Darwin

Neuroevolution & Evolutionary Algorithms Framework

Leonard Mosescu

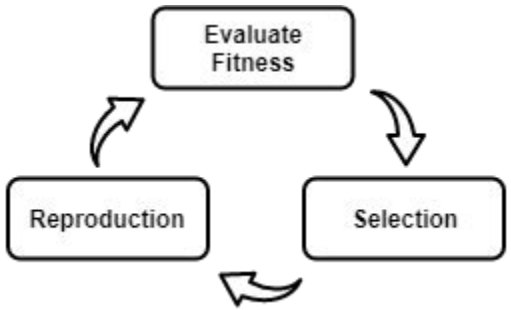
Outline

Evolutionary Algorithms

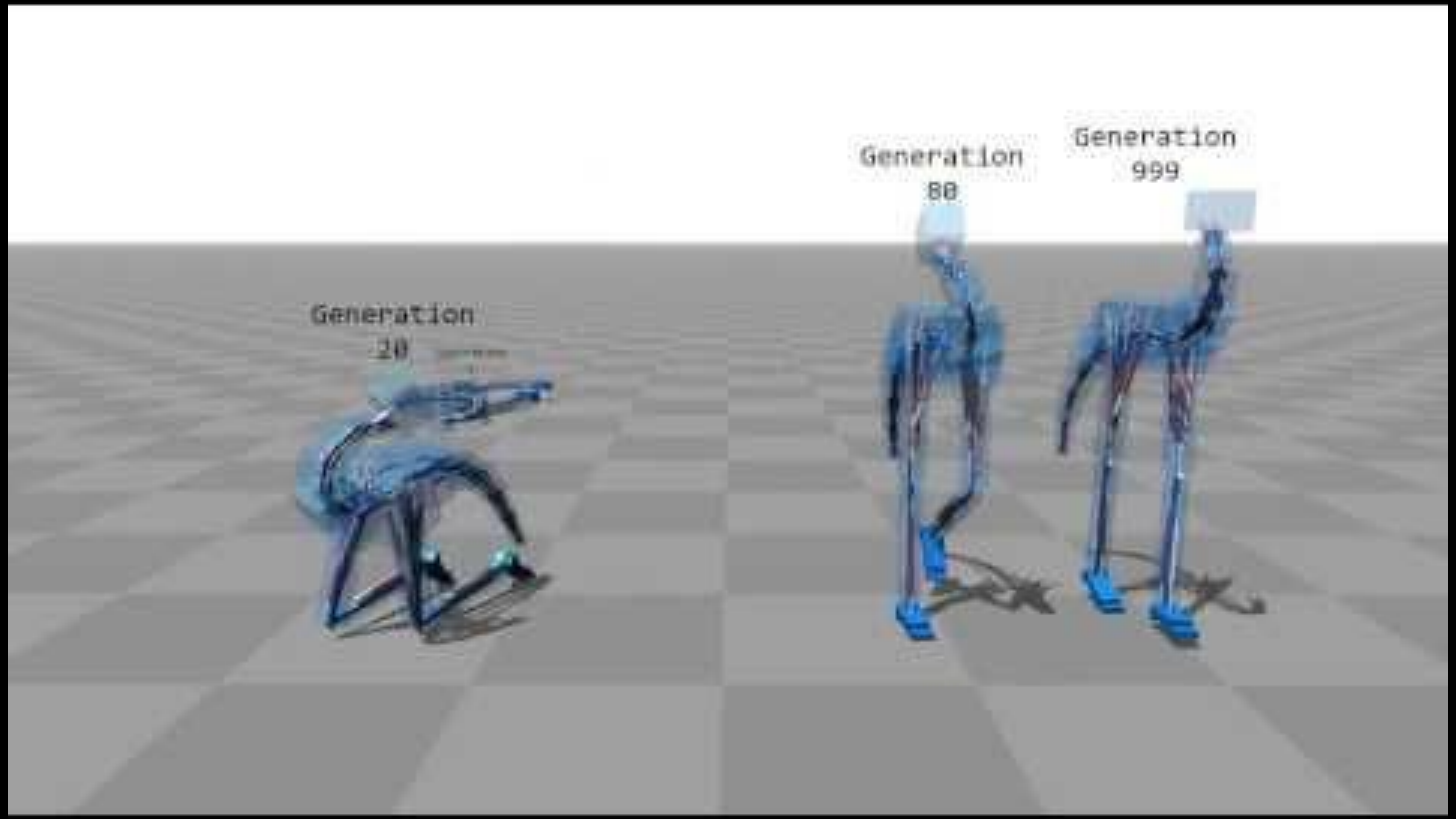
Darwin Framework Overview

Design & Implementation

Evolutionary Algorithms




Score : 29
Generation : 68
Alive : 50 / 50

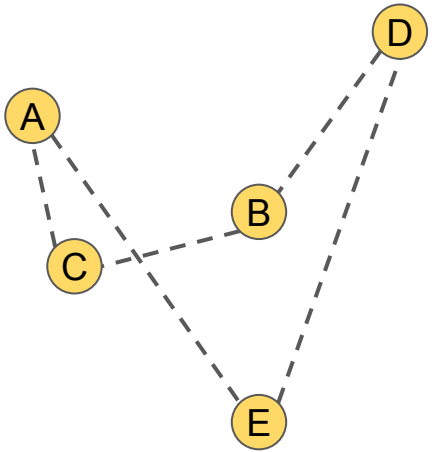

Generation
20
Generation
80
Generation
999

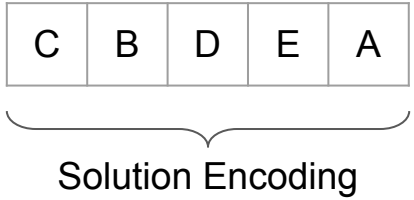
Evolutionary Algorithms Template

Evolution loop (one generation)

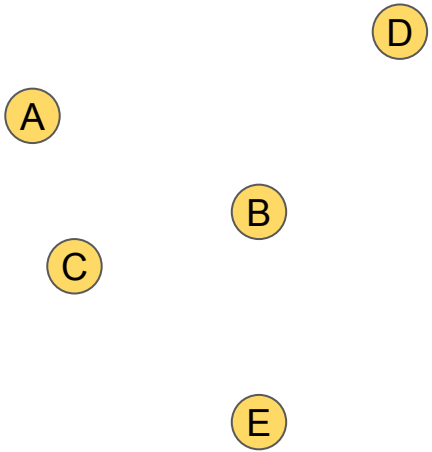
```
initialize_population
while(not satisfied):
    for_each individual:
        evaluate_fitness
    create_next_generation:
        select_parents
        use crossover & mutation to generate children
```

Example: Travelling Salesman Problem





Example: Travelling Salesman Problem



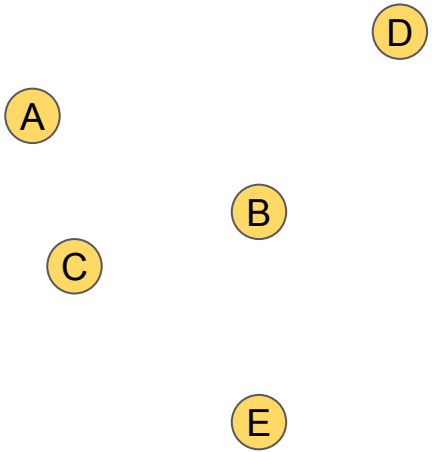
C	B	D	E	A

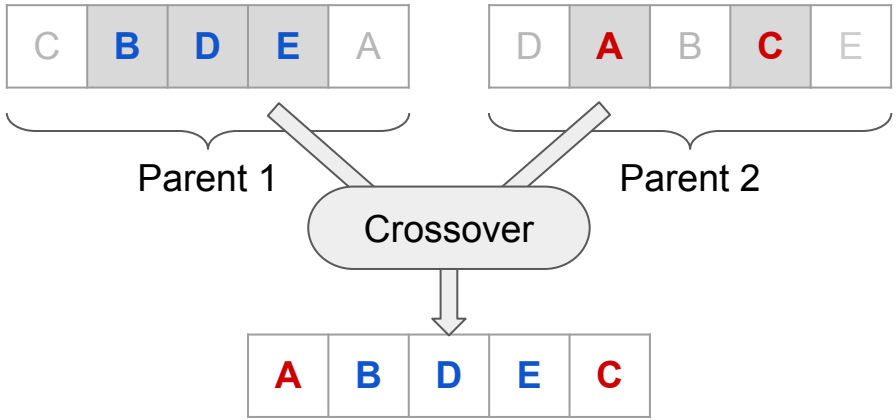
Parent 1

D	A	B	C	E

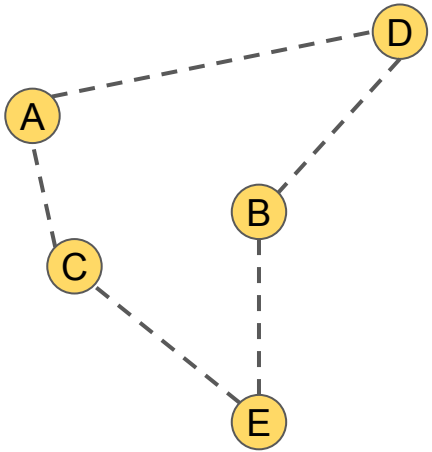
Parent 2

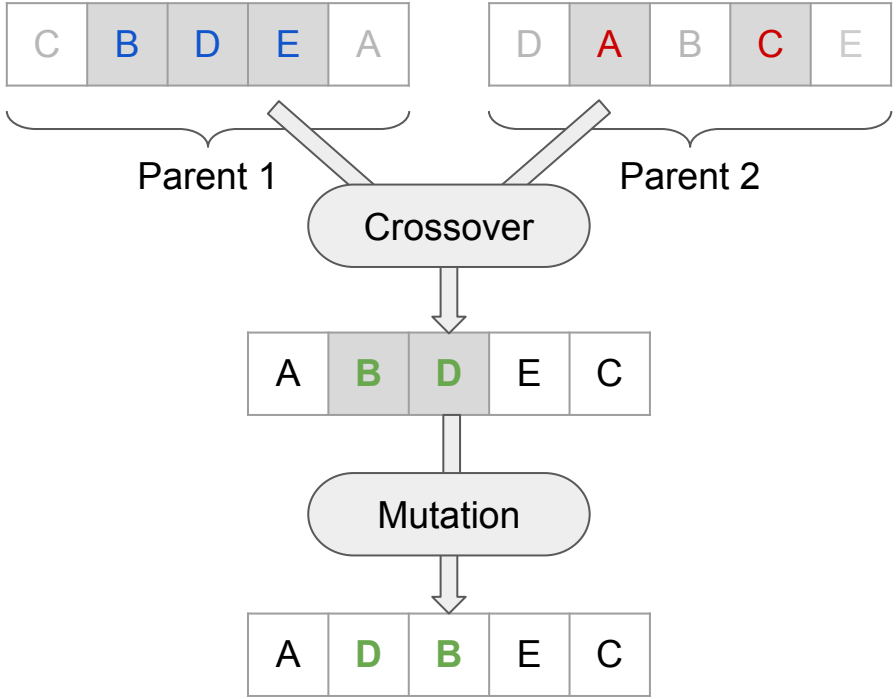
Example: Travelling Salesman Problem





Example: Travelling Salesman Problem

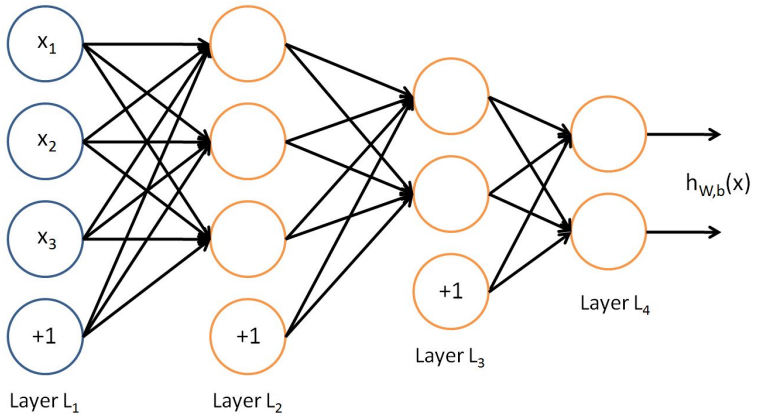




Evolutionary Algorithms Applications

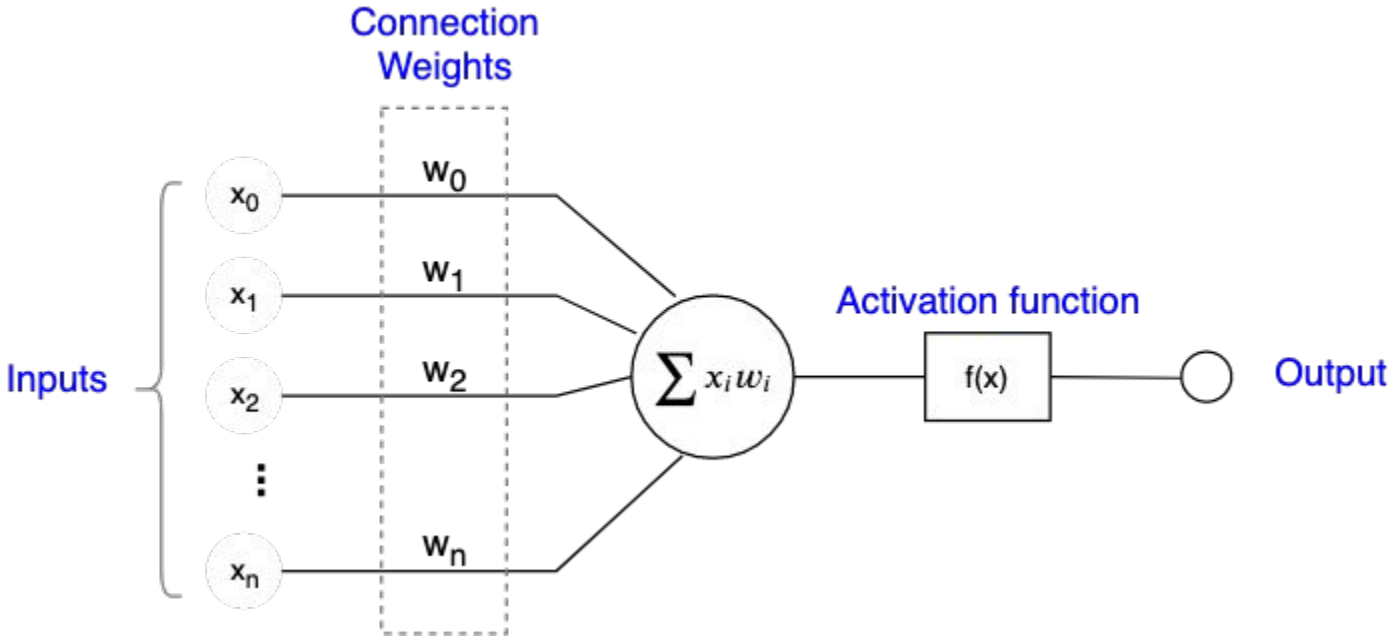
- VLSI design (routing / placement)
- Fuzzy testing
- Floor plan design
- Resource allocation
- ...
- Robotics
- Artificial Life & AI

Artificial Neural Networks (ANNs)



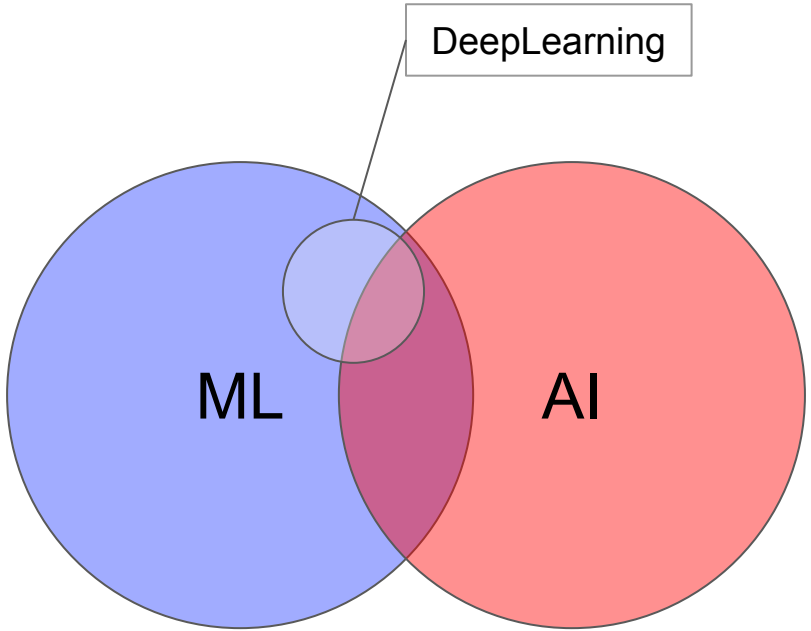
Source

An (Artificial) Neuron



Neural Networks Topologies

AI & Machine Learning

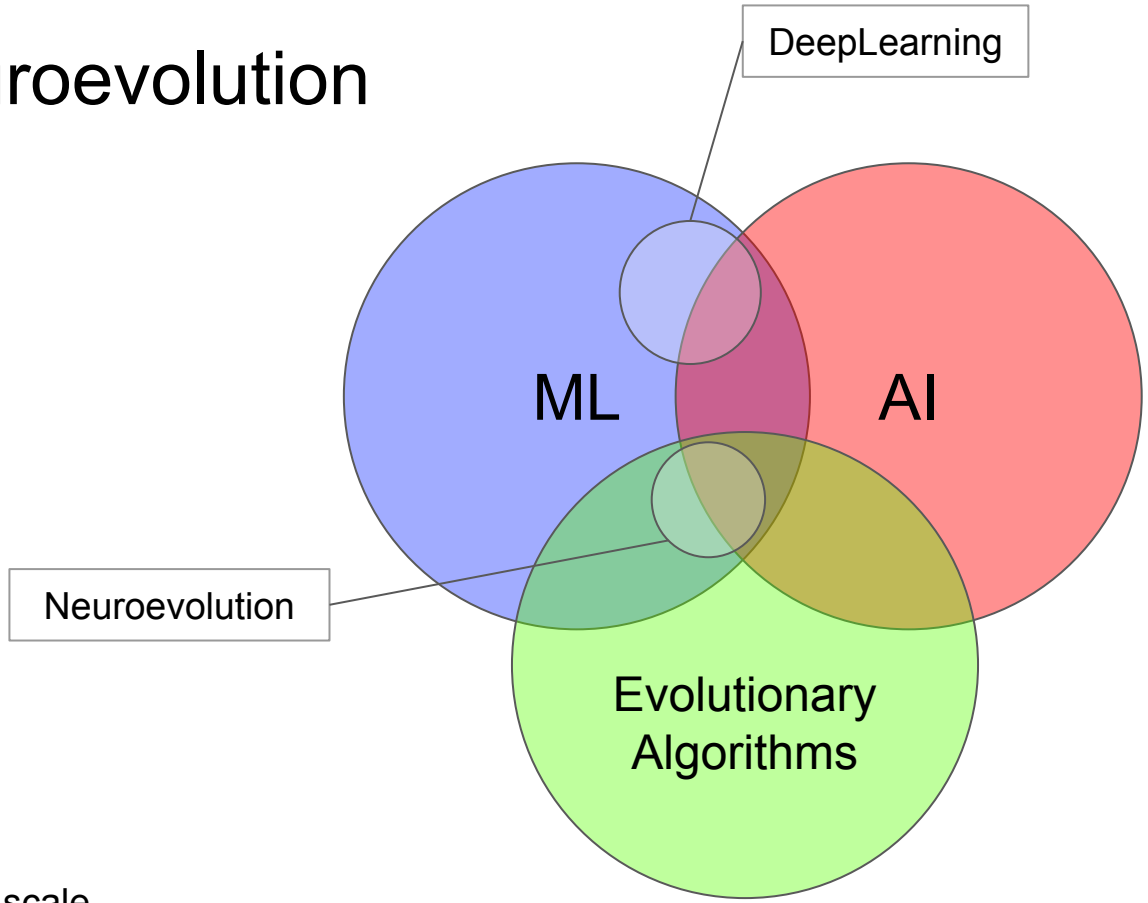


* Not to scale

Evolutionary Algorithms +
Artificial Neural Networks =

Neuroevolution

Neuroevolution



* Not to scale

Darwin Framework Overview

https://github.com/tlemo/darwin

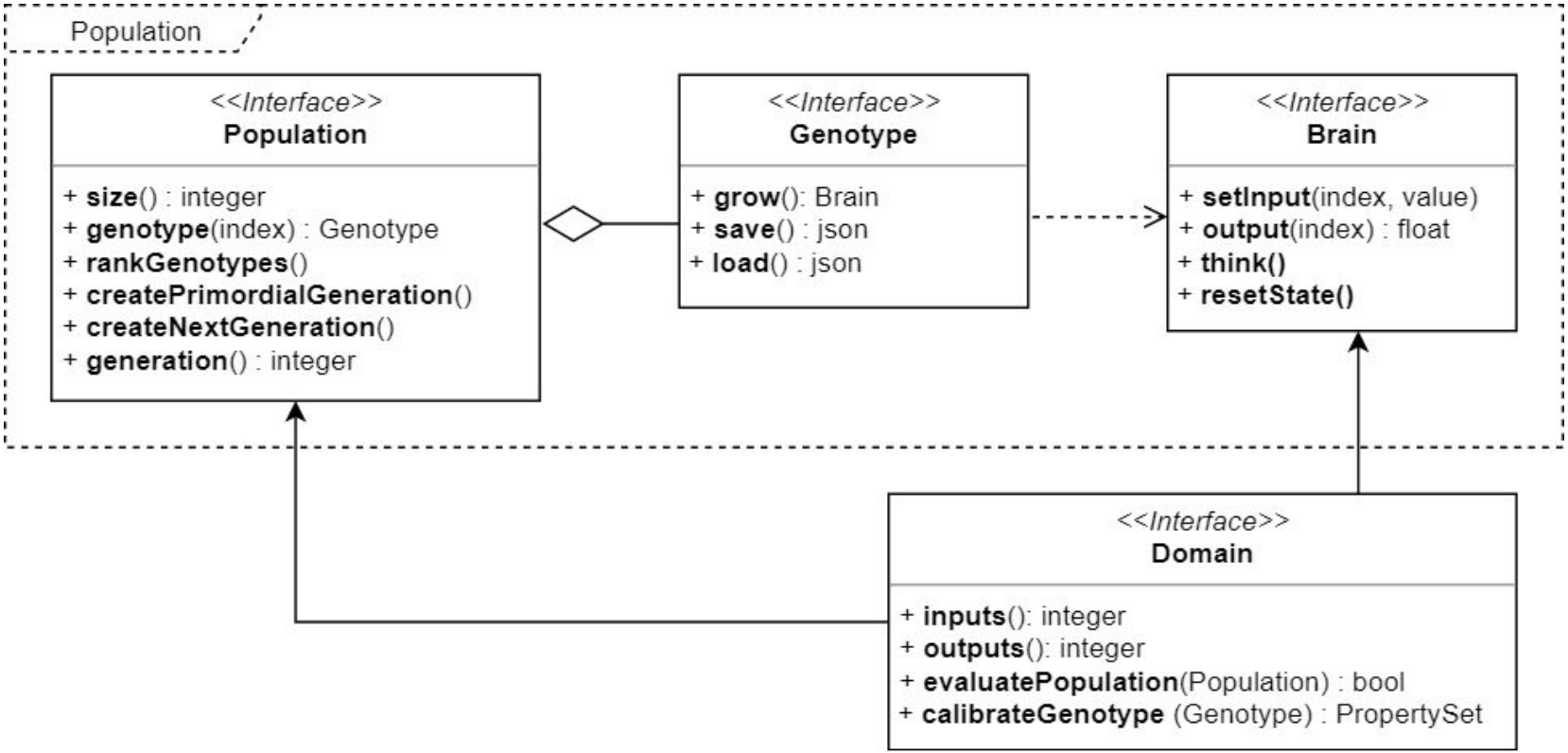
Motivation

- More time building fun stuff, less time on repetitive stuff
- N domains x M algorithms -> N + M
- Structured experimentation
- Amortize the cost of tool building

Key EA Abstractions

- Solution encoding (genotype)
- Solution materialization (phenotype)
- Population
- Fitness Evaluation (domain)

Populations and Domains



Evolution Loop

```
population->createPrimordialGeneration(population_size);
while (domain->evaluatePopulation(population)) {
    population->rankGenotypes();
    population->createNextGeneration();
}
```

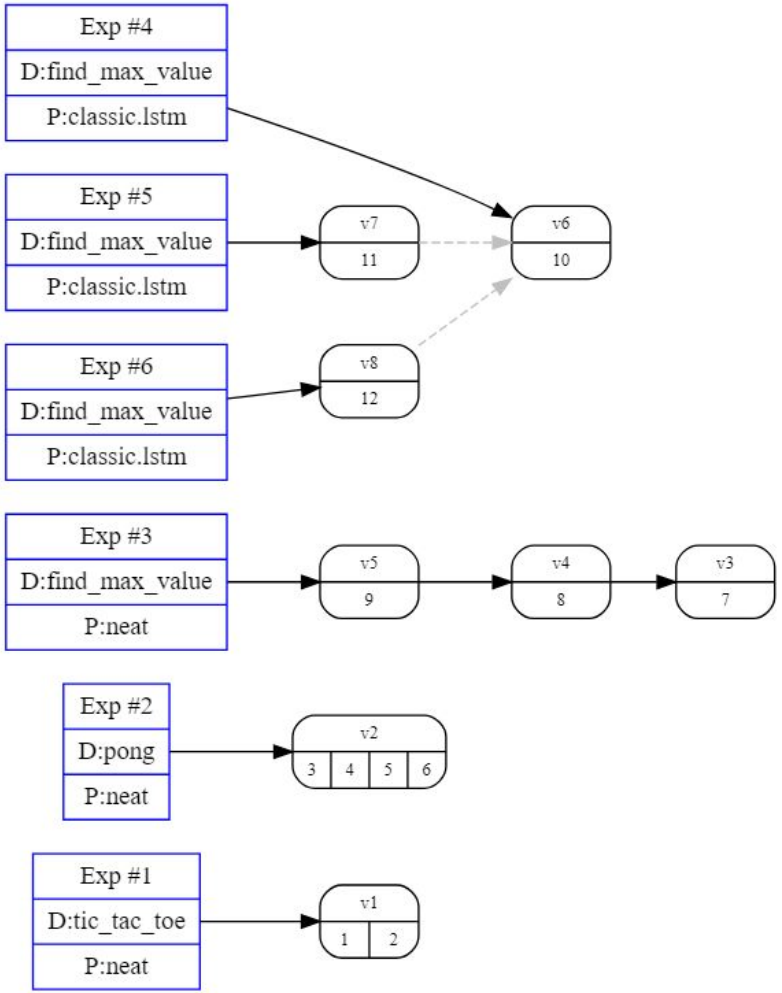
Darwin Universe Database

Universe

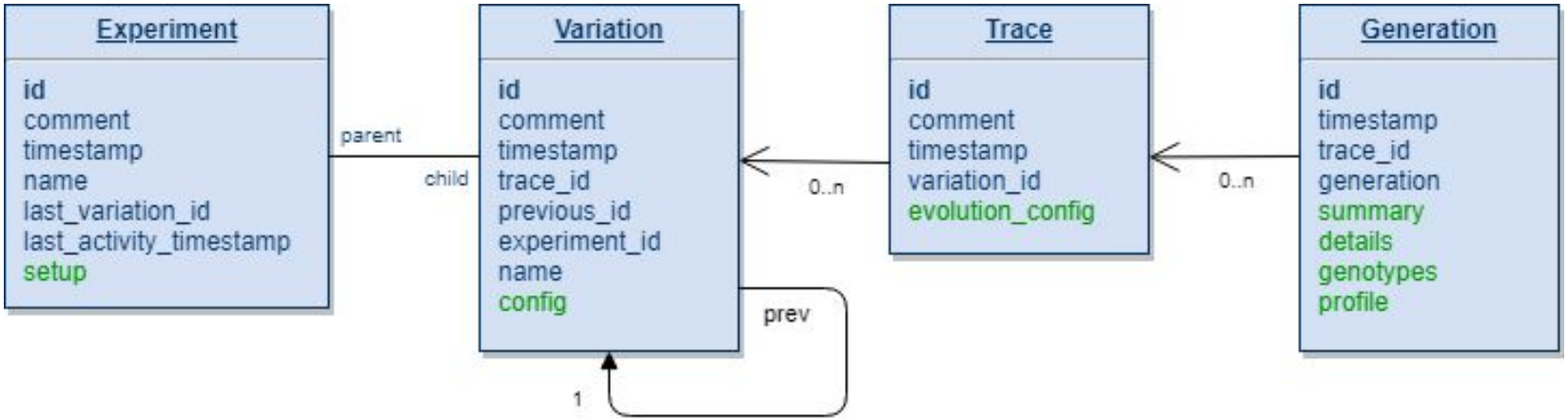
Experiment

Variation

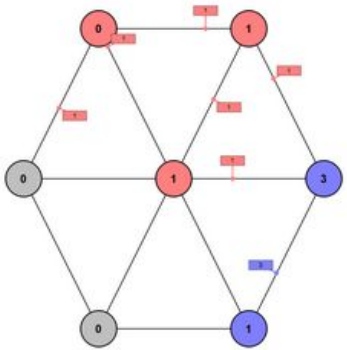
Trace

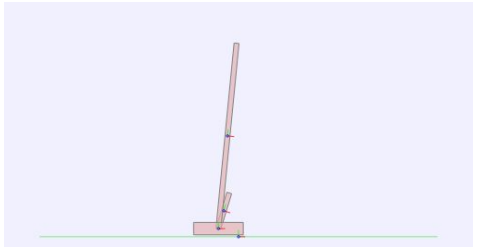


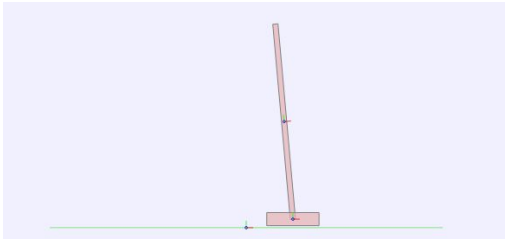
Darwin Universe Database

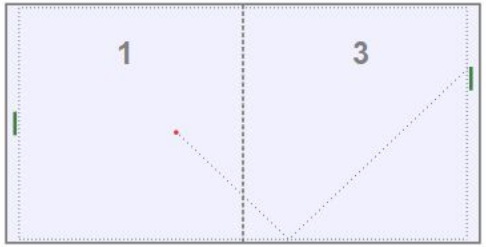


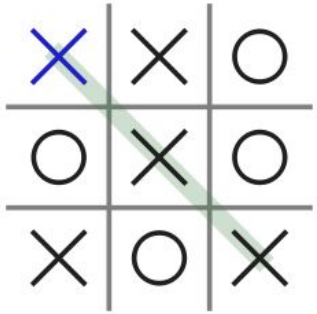
Built-in Domains

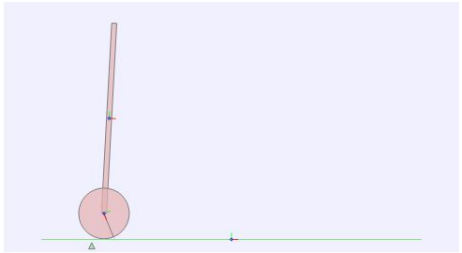


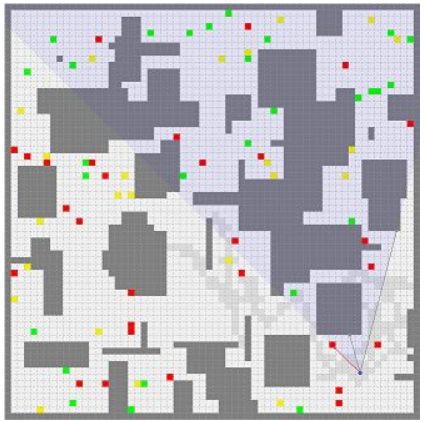


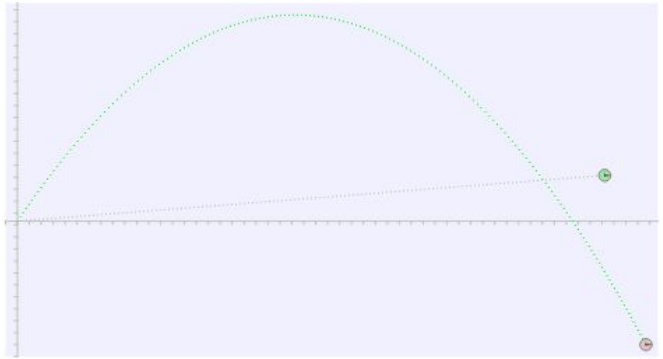












Built-in Populations

Conventional Neuroevolution (CNE)

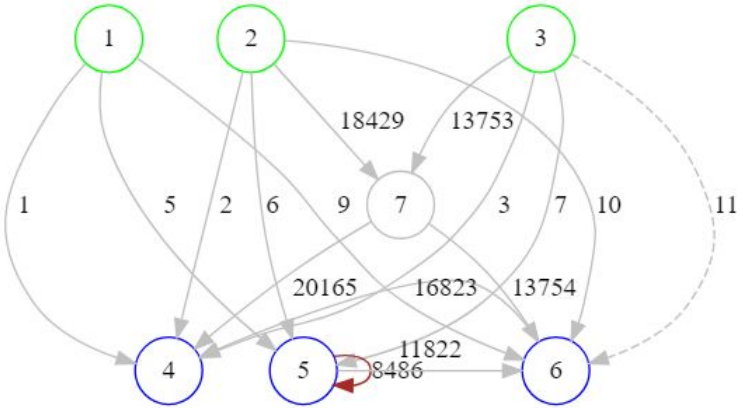
- Feedforward
- LSTM
- RNN

Neuroevolution of augmenting topologies (NEAT)

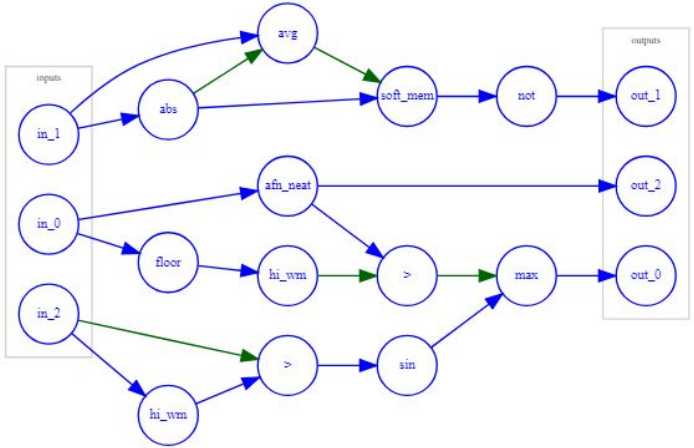
Cartesian Genetic Programming (CGP)

Experiment Results: Visualizing Genotypes

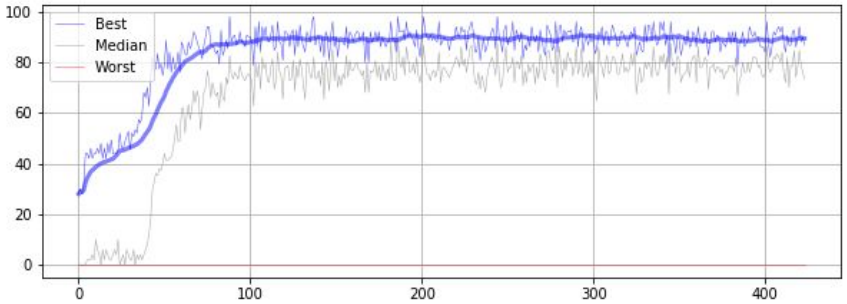
NEAT Artificial Neural Network

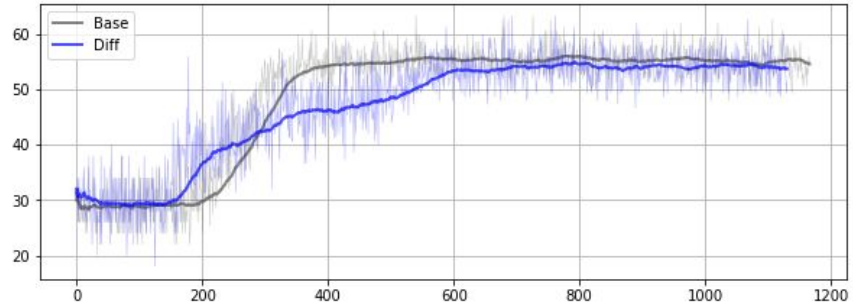


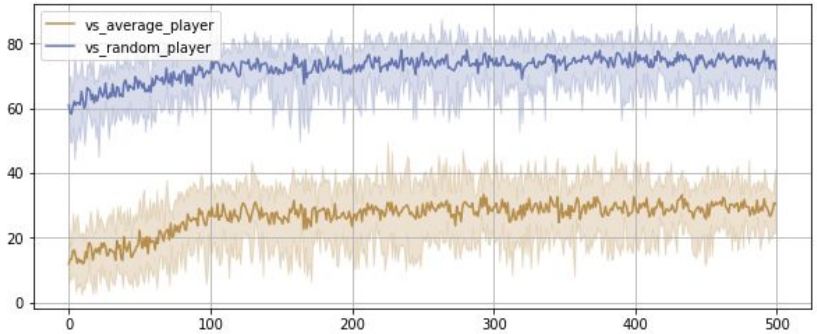
Cartesian Genetic Programming

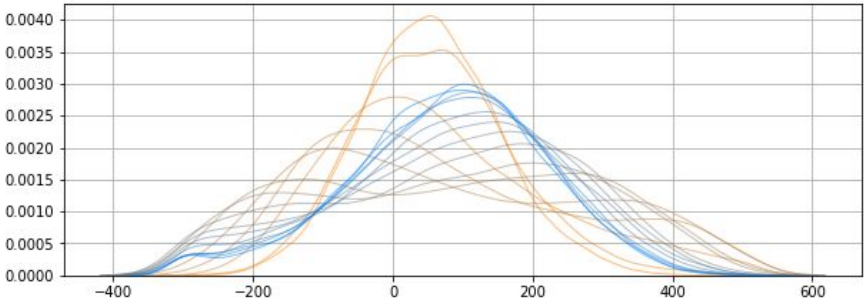


Experiment Results: Fitness

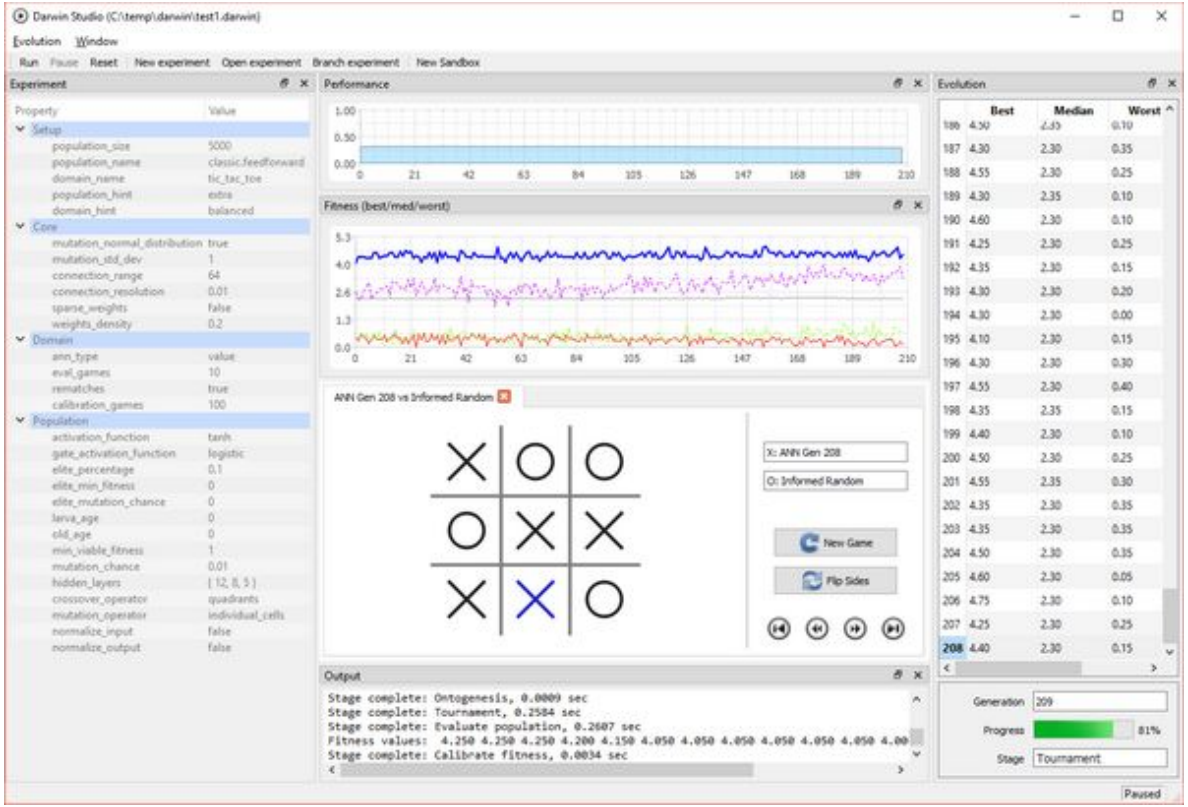








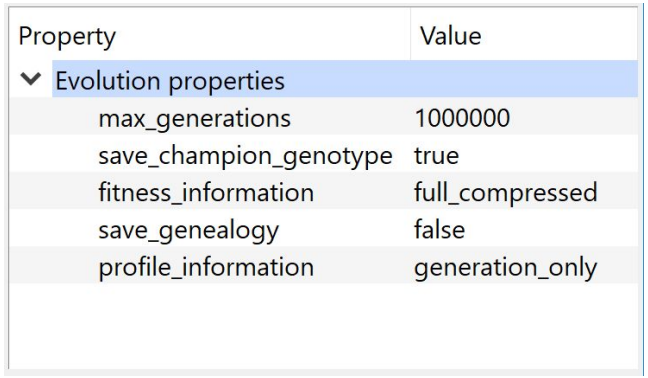
Darwin Studio



Design & Implementation

core::PropertySet

- Basic reflection for structures
  - Needed for UI & serialization (and a bit of type erasure)
- As cheap as a plain struct when accessing members
- Built-in to/from JSON
  - fromJson() is not strict
- Dedicated UI component: core_ui::PropertiesWidget
- See also: core::PropertySetVariant



core::PropertySet

```
struct Config : public core::PropertySet {
 PROPERTY(max_value, int, 100, "Maximum value");
 PROPERTY(resolution, float, 0.3f, "Display resolution");
 PROPERTY(name, string, "darwin", "Name");
 PROPERTY(layers, vector<int>, {}, "Hidden layer sizes");
};
```

Defining properties

```
Config config;
// set Config::max_value
config.max_value = 75;
// read Config::name
auto name = config.name;
```

Direct member read / write

```
void printProperties(const core::PropertySet* config) {
 // enumerate properties
 for (const auto& property : config->properties()) {
   // read the property name and value as strings
   core::log("%s = %s\n", property->name(), property->value());
 }
}
```

Runtime reflection

Using “cutting edge” C++

- Coding style based on Chromium style (Google style)
  - … but with no restrictions
- May use any feature which is supported across all the target platforms
  - Currently this is C++ 17
- Examples
  - Function return type deduction [C++14]
  - Generic lambdas [C++14]
  - Lambda capture expressions [C++14]
  - [[maybe_unused]] [C++17]
  - Fold expressions [C++17]
  - Structured binding declarations [C++17]
  - std::filesystem (std::experimental::filesystem) [C++17]
  - std::optional [C++17]

Error Handling

- Assert everything!
- … and always check your assertions!
- Fail fast!
- C++ exceptions

Generating Random Numbers

- C++11’s <random>
- Seeding & Entropy
- Generators
- Performance

Cheap Dependencies Have a High Cost

- Costs
  - Identify requirements up front
  - Research and evaluate options
  - Integration
  - Learn APIs
  - Implementation surprises
  - API seams
- High bar for third party code
  - Compatible license
  - Platform support
  - Build system, test, support, documentation, dependencies, etc
  - API surface / functionality ratio.

Third Party Libraries In Darwin

- Google-style third_party: dependencies are part of the source tree
- Integration tests for third party libraries
- No package manager
- Git submodules
- Start with a custom solution & shop for a mature library later

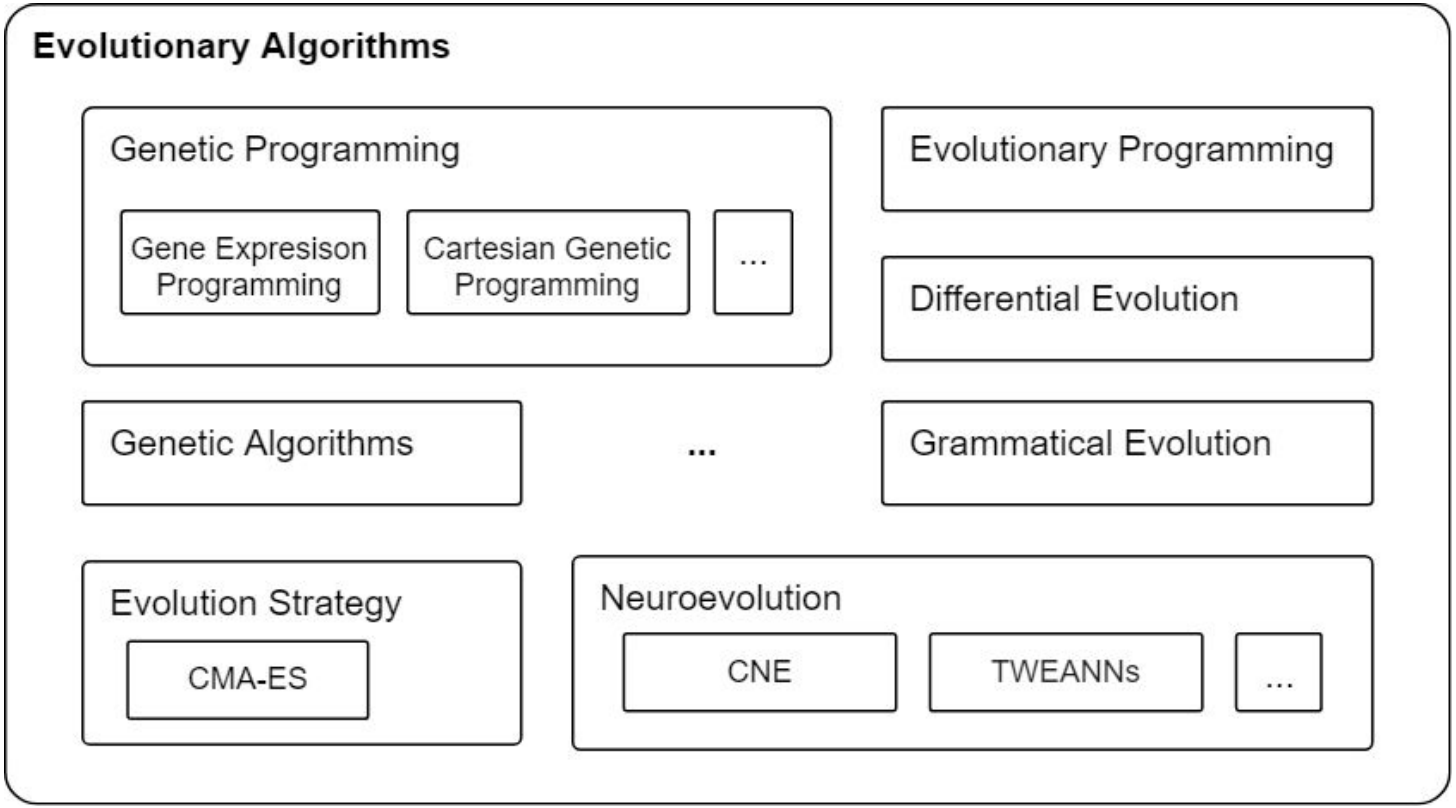
References

- Genetic Programming: An Introduction
- The Selfish Gene
- Blondie24: Playing at the Edge of AI
- A Field Guide to Genetic Programming
- Darwin project on GitHub

QUESTIONS?

Bonus Slides

Evolutionary Algorithms Taxonomy



Why Evolutionary Algorithms?

- Simple, generic and reusable structure
- Applicable to complex domains
  - No differentiable landscape requirement compared to Gradient Descent
- Natural expression of the domain problem
- Scales to huge search spaces
- Potential for creativity & emergence
- Can be resilient to local optima traps*

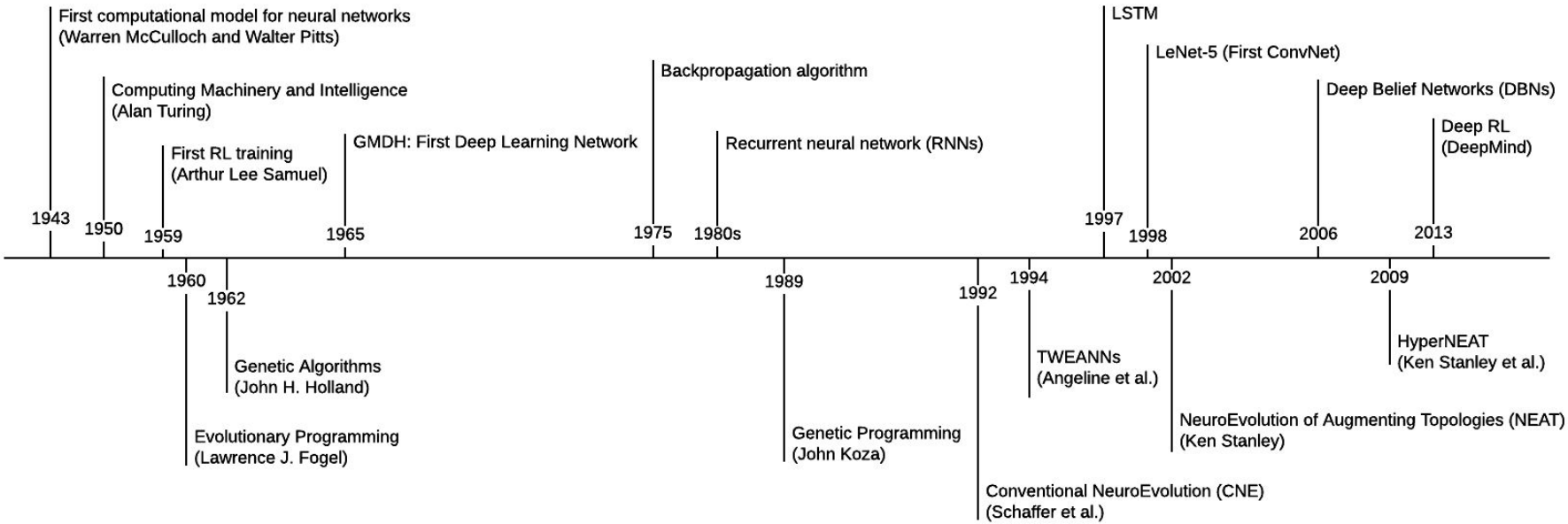
Why ***not*** Evolutionary Algorithms?

- Outperformed by specialized algorithms
- Designing genetic encodings and operators
- Blackbox
- Can be hard to debug and interpret solutions
- Stochastic nature
- Resilient to implementation mistakes

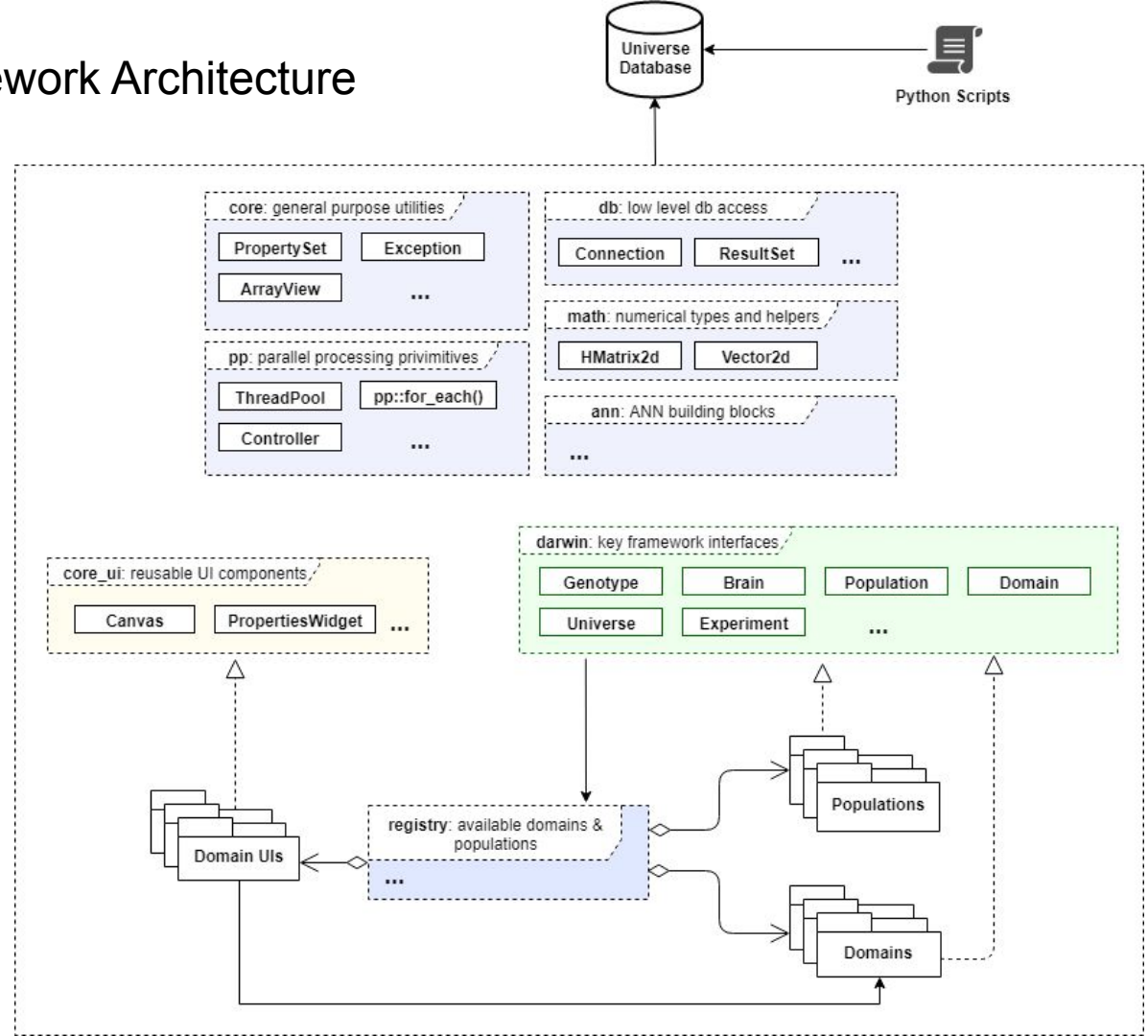
Isn’t EA the same as Reinforcement Learning?

- Agent / Environment
- Sparse Rewards
- Model-based vs. Model-free RL
- Timescales: Generation vs. Episode
- DeepRL comes with the same DL limitations

Timeline: Neuroevolution & Machine Learning



Darwin Framework Architecture



Darwin Framework: Core Pillars

- First class support for Evolutionary Algorithms concepts
- Capable of running interesting experiments on easily available hardware
- Complete package
- Structured approach to experimentation
- Cross-platform with minimal external dependencies

Language Choice: C++

- Mature cross-platform tooling
- Reduced dependencies / layers
- Qt is still one of the best desktop GUI toolkits
- Good selection of native, 3rd party libraries
- C++ provides one of the best resource management solutions

Cons:

- Still missing basic features: reflection, run-time code generation
- Poor build model
- Too much flexibility in the wrong places (lack of universal conventions)
- Complexity / cognitive load

Misc

Coding style

- Adopted a mature, well documented style (Chromium / Google style)
- Use clang-format!

Tests

Documentation

- Doxygen is still the most flexible documentation tool for C++
- github.io is a great documentation hosting platform

Build System

How evolutionary selection can train more capable self-driving cars

"*Now, Waymo, in a research collaboration with DeepMind, has taken inspiration from Darwin's insights into evolution to make this training more effective and efficient.*

...

Population Based Training (PBT) enabled dramatic improvements in model performance. ... A chief advantage of evolutionary methods such as PBT is that they can optimise arbitrarily complex metrics" (DeepMind and Waymo)

The gap between research and application

"*Similar to what happened in Computer Vision, the progress in RL is not driven as much as you might reasonably assume by new amazing ideas. In Computer Vision, the 2012 AlexNet was mostly a scaled up (deeper and wider) version of 1990's ConvNets. Similarly, the ATARI Deep Q Learning paper from 2013 is an implementation of a standard algorithm [...]*" (Deep Reinforcement Learning: Pong from Pixels, May 2016)

The Law of Uphill Analysis and Downhill Synthesis
(Braitenberg's law)

- Could a Neuroscientist Understand a Microprocessor?

- "*in order to make a perfect and beautiful machine, it is not requisite to know how to make it*"

- "*To my way of thinking there is still something mysterious about evolution, with its apparent 'groping' towards some future purpose. Things at least seem to organize themselves somewhat better than they 'ought' to, just on the basis of blind-chance evolution and natural selection*" (Sir Roger Penrose)