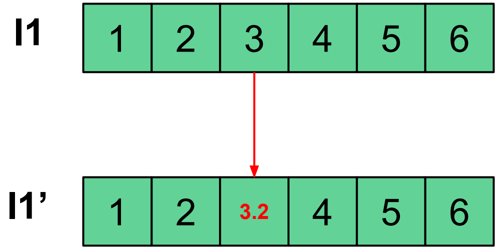


Two-point



Mutation

Single-point



How to mutate?

- Add a random constant on [-**α**,**α**]
- Select a random choice (if an enum)
- Toggle from True to False
- ...

Why?

MARVEL COMICS
1ST ALL-TOXIC COLLECTOR'S ITEM ISSUE!
THE FIRST SUPER HERO . . . FROM NEW JERSEY!
THE TOXIC AVENGER
"A HIDEOUSLY DEFORMED CREATURE OF SUPERHUMAN SIZE AND STRENGTH IS BORN!"
50 YEARS
CAPTAIN AMERICA 1941 – 1991

Genetic Algorithms (GA)

Created at...

MICHIGAN

by John Holland in the 70’s [1]

GA

(1) Create **random** population of solutions
(2) Evaluate against fitness criteria
(3) Apply genetic operators (selection/mutation/crossover)
(4) Go back to (2) and iterate until optimal solution found

Again, **just a for loop** with some additional steps

https://www.youtube.com/watch?v=bBt0imn77Zg
https://youtu.be/pgaEE27nsQw?t=52

GA Parameters

Highly-configurable process (often a sensitivity analysis is required)

Number of generations:	How long to iterate the GA
Population size:	How many candidate individuals / generation
Crossover rate:	% of population to crossover
Mutation rate:	% of population to mutate
Crossover/mutation/selection type:	Hopefully self-evident...
Elite preservation:	Preserving elites? How many?

Consider computation time:

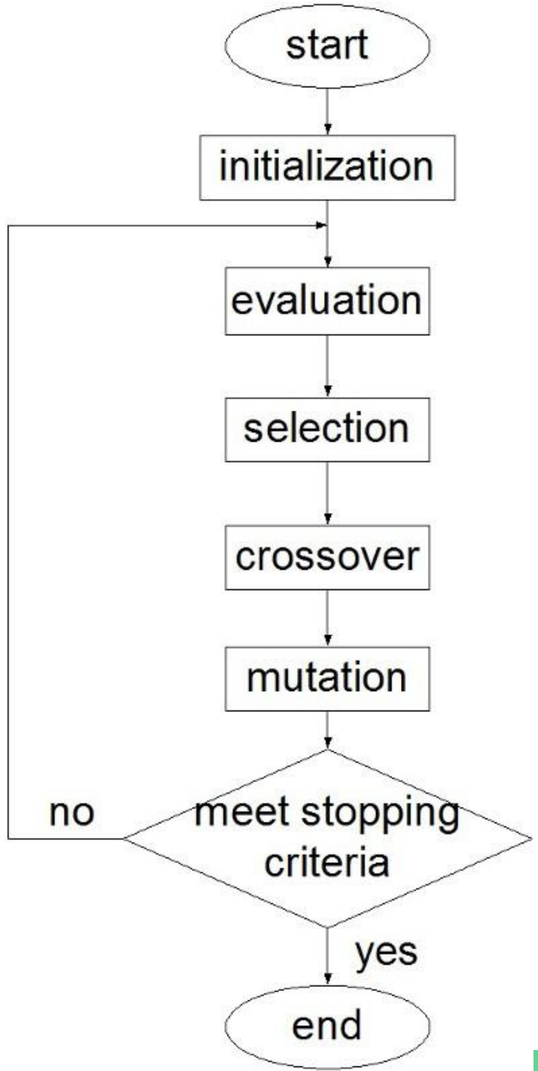
If an evaluation takes **20 seconds** (may be a physics simulation)...

20 seconds * 100 generations * 500 individuals = 1,000,000s ~= 278hrs ~= **11.5 days**

GA Flowchart

Generally toss in an additional
eval after meeting stopping criteria

- Simply to check any new children or mutants



Impact of Parameters

General rule of thumb:

- High crossover rate `~ 0.7`
- Low mutation rate `~ <= 0.2`
- Make your population as large as your computational abilities can handle

Size of genome determines size of search space!

- A small genome pretty much precludes the need for search (just enumerate!!)
- But, a genome that comprises 30 bits…
  - 2^{30} = **1.07 billion possible answers**

GA issues (and EC in general)

No formal way to guarantee **convergence to optimal solution**

We may converge prematurely to a local optima

Generally only a single objective is being optimized

- What about the rest of the search space?

