

# Lessons Learned Developing Evolutionary Algorithms in C++

Manlio Morini, *software developer*  
[morini@eosdev.it](mailto:morini@eosdev.it)



Associazione  
Italiana per  
l'Intelligenza  
Artificiale

C++Day Pavia 2018



[www.italiancpp.org](http://www.italiancpp.org)



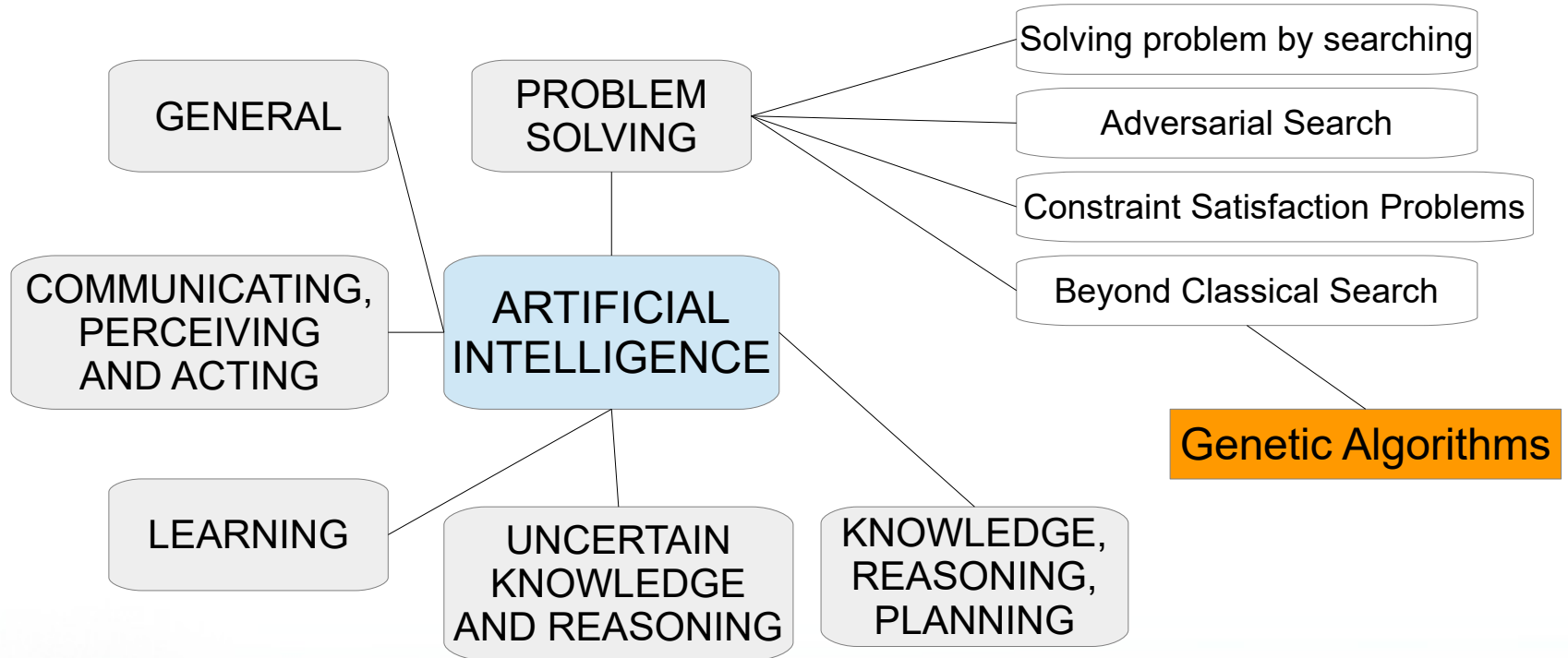
# Contents

- 1) Understanding the big picture  
(with code interludes)
- 2) Evolutionary frameworks