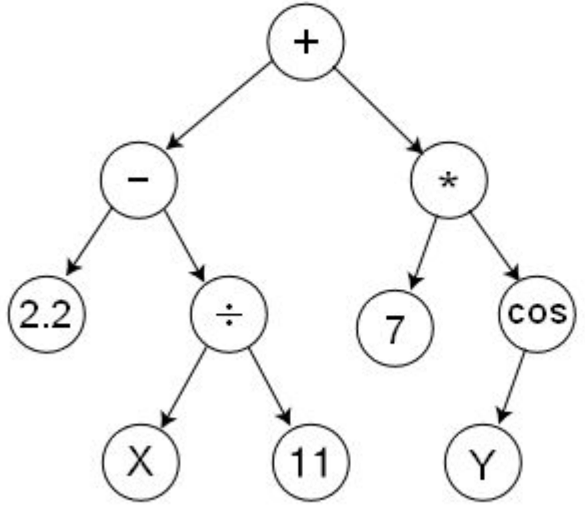
Long run time!

Genetic programming (GP) [2]

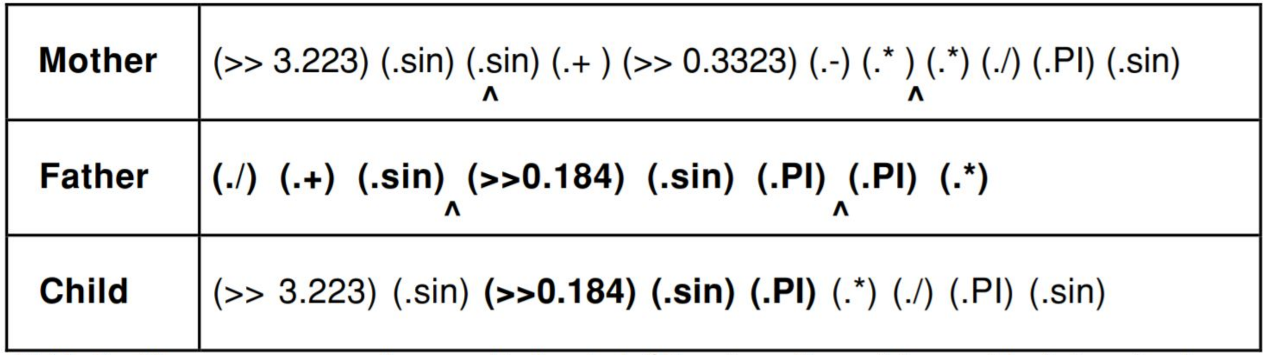
Where a GA evolves a solution, a GP evolves a *program*

Representations:

- Abstract syntax tree
- Stack [4]



$$\left(2.2 - \left(\frac{X}{11}\right)\right) + \left(7 * \cos(Y)\right)$$



Mother	(>> 3.223) (.sin) ^ (.sin) (.+) (>> 0.3323) (.-) (.*) ^ (.*) (./) (.PI) (.sin)
Father	**(./) (.+) (.sin) ^ (>>0.184) (.sin) (.PI) ^ (.PI) (.*)**
Child	(>> 3.223) (.sin) **(>>0.184) (.sin) (.PI)** (.*) (./) (.PI) (.sin)

Table 2. The crossover operation used in the stack-GP system. Two points are picked at random in each of the parents, and one child sequence is created by inserting the sequence enclosed by the points in the father into the space defined by the points in the mother.

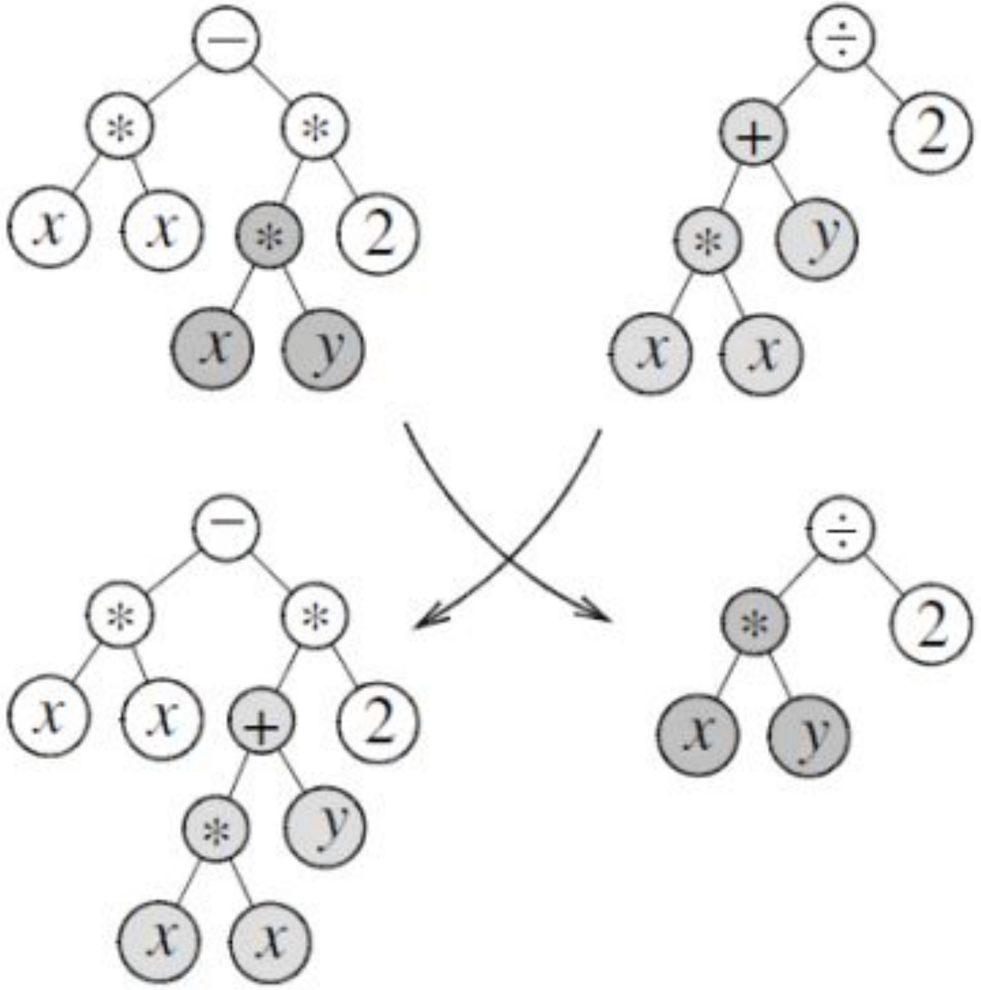
Stack-based genetic programming. Perkis 1993.

GP

Still uses same procedure as a GA except:

- Evaluation now means parsing and executing the tree (or stack)
- Crossover/mutation are now tree-specific (or stack)
  - Swapping/modifying subtrees (or stacks)
- Selection remains largely the same

GP example



GP

Biggest issue though?

GP produces a lot of garbage output

- Trees are quite complex
- Swapping in/out sub trees can be quite destructive

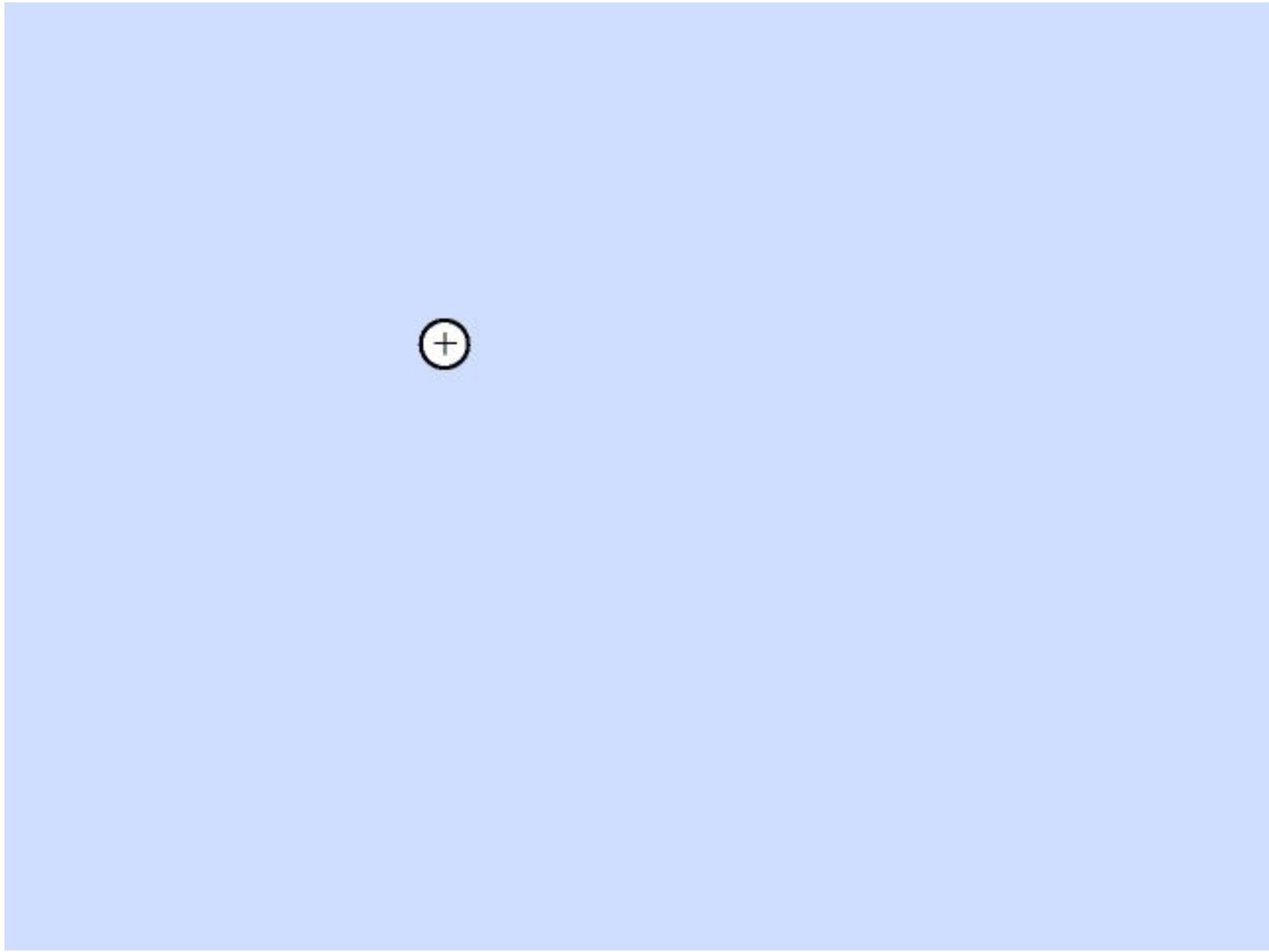
Generally there is some additional work to ensure valid trees are mainly generated

- E.g., grammar-based trees

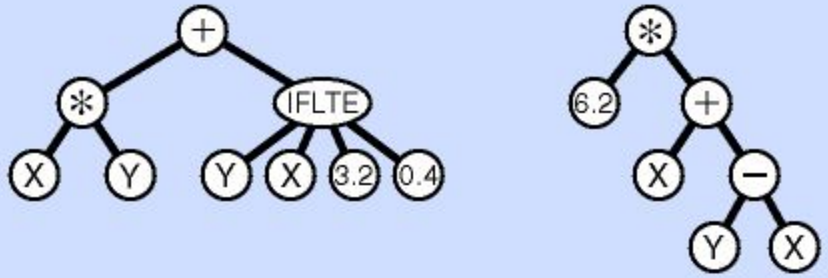
GP

Few examples: http://www.genetic-programming.com/gpflowchart.html

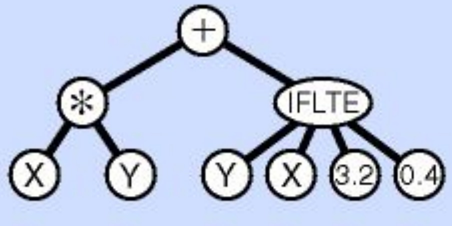
Creation

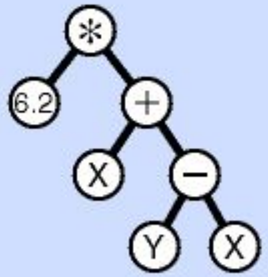


Mutation



Crossover





GP: Grammar [3]

Tree must conform to a formal grammar (grammatical evolution)

Remember your lex/yacc or flex/bison days?

Subset of C: https://web.archive.org/web/20110721124315/http://www.grammaticalevolution.org/tutorial.pdf

```
<func>         ::= <header>
<header>       ::= float symb(float X) <body>
<body>         ::= <declarations><code><return>
<declarations> ::= float a;
<code>         ::= a = <expr>;
<return>       ::= return (a);
```

```
float symb(float x)
{
  float a;
  a = <expr>;
  return(a);
}
```

Evolutionary Strategy (ES)

Very *very* similar to GAs, however:

ES tend to encode their parameters as *real values*

- GAs can too, but more standard in ES

Also, ES will encode its evolutionary parameters within the genome

- Mutation/crossover rates carried along
- Self-adapting

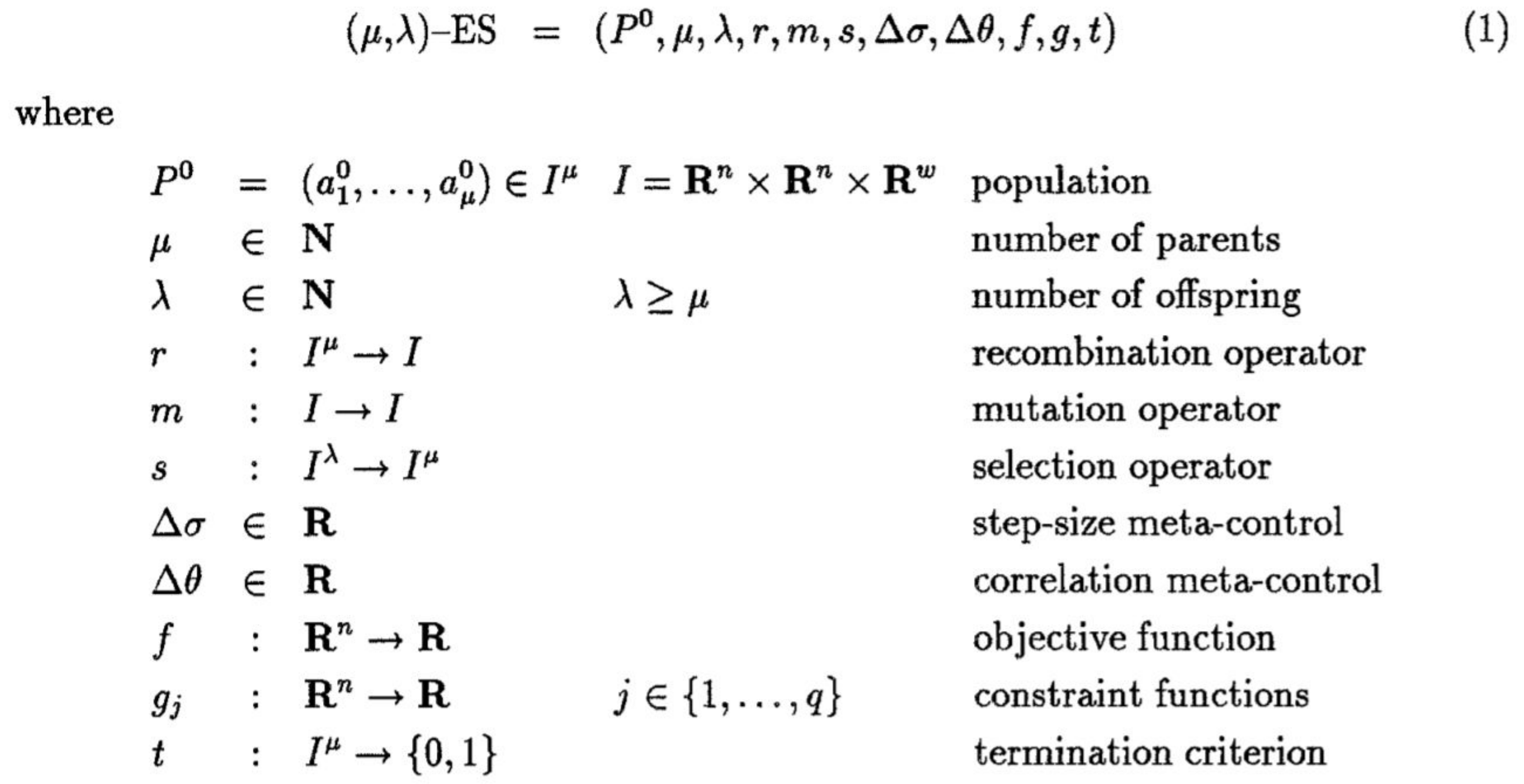
ES

Initial: (1+1) EA

1 parent, 1 offspring

More common: (μ+$\boldsymbol{\alpha}$) EA

ES



$$(\mu,\lambda)\text{–ES} \quad = \quad (P^0, \mu, \lambda, r, m, s, \Delta\sigma, \Delta\theta, f, g, t) \tag{1}$$

where

P^0	$=$	$(a_1^0, \ldots, a_\mu^0) \in I^\mu$	$I = \mathbf{R}^n \times \mathbf{R}^n \times \mathbf{R}^w$	population
μ	$\in$	$\mathbf{N}$		number of parents
λ	$\in$	$\mathbf{N}$	$\lambda \geq \mu$	number of offspring
r	$:$	$I^\mu \to I$		recombination operator
m	$:$	$I \to I$		mutation operator
s	$:$	$I^\lambda \to I^\mu$		selection operator
$\Delta\sigma$	$\in$	$\mathbf{R}$		step-size meta-control
$\Delta\theta$	$\in$	$\mathbf{R}$		correlation meta-control
f	$:$	$\mathbf{R}^n \to \mathbf{R}$		objective function
g_j	$:$	$\mathbf{R}^n \to \mathbf{R}$	$j \in \{1, \ldots, q\}$	constraint functions
t	$:$	$I^\mu \to \{0, 1\}$		termination criterion

https://link.springer.com/chapter/10.1007/BFb0029787

(1+1)-ONLINE EA [5]

Very interesting technique for **run-time evolution**

There are only ever **two** individuals at any given time

Individual 1: Allowed to be evaluated for set amount time
Individual 2: Takes over after 1's time is done and is then evaluated for same time

Whoever wins, lives
New individual generated by mutation (or by randomness)

Interesting fact ... replicable results difficult, as 1 and 2 **don't necessarily start in the same place**

(1+1)-ONLINE EA

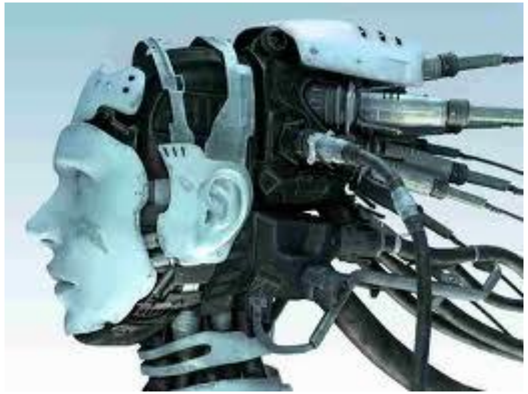
1) Where would this be useful?

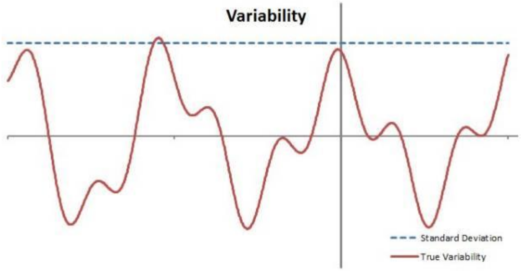
2) What is being sacrificed here?

(1+1)-ONLINE EA

1) Where would this be useful?

2) What is being sacrificed here?



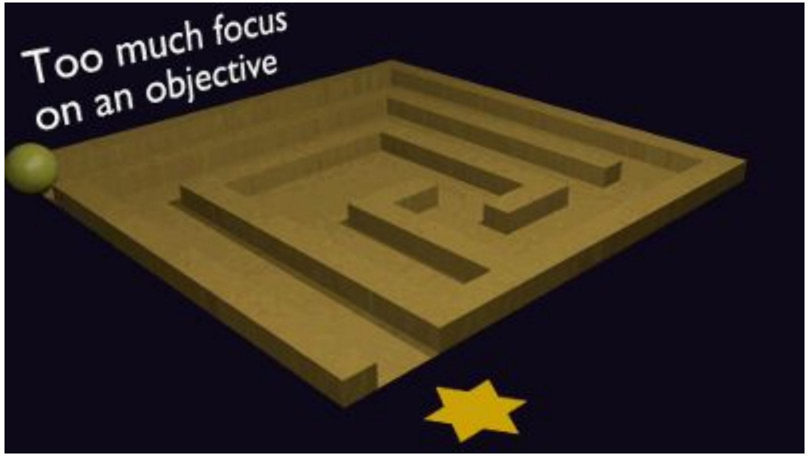


Novelty search [6]

Distinctly related to a GA

However

- We are no longer optimizing for the “best” solution, but the **most diverse set of solutions**



Novelty search

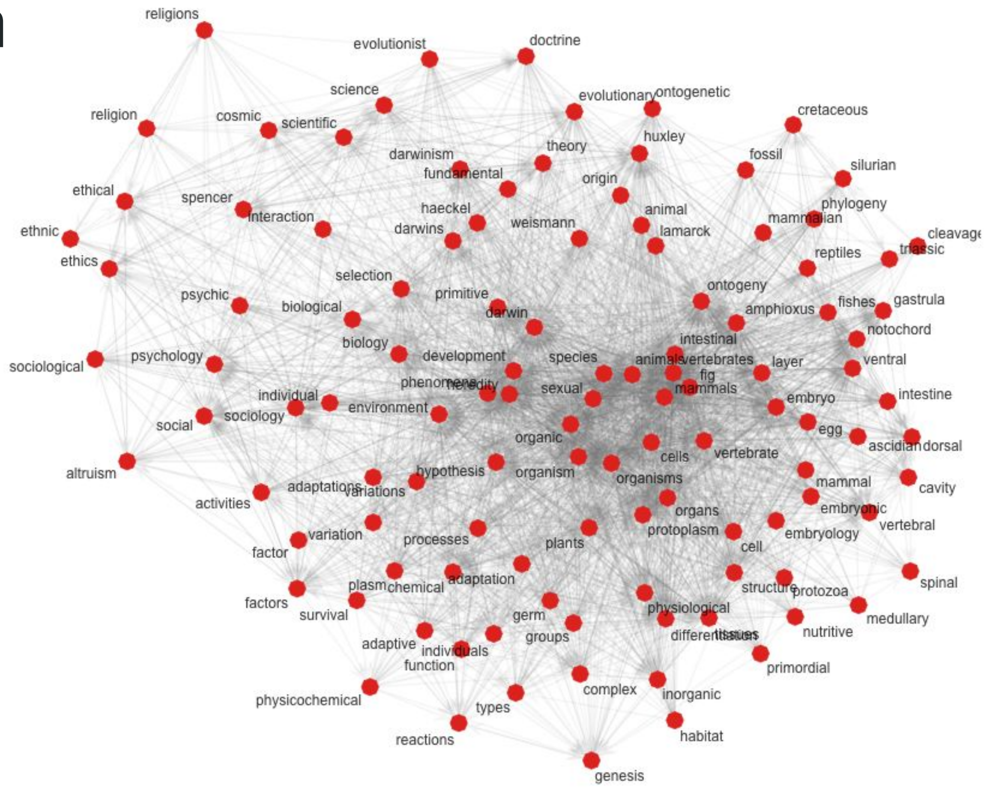
Works by **clustering** in the solution space:

1) (Generally) replace fitness function with novelty function
    a) Still ensure solutions are valid!

2) Maximize the **distance** between each solution
    a) Pair-wise distance → Euclidean or Manhattan distance, for example
    b) How different two genomes are

3) Maintain a **novelty archive** of the **most diverse solutions**
    a) Diversity = **largest distance**
    b) Novelty threshold → minimum novelty required to keep in archive

Novelty search



https://www.youtube.com/watch?v=JIQP15tt5AI

Multi-Objective Optimization

Applications to SE

Sorry, but this is going to be mainly my work

Ragnarok/Valkyrie

- Automatically generating SAS configurations (GA) to:
    - Violate as many requirements as possible
    - And then generate configurations to fix those violations

Fenrir

- Generate as many code paths as possible for an SAS using novelty search

SAGE (GP)

- Automatically compose software modules

Fenrir

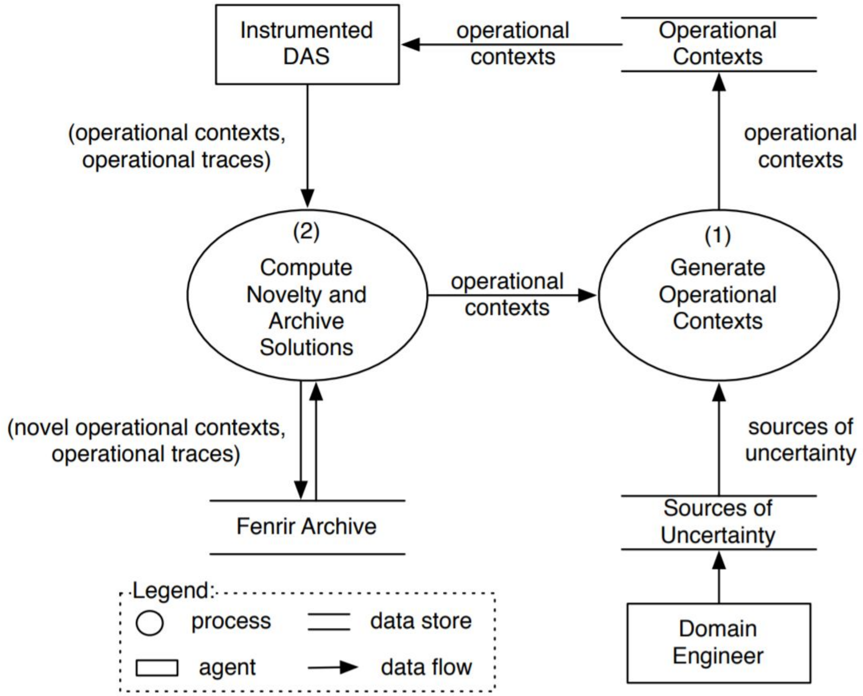


Fig. 3. DFD diagram of Fenrir process.

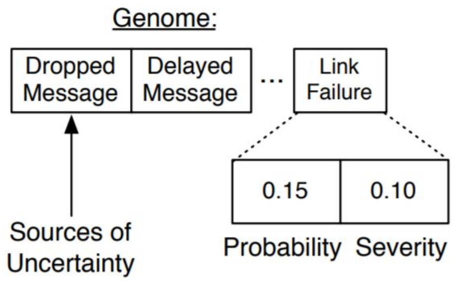
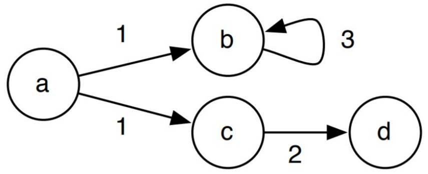


Fig. 4. Genome representation.

http://efredericks.net/ou/wp-content/papercite-data/pdf/fredericks.2013.ssbse.pdf

Fenrir

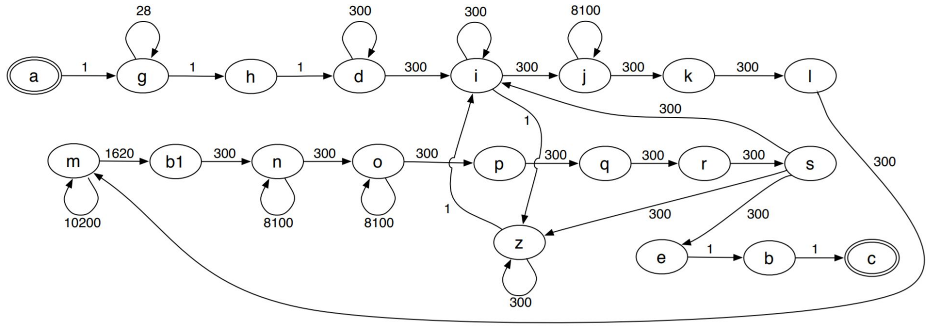


a. WCG

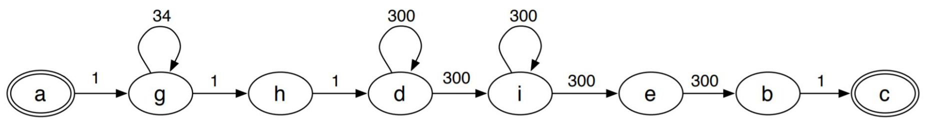
```
    void main() {
a:    wrapper(TRUE);
    }

    void wrapper(bool flag) {
      int i, k = 0;
      if (flag) {
b:          while (i < 3) ++i;
      else
c:          while (k < 2) --k;
d:      callFunction(i, k);
    }
```

b. Corresponding code.

Fig. 5. WCG Example.



(a) Execution Path 1



(b) Execution Path 2

Fig. 7. Unique execution paths.

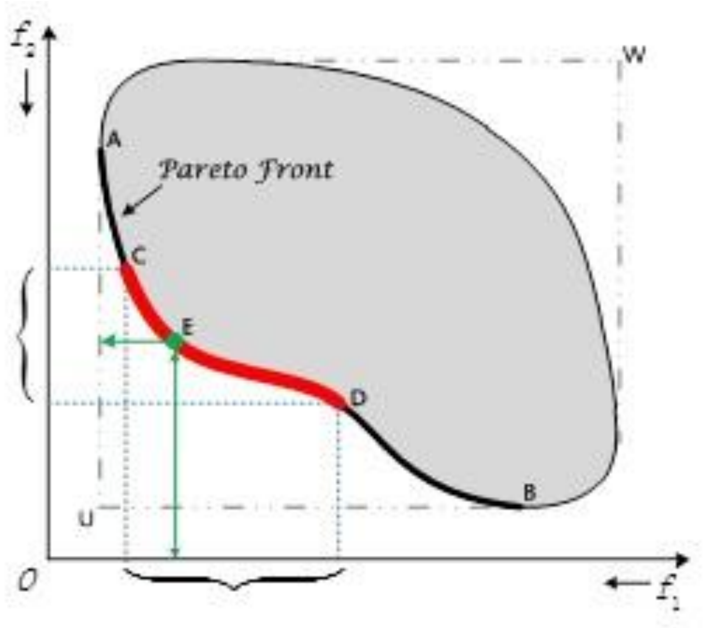
Competing Concerns

What happens if two (or more) objectives ***directly impact the other?***

Here, optimizing f_1 causes f_2 to perform worse

How do we deal with this?

- Multi-objective optimization
- Topic in and of itself, but worth looking at if you are interested in this!



Multi-objective optimization

Refs

https://www.slideshare.net/astensby/introduction-to-genetic-algorithms-and-evolutionay-co

[1] Holland, J.H.: Adaptation in Natural and Artificial Systems. MIT Press, Cambridge (1992)
[2] J. R. Koza. Genetic programming as a means for programming computers by natural selection. Statistics and Computing, 4(2):87–112, 1994
[3] O’Neill, M., Ryan, C.: Grammatical evolution. IEEE Transactions on Evolutionary Computation 5, 349–358 (2001)
[4] T. Perkis. Stack-based genetic programming. In Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conf. on, pages 148–153. IEEE, 1994.
[5] N. Bredeche, E. Haasdijk, and A.E. Eiben. On-line, on-board evolution of robot controllers. In Pierre Collet, Nicolas Monmarch´ e, Pierrick Legrand, Marc Schoenauer, and Evelyne Lutton, editors, Artificial Evolution, volume 5975 of Lecture Notes in Computer Science, pages 110–121. Springer Berlin Heidelberg, 2010.
[6] Joel Lehman and Kenneth O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. In Proceedings of the Eleventh International Conference on Artificial Life (ALIFE XI). MIT Press, 2004.