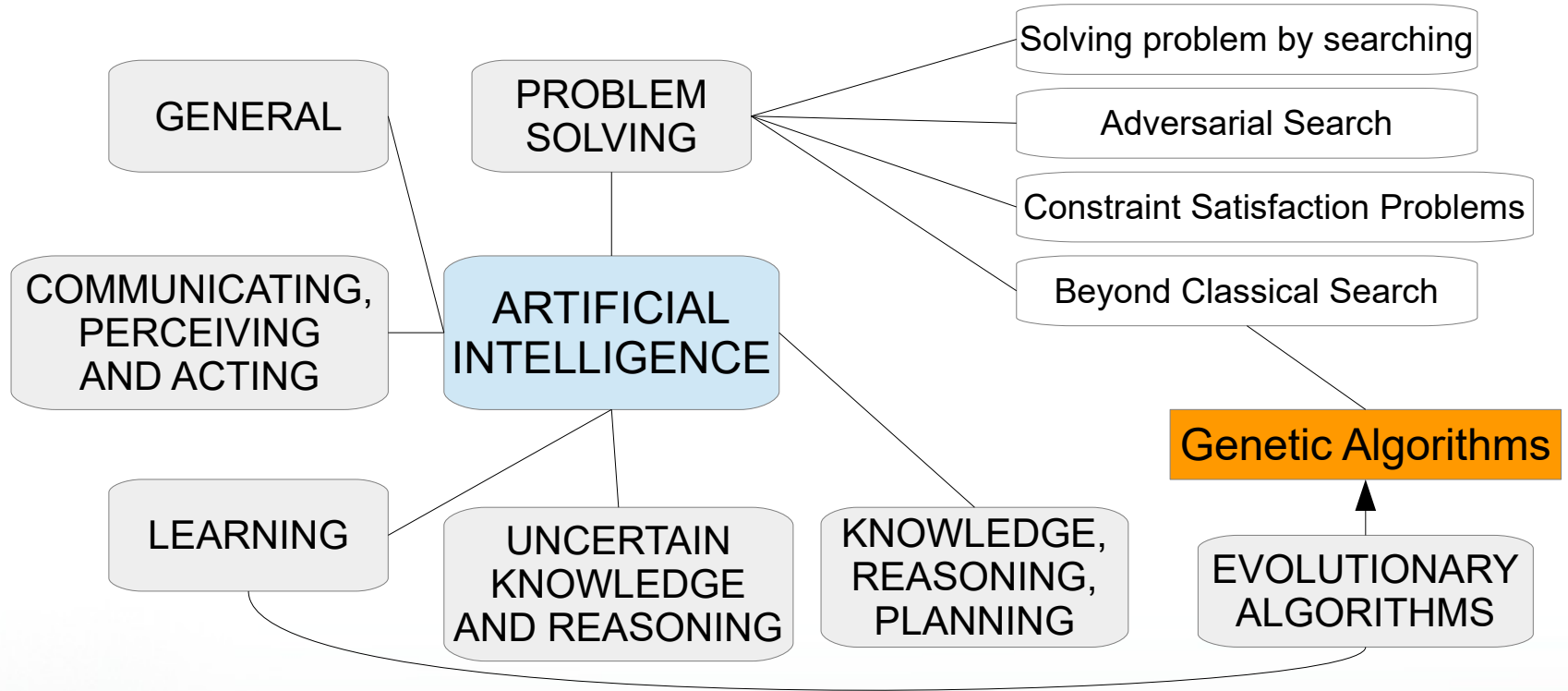


# UNDERSTANDING THE BIG PICTURE

# AI Mind Map (ACM – AI Norvig Russell)



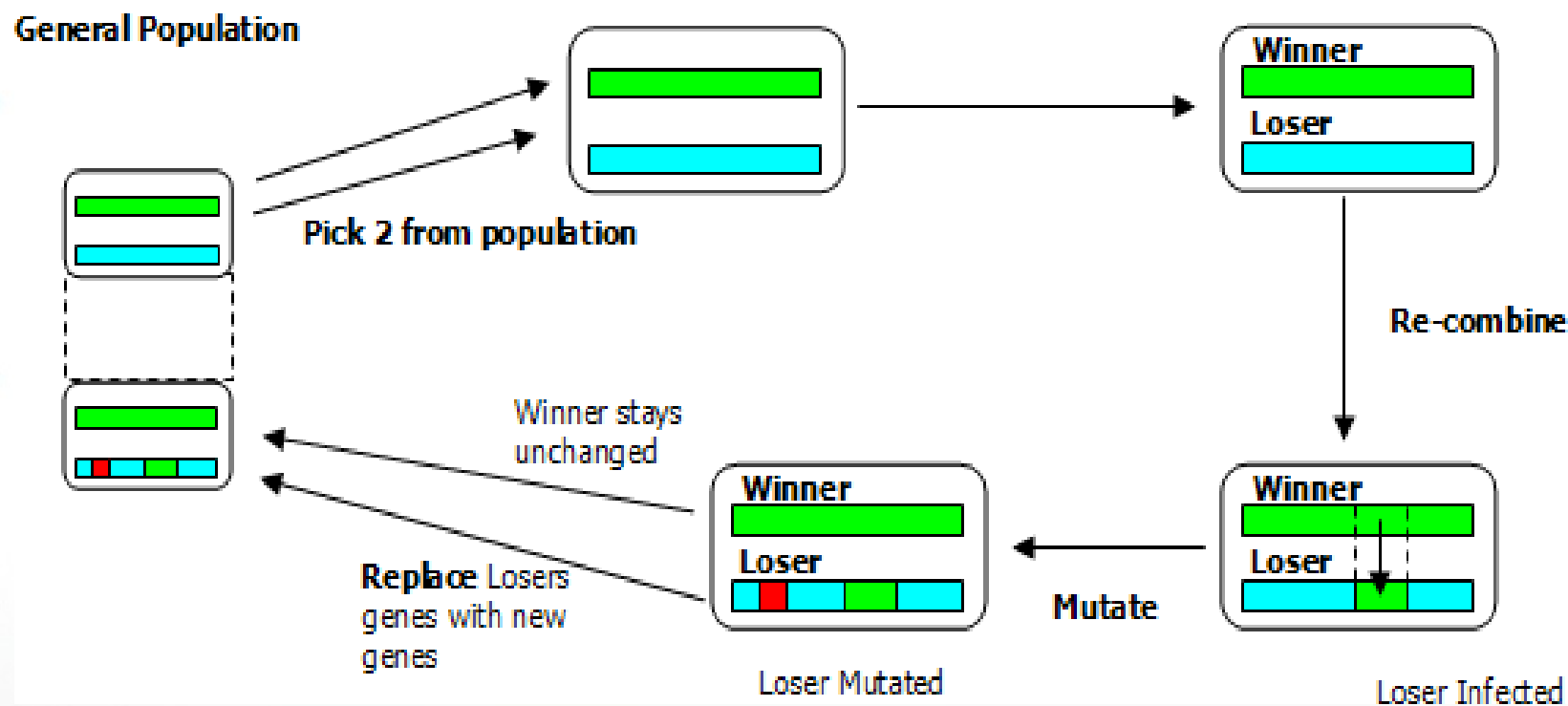
# AI Mind Map (ACM – AI Norvig Russell)



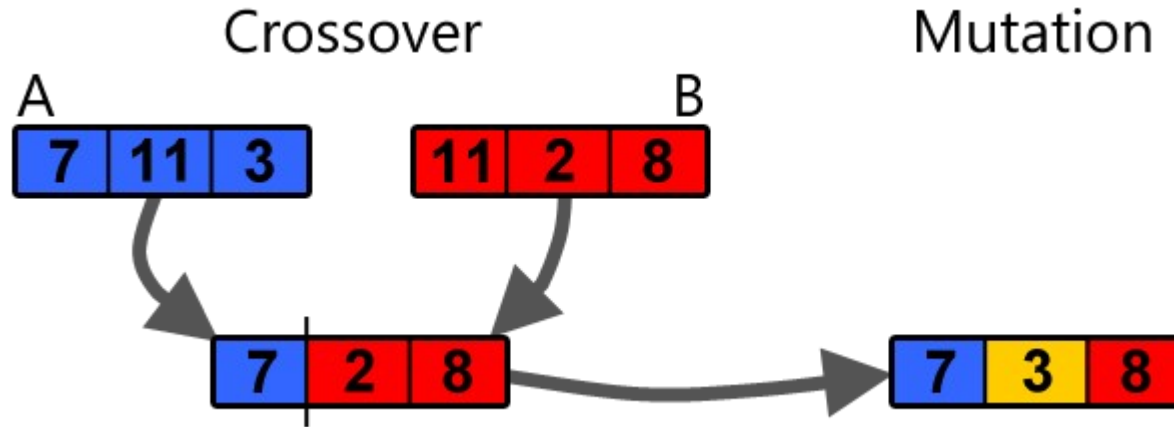


# Distinctive features

- A **population-based** meta-heuristic optimization algorithm. Candidate solutions to the optimization problem play the role of **individuals** in a population.
- Mechanisms **inspired by biological evolution** (e.g. reproduction, mutation, recombination and selection).
- A **fitness function** determines the quality of the solutions. Evolution of the population then takes place after the repeated application of the above operators.

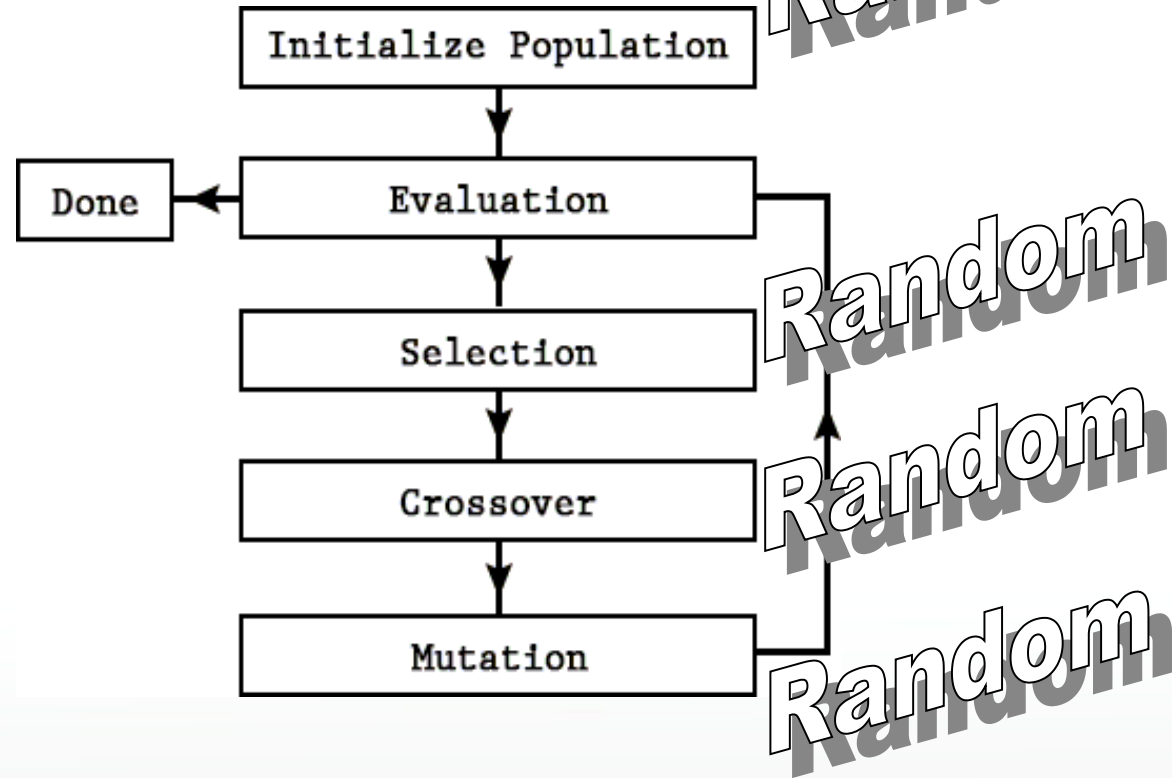


# Crossover / mutation





# EA flowchart



# 1<sup>st</sup> Code Interlude



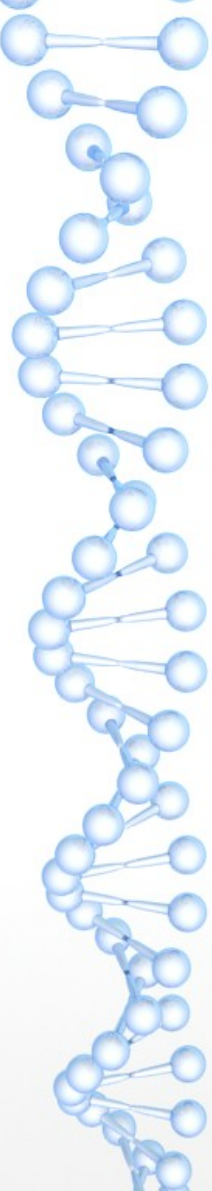
Scott Adams - <https://dilbert.com/>



# Usual approach

```
#include <random>

int between(int min, int max)
{
    thread_local std::mt19937 rng; // ← std::random_device{}()
    std::uniform_int_distribution<int> d(min, max);
    return d(rng);
}
```



# XOSHIRO 256\*\* - Source

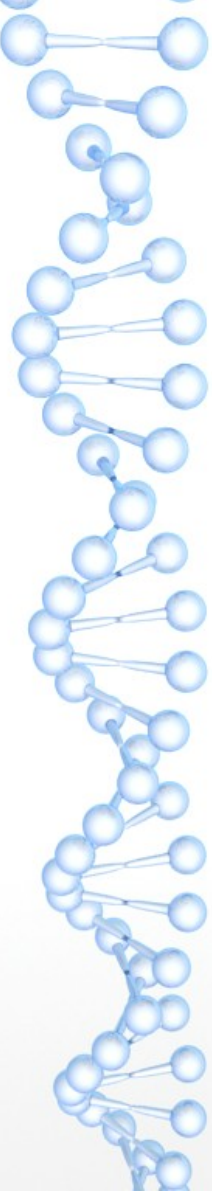
```
static uint64_t s[4];

uint64_t next(void) {
    const uint64_t result_starstar = rotl(s[1] * 5, 7) * 9;
    const uint64_t t = s[1] << 17;

    s[2] ^= s[0];
    s[3] ^= s[1];
    s[1] ^= s[2];
    s[0] ^= s[3];

    s[2] ^= t;
    s[3] = rotl(s[3], 45);

    return result_starstar;
}
.
// Written in 2018 by David Blackman and Sebastiano Vigna (vigna@acm.org)
// http://xoshiro.di.unimi.it/
```



# XOSHIRO 256\*\* - Features

- All-purpose, rock-solid generator.
- Excellent speed.
- Internal state large enough (256 bits) for any parallel application.
- Passes **BigCrush** suite of tests / tests for Hamming-weight dependencies



# XOSHIRO256\*\* vs Mersenne Twister

PRNG	Footprint (bits)	Period	BigCrush Failure	cycles/B
xoshiro256**	256	$2^{256} - 1$	-	0.41
MT19937-64	20032	$2^{19937} - 1$	LinearComp	1.26

- $++x$  has a period of  $2^k$  for any  $k \geq 0$  provided that  $x$  is represented using  $k$  bits.
- Heuristic rule of thumb:
  - if you need to use  $t$  values, you need a generator with period at least  $t^2$ ;
  - if you run  $n$  independent computations starting at random seeds, the sequences used by each computation should not overlap.



# XOSHIRO256\*\*

## How to plug into a random number distribution

- **UniformRandomBitGenerator**
- [https://en.cppreference.com/w/cpp/named\\_req/UniformRandomBitGenerator](https://en.cppreference.com/w/cpp/named_req/UniformRandomBitGenerator)
- A function object returning unsigned integer values such that each value in the range of possible results has (ideally) equal probability.
- Not intended to be used as random number generators (just a source of random bits).
- **RandomNumberEngine**
- [https://en.cppreference.com/w/cpp/named\\_req/RandomNumberEngine](https://en.cppreference.com/w/cpp/named_req/RandomNumberEngine)
- Advances internal state as if by x consecutive calls to  $\text{pRNG}_{15}$



# XOSHIRO256\*\*

## How to plug into a random number distribution

Expression	Return type	Requirements
<code>G::result_type</code>	<code>T</code>	<code>T</code> is an unsigned integer type
<code>G::min()</code>	<code>T</code>	Returns the smallest value that <code>G</code> 's <code>operator()</code> may return. The value is strictly less than <code>G::max()</code>
<code>G::max()</code>	<code>T</code>	Returns the largest value that <code>G</code> 's <code>operator()</code> may return. The value is strictly greater than <code>G::min()</code>
<code>g()</code>	<code>T</code>	Returns a value in the closed interval <code>[G::min(), G::max()]</code> . Has amortized constant complexity.



# XOSHIRO256\*\*

## How to plug into a random number distribution

```
class xoshiro256ss
{
public:
    using result_type = std::uint64_t;

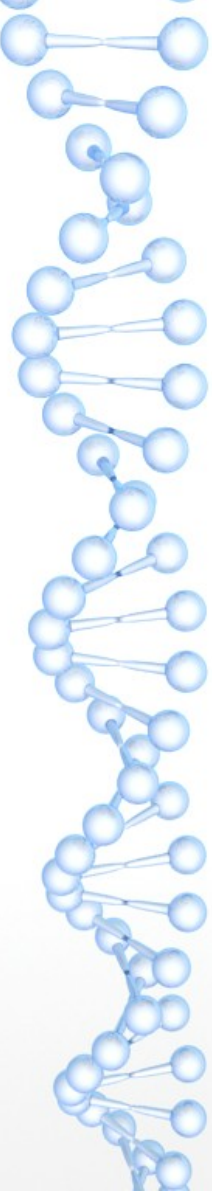
    explicit xoshiro256ss(result_type s = def_seed) noexcept;

    constexpr static result_type min() noexcept { return 0; }

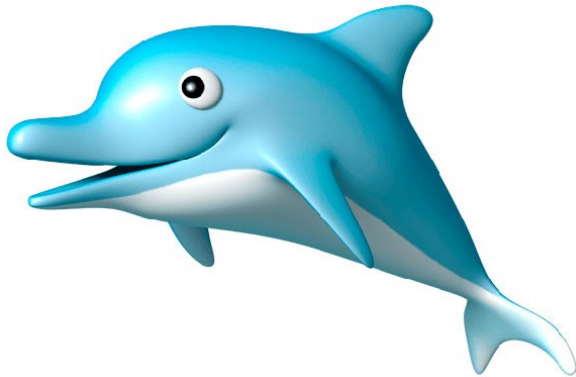
    constexpr static result_type max() noexcept
    { return std::numeric_limits<result_type>::max(); }

    result_type operator()() noexcept { /* next() */ }

private:
    std::array<std::uint64_t, 4> state;
};
```



*Recent studies showed that humans are using only 15% of C++ features... it is also estimated that dolphins are capable of using 20% of C++!*



End of code interlude 1

# EAs Sub-Varieties



Genetic Algorithm  
(GA)

Genetic  
Programming (GP)

Evolutionary  
Strategy (ES)

Evolutionary  
Programming (EP)

Learning Classifier  
System (LCS)

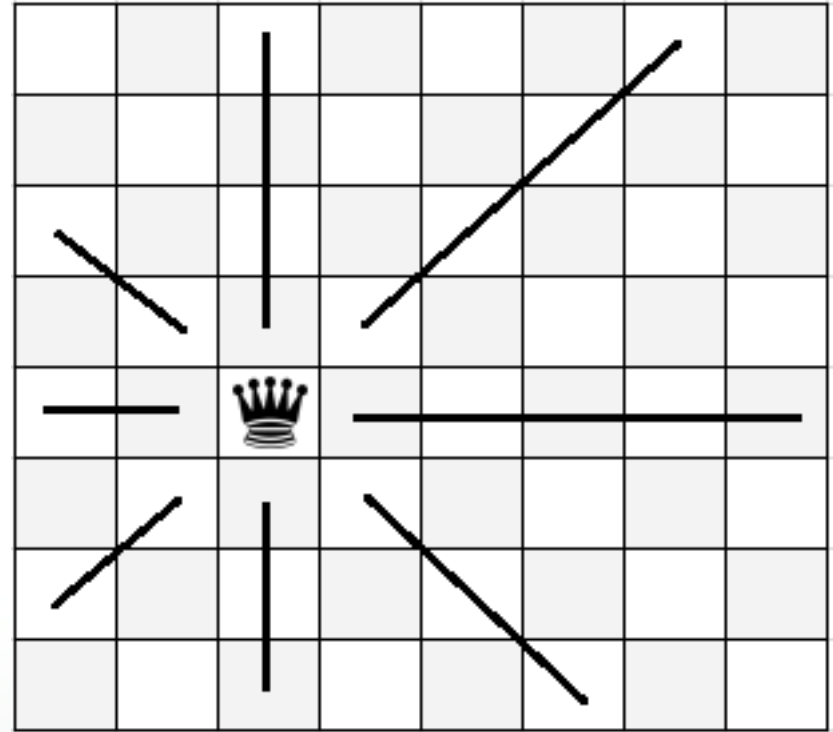


# Sub-Varieties

- **Genetic algorithms.** String of (*traditionally binary*) numbers. Used for optimization problems.
- **Genetic programming.** Computer programs (tree / linear form). Fitness determined by the ability to solve a problem.
- **Differential evolution.** Numerical optimization problem.
- **Neuroevolution.** EA used to generate parameters / topology of artificial neural networks.
- **Evolutionary programming.** Similar to genetic programming, but the structure of the program is fixed and its numerical parameters are allowed to evolve.
- **Evolution strategy.** Vectors of real numbers as representations of solutions. Typically uses self-adaptive mutation rates.

# 8-queens problem – Generalities

- The goal is to place **eight queens** on a chessboard such that **no queen attacks any other**.
- A queen attacks any piece in the same row, column or diagonal
- *Following example is taken from Artificial Intelligence (Norvig - Russell)*





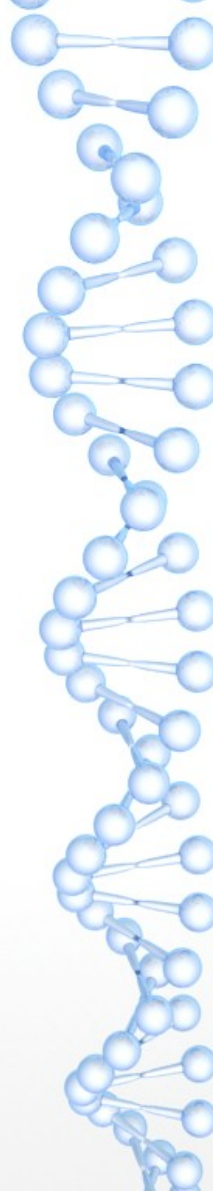
## 8-queens - State/Individual

- Classical GAs: a string of 0s and 1s (24 bit required to specify the position of 8 queens, each in a column of 8 squares).
- In the following example: each individual is represented as 8 digits (in the [1;8] range).
- Performance: permutation encoding of 8 integers.
- Encoding is important: different encodings behave differently.

# 8-queens problem – Fitness function 1

- Heuristic cost function  $h$  is the number of pairs of queens that are attacking each other, either directly or indirectly.
- The global minimum is 0 (perfect solutions).
- Figure shows a state with  $h=17$  and the values of all its successors.

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♔	13	16	13	16
♔	14	17	15	♔	14	16	16
17	♔	16	18	15	♔	15	♔
18	14	♔	15	15	14	♔	16
14	14	13	17	12	14	12	18

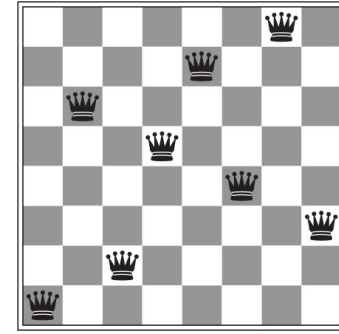
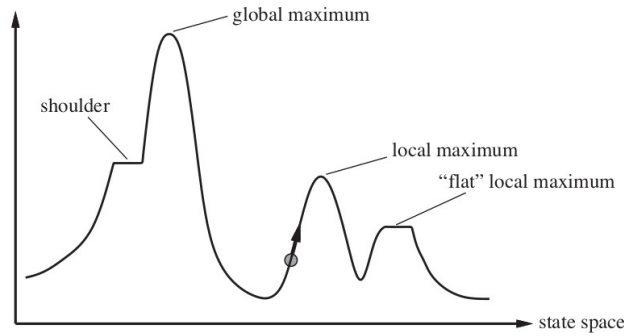


## 8-queens problem – Fitness function 2

- Usually the fitness function should return higher values for better states.
- In general minimizing a function  $f$  is equivalent to maximizing  $-f$ .
- Some frameworks have problem working with negative values.
- For the 8-queens problem we can use the number of *non-conflicting* pairs of queens (28 is a solution):  
$$f = 28 - \text{conflicts} \qquad (== 28 - h)$$

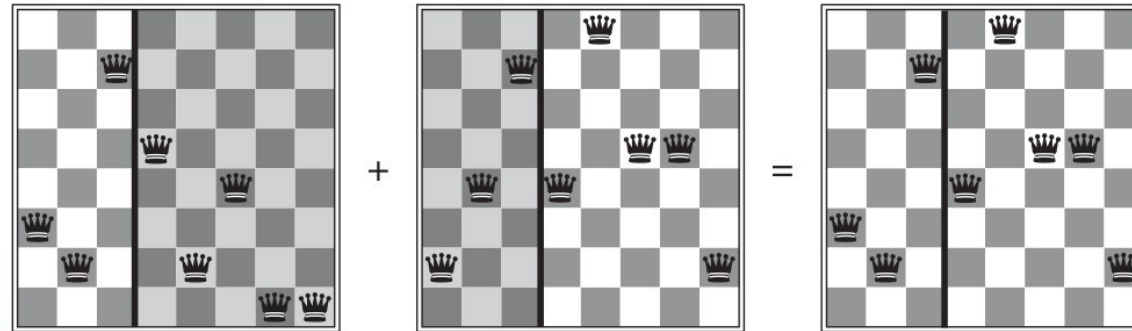
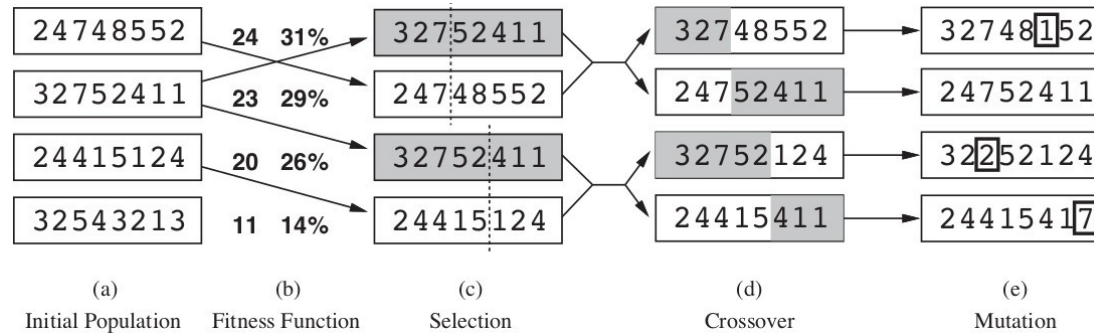


# 8-queens problem - Not straightforward



- A one dimensional state space landscape in which elevation corresponds to the objective function.
- A local minimum in the 8-queens state space ( $h=1$ , every successor has a higher cost).

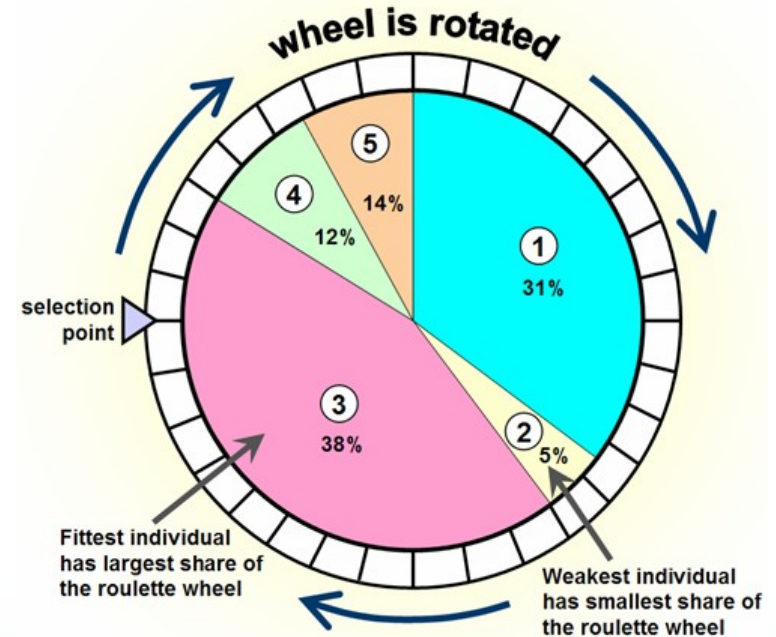
# 8-queens problem - Selection/Crossover/Mutation



# 8-queens problem - Selection

$$p_i = \frac{f_i}{\sum_{j=1}^N f_j}$$

- Here the probability of being chosen for reproducing is directly proportional to the fitness score (percentage shown next to the the raw score)
- One individual is selected twice and one not at all.





## 8 queens problem - Crossover

- Crossover point is chosen randomly from the positions in the string.
- The encoding matters. With a 24 bit encoding (instead of 8 digits) the crossover point has a  $\frac{2}{3}$  chance of being in the middle of a digit, which result in an essentially arbitrary mutation of that digit.
- Initially crossover takes large steps in the state space and smaller steps later (when individuals are more similar).



## 8 queens problem - Mutation

- Moves a queen to a random square in its column.
- In the example one digit was mutated in the 1st, 3rd and 4th offspring.
- Advantage comes from the ability of crossover to combine large blocks of data that have evolved independently to perform useful functions, thus raising the level of granularity at which the search operates (**theory of schema**).



## 2<sup>nd</sup> Code Interlude

Talk is cheap  
Show me the  
**CODE**

*Linus Torvalds*



## 8-queens with Vita – individual (aka chromosome / genome)

```
const int NQUEENS(8);

// Encodes the queen's placement (i.e. a gene in the chromosome).
struct row : vita::ga::integer
{
    row() : vita::ga::integer({0, NQUEENS}) {} // open range
};

int main()
{
    vita::problem prob;
    prob.chromosome<row>(NQUEENS);

    // ...
}
```





## 8-queens with Vita – fitness function

```
auto f = [] (const vita::i_ga &x) -> vita::fitness_t
{
    double attacks(0);

    for (int queen(0); queen < NQUEENS - 1; ++queen)
    {
        const int row(x[queen].as<int>());

        for (int i(queen + 1); i < NQUEENS; ++i)
        {
            const int other_row(x[i].as<int>());

            if (other_row == row                                // same row
                || std::abs(other_row - row) == i - queen)    // or diagonal
                ++attacks;
        }
    }

    return {-attacks};
};
```



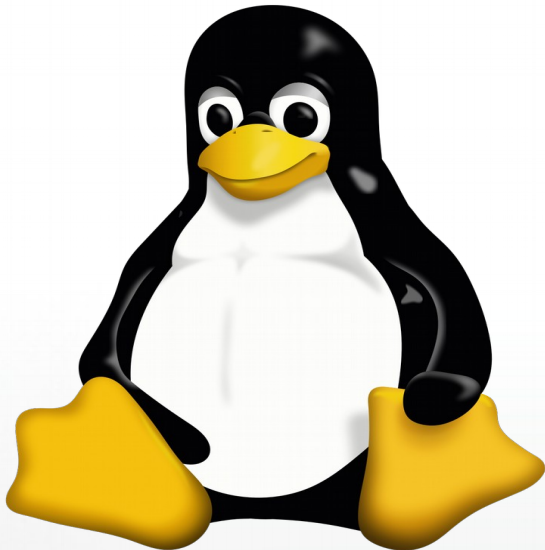


## 8-queens with Vita – let's go

```
vita::ga_search<decltype(f)> search(prob, f);  
auto result = search.run();  
  
// Prints result.  
std::cout << "\nBest result: [";  
for (auto gene : result.best.solution)  
    std::cout << " " << gene.as<int>();  
  
std::cout << " ] (fitness "  
          << result.best.score.fitness << ")\n";
```



*Penguins are allergic to C++...*



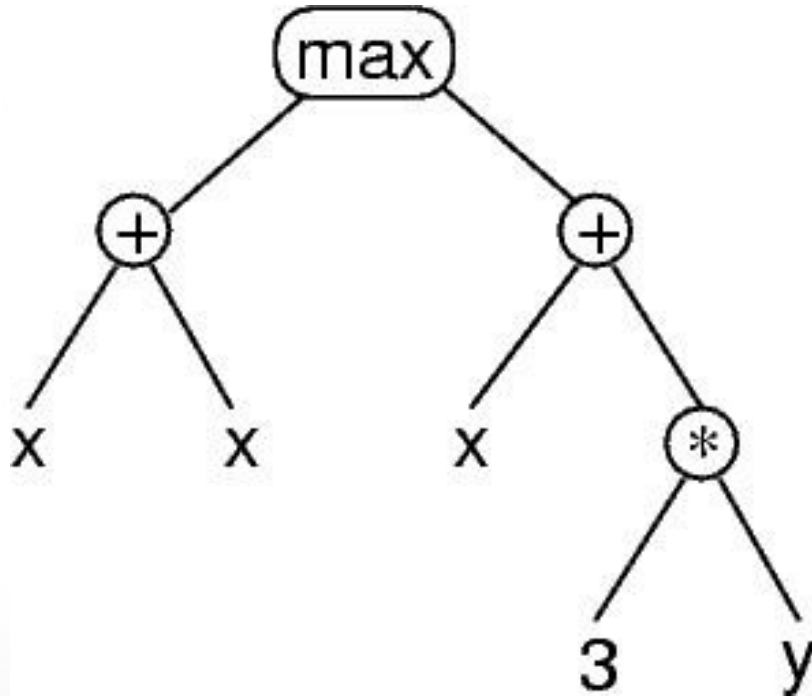
End of code interlude 2



# Genetic Programming

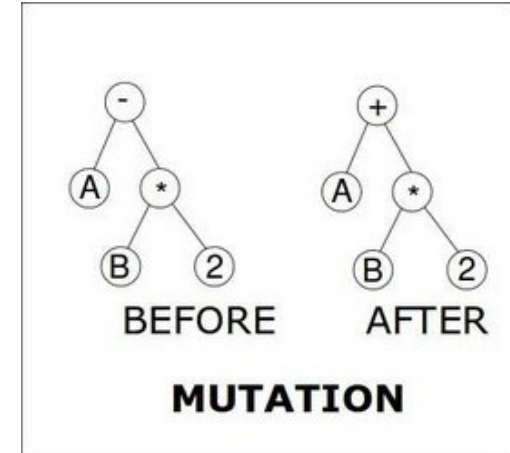
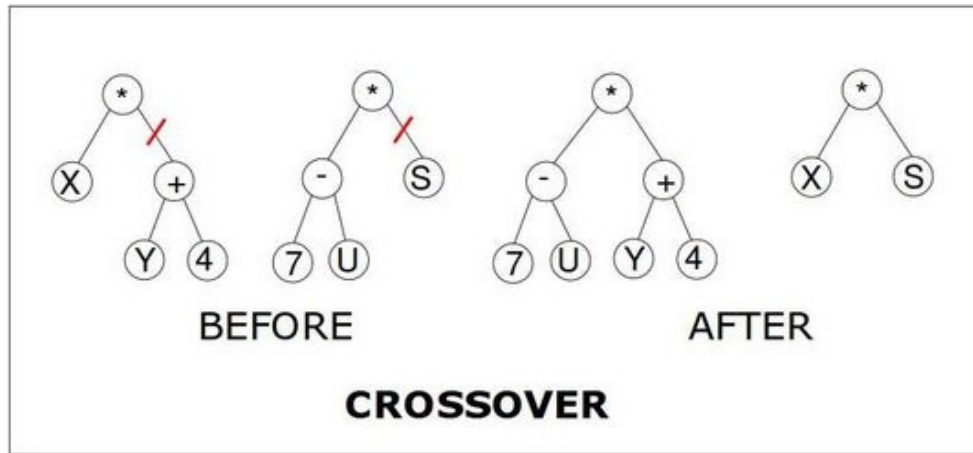
- GP “applies the genetic algorithm to a population of computer programs” (not just *raw data*).
- Structural difference: programs do not have a fixed length.

# Genetic Programming – Representation



- A tree representing  $\max(x+x, x+3*y)$
- It's common in the GP literature to represent programs with trees.
- How one implements GP trees will obviously depend a great deal on the programming languages and libraries being used.
- Symbols are classified in a terminal set / function set.

# Genetic Programming - Recombination





# Genetic Programming - Closure

- For GP to work effectively, most function sets are required to have an important property known as **closure**, which can in turn be broken down into the properties of **type consistency** and **evaluation safety**.
- **Type consistency** is required because sub-tree crossover can mix and join nodes arbitrarily.
- **Evaluation safety** is required because many commonly used functions can fail at run time (e.g. division by 0).
- Symbol set should have **sufficiency**: it should be possible to express a solution to the problem using the elements of the symbol set.

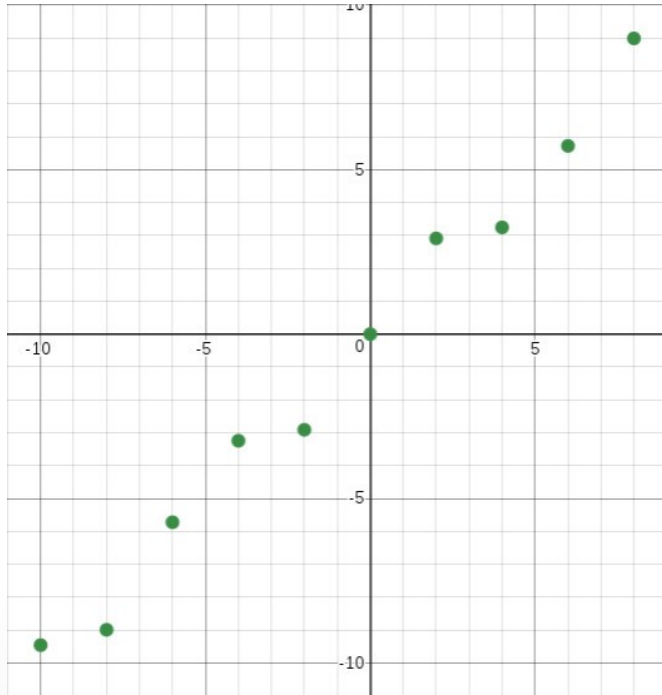


# Symbolic regression with Genetic Programming

X	Y
-10	-9.456
-8	-8.989
-6	-5.721
-4	-3.243
-2	-2.909
0	0.000
2	2.909
4	3.243
6	5.721
8	8.989

**Symbolic regression:**  
you want to discover the  
analytic form of an  
unknown function that  
matches some given  
data.

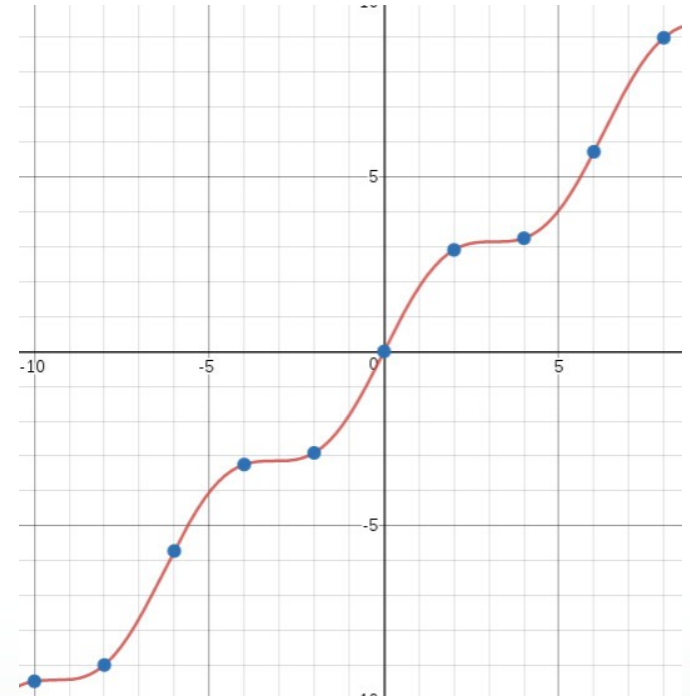
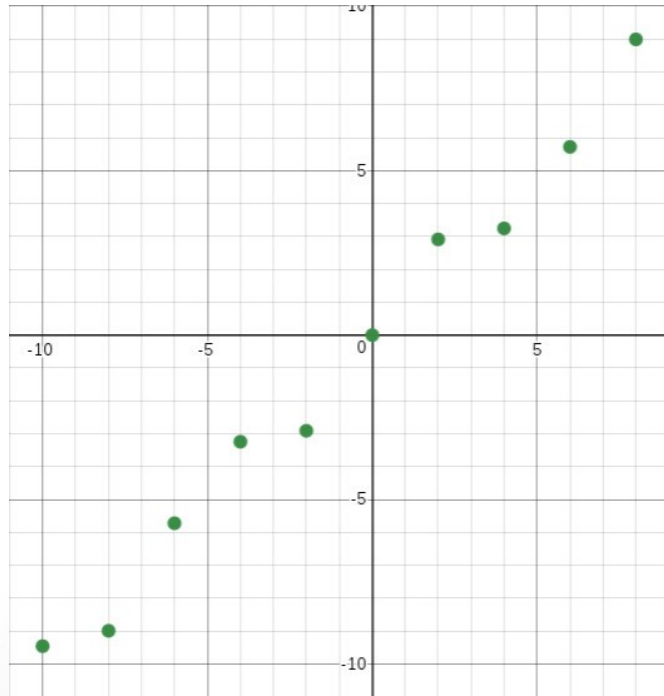
# Symbolic regression with Genetic Programming



- Function set:  $\sin$ ,  $\cos$ ,  $+$ ,  $-$ ,  $/$ ,  $*$ .
- Terminal set:  $x$ .



# Symbolic regression with Genetic Programming



## 3<sup>rd</sup> Code Interlude





# Symbolic regression with Genetic Programming Setup

```
int main()
{
    vita::src_problem prob;

    // DATA
    prob.data().read_csv(training);

    // SYMBOLS
    prob.sset.insert<vita::real::sin>();
    prob.sset.insert<vita::real::cos>();
    prob.sset.insert<vita::real::add>();
    prob.sset.insert<vita::real::sub>();
    prob.sset.insert<vita::real::div>();
    prob.sset.insert<vita::real::mul>();
    prob.setup_terminals();

    // ...
}
```



# Symbolic regression with Genetic Programming

## Let's go

```
vita::src_search<> s(prob);  
const auto result(s.run());  
  
std::cout << "\nCANDIDATE SOLUTION\n"  
            << vita::out::print_format(vita::out::c_language_f)  
            << result.best.solution  
            << "\n\nFITNESS\n" << result.best.score.fitness  
            << '\n';
```

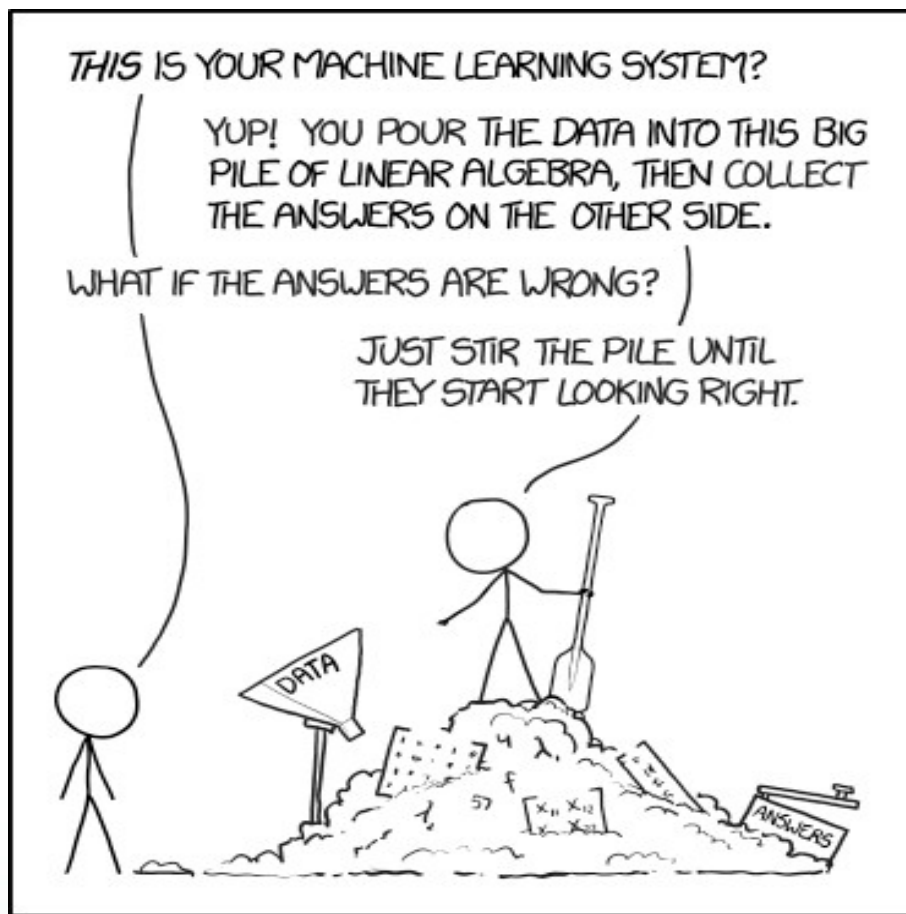
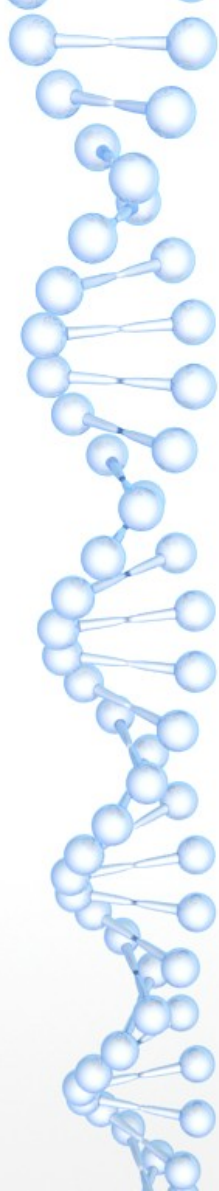


# Symbolic regression with Genetic Programming

## Let's go

Generation	Result	Fitness
100	$X + \sin(X)$	-0.0071387
5000	$\frac{(((X/\cos(X*X)) + \sin(X)) * ((X*X) * ((X*X) * (X*X)))) * ((X*X) * ((X*X) * (X*X))))}{(\cos(X*X) * \cos(X*X)) * (\cos(X*X) * \cos(X*X)) + ((X - (((X/\cos(X*X)) + \sin(X)) * ((X*X) * ((X*X) * (X*X)))) * ((X*X) * ((X*X) * (X*X)))))) + \sin(X)}$	-0.00540735

- Overfitting, bloating
- Regularization term, penalty function



End of code interlude 3



# How to identify when to use Evolutionary Algorithms

The key point in deciding whether or not to use genetic algorithms for a particular problem centers around the question:

what is the space to be searched?



# How to identify when to use Evolutionary Algorithms

If that space is **well-understood** and contains structure that can be exploited by **special-purpose search techniques**, the use of genetic algorithms is generally computationally less efficient.

If the space to be searched is **not so well understood** and **relatively unstructured**, and if an effective GA representation of that space can be developed, then GAs provide a surprisingly powerful search heuristic for large, complex spaces.

*(De Jong, Machine Learning, 1990 nr 5, pg. 351)*



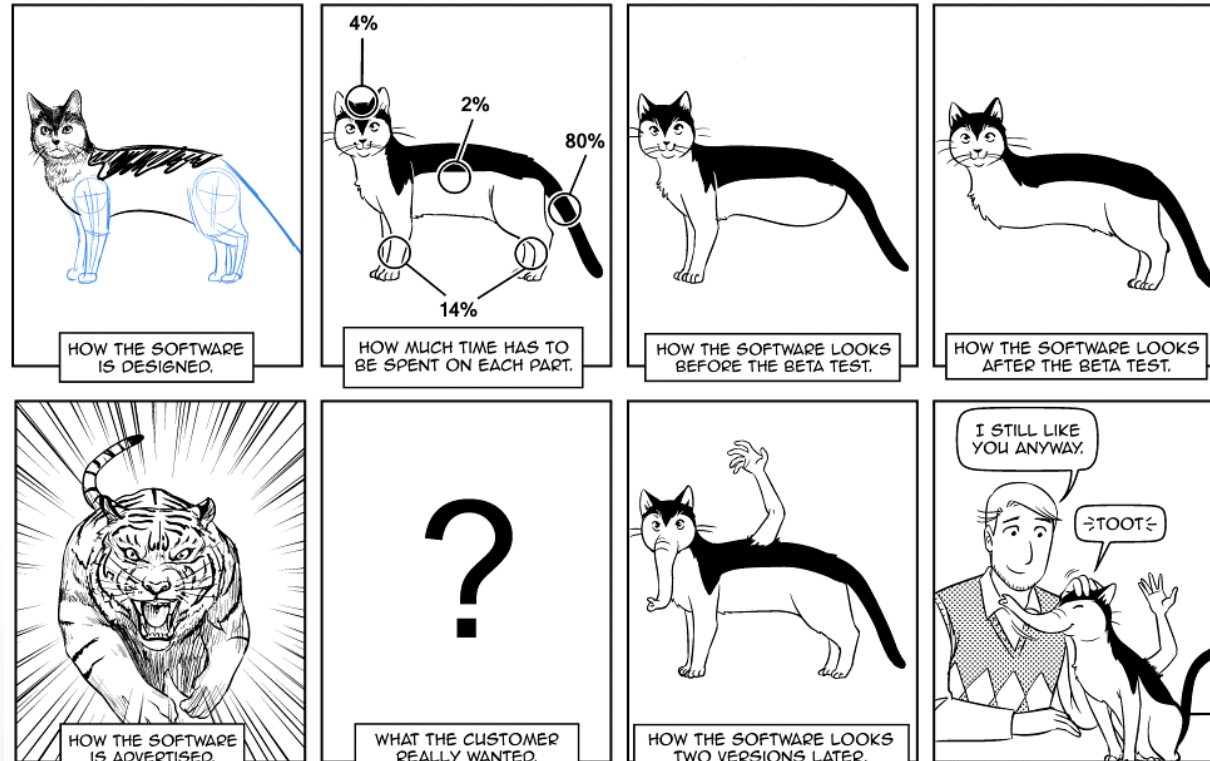


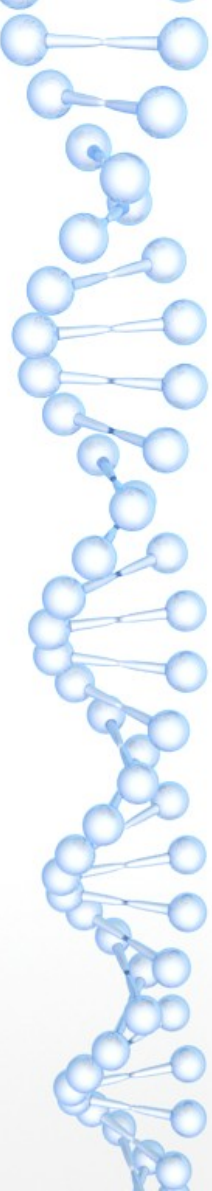
# How to identify when to use Evolutionary Algorithms

In general EAs are more suited to find good solutions quickly than to finding the absolute best solution to a problem.

# EVOLUTIONARY FRAMEWORKS

## Richard's guide to software development





# Evolutionary frameworks

## **Don't Reinvent The Wheel, Unless You Plan on Learning More About Wheels**

<https://blog.codinghorror.com/dont-reinvent-the-wheel-unless-you-plan-on-learning-more-about-wheels/>



# Evolutionary framework

- Deeply educate yourself about all the existing solutions.
- Do not undermine those who legitimately want to build something better or improve on what's already out there.
- Consider the *double nature* of genetic algorithms: if there is a way to crash your code they will find it and, at the same time, they will bypass implementation errors as much as they can (at the expense of performance).

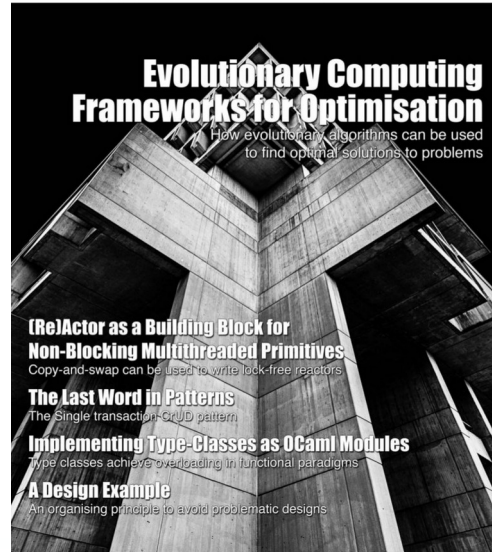


# C++ Evolutionary frameworks for optimization

Framework	Remarks
<b>Evolutionary Computation Framework (ECF)</b>  <a href="http://ecf.zemris.fer.hr/">http://ecf.zemris.fer.hr/</a>	<ul style="list-style-type: none"><li>• Straightforward configuration with few parameters</li><li>• Boost dependencies</li><li>• Support parallelism (MPI)</li><li>• MIT License</li></ul>
<b>PaGMO / PaGMO 2</b>  <a href="https://esa.github.io/pagmo2/">https://esa.github.io/pagmo2/</a>	<ul style="list-style-type: none"><li>• C++14 / 17</li><li>• Massively parallel optimization</li><li>• Multi Objective Optimization</li><li>• EAs sided with state of the art optimization algorithms</li><li>• Boost, Eigen... dependencies</li><li>• Also a Python version</li><li>• GPL3 License</li></ul>
<b>JmetalCpp</b>  <a href="https://github.com/jMetal/jMetalCpp">https://github.com/jMetal/jMetalCpp</a>	<ul style="list-style-type: none"><li>• Available for C++, Java, Python</li><li>• Multi Objective Optimization</li><li>• Last commit 3 years ago</li><li>• MIT License</li></ul>

# Up to date references

overload 142  
DECEMBER 2017 £3



A magazine of ACCU

ISSN: 1354-3172

- <https://accu.org/index.php/journals/2444>

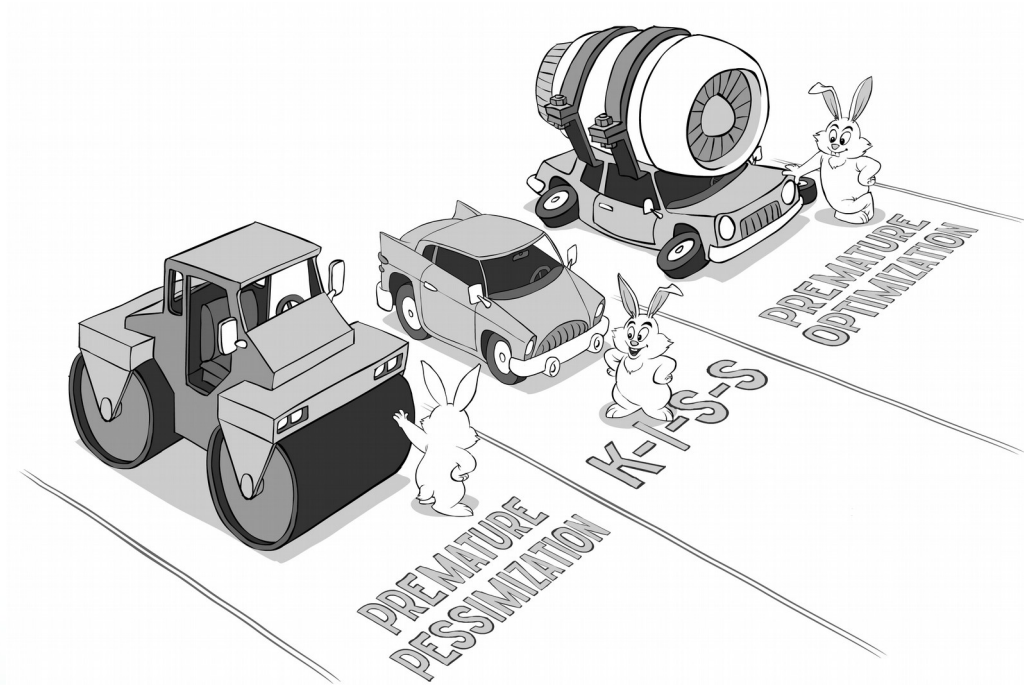


# A C++ framework for Genetic Programming

- Vita  
<https://github.com/morinim/vita>  
C++14  
MPL2 License



## 4<sup>th</sup> Code Interlude

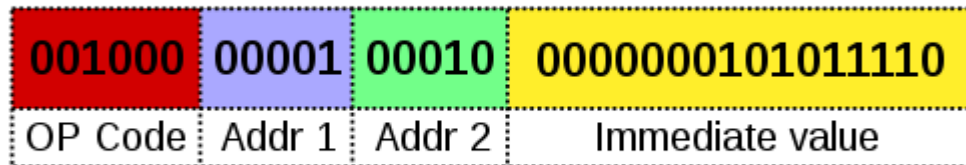


<http://ithare.com/c-performance-common-wisdoms-and-common-wisdoms/>



# Small Vector

## MIPS32 Add Immediate Instruction



Equivalent mnemonic: **addi** \$r1, \$r2, 350

- GP-Programming instructions in Vita include an opcode and zero or more operand specifier (just like a microprocessor instruction).
- With high probability you can say that most of the time the number of elements in your containers do not exceed N. Maybe the number is even less.
- We can “allocate memory” on the stack (which is just shifting the stack pointer). Only if we exceed this preallocated buffer we will fallback to the heap storage.



# Small vector - Skeleton

```
template <class T, std::size_t N>
class SmallVector
{
public:
    SmallVector();
    ~SmallVector();

    void push_back(T value);
    T &operator[](std::size_t index);
    T &at(std::size_t index);

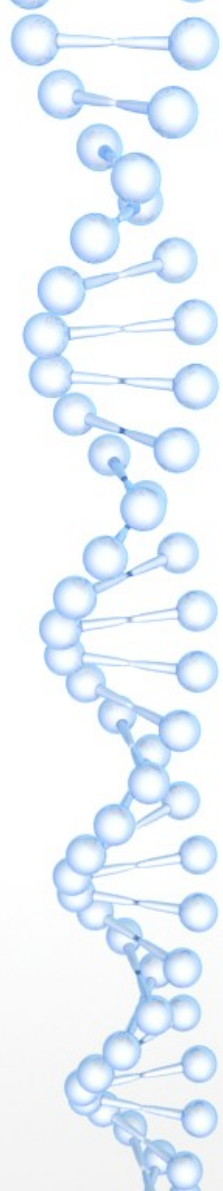
private:
    std::array<T, N> small_buffer_;
    std::size_t size_;
    std::size_t capacity_;
    T *heap_vector_;
};
```



## Small vector – Constructor / access

```
template <class T, std::size_t N>
SmallVector<N>::SmallVector()
    : size_(0), capacity_(N),
      heap_vector_(small_buffer_.data())
{
    for (auto &e : small_buffer_)
        e = T();
}

template <std::size_t N>
T &SmallVector<N>::operator[](std::size_t index)
{
    return heap_vector_[index];
}
```



## Small vector – push\_back

```
template <class T, std::size_t N>
void SmallVector<N>::push_back(T value)
{
    if (size_ == capacity_)
    {
        T *new_storage(new T[capacity_ * 2]);
        capacity_ *= 2;
        std::memcpy(new_storage, heap_vector_,
                    size_ * sizeof(T));

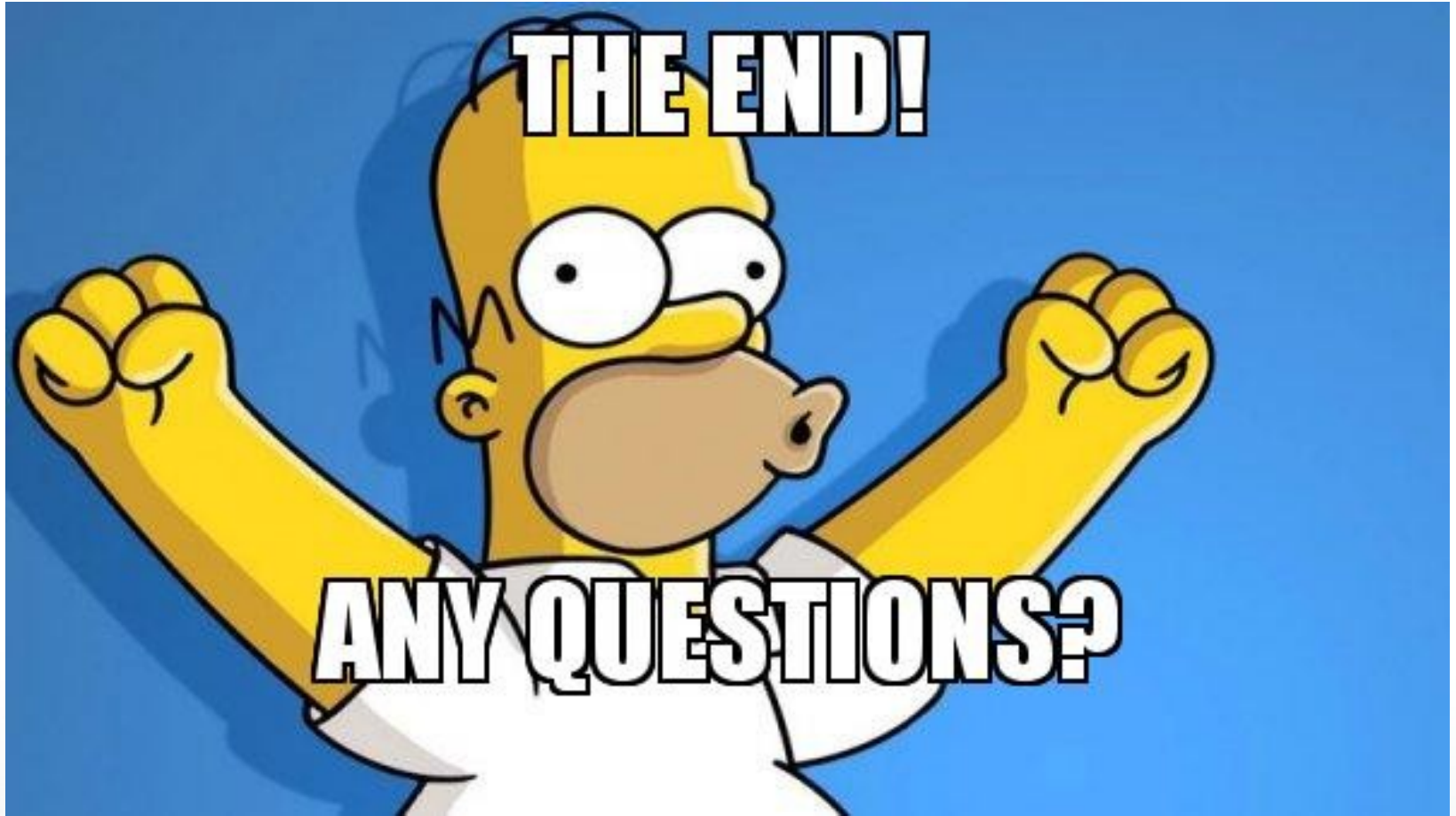
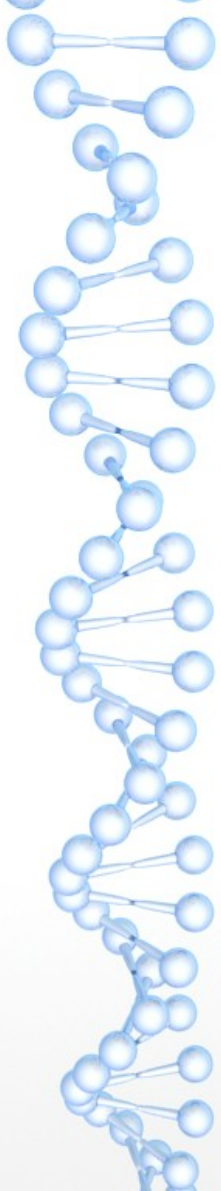
        if (heap_vector_ != small_buffer_.data())
            delete [] heap_vector_;
        heap_vector_ = new_storage;
    }

    heap_vector_[size_++] = value;
}
```



# Small vector - Libraries

- Boost (from v1.58) has the `small_vector` container  
[https://www.boost.org/doc/libs/1\\_68\\_0/doc/html/container/non\\_standard\\_containers.html](https://www.boost.org/doc/libs/1_68_0/doc/html/container/non_standard_containers.html)
- You can borrow the `SmallVector` implementation from LLVM  
<http://llvm.org/docs/ProgrammersManual.html#llvm-adt-smallvector-h>





# References

- **Artificial Intelligence. A Modern Approach** – Peter Norvig, Stuart Russell (3<sup>rd</sup> edition)
- **Essentials of Metaheuristics** – *Sean Luke*  
<https://cs.gmu.edu/~sean/book/metaheuristics/>
- **Genetic Algorithms in Search, Optimization & Machine Learning** – *David E. Goldberg*  
(Addison-Wesley Publishing Company, 1989!)
- **A Field Guide to Genetic Programming**  
<http://www.gp-field-guide.org.uk/>





# References - Articles

- Evolutionary Computing Frameworks for Optimisation  
<https://accu.org/index.php/journals/2444>
- What Makes a Problem Hard for a Genetic Algorithm? Some Anomalous Results and Their Explanation.  
<http://web.cecs.pdx.edu/~mm/Forrest-Mitchell-ML.pdf>
- Fitness distance correlation as a Measure of Problem Difficulty for Genetic Algorithms  
<https://www.cs.unm.edu/~forrest/publications/fdc-icga95.pdf>
- Xoshiro pRNG  
<http://xoshiro.di.unimi.it/>
- GAs string guessing  
[https://github.com/morinim/documents/blob/master/ga\\_string\\_guessing/article.md](https://github.com/morinim/documents/blob/master/ga_string_guessing/article.md)





## References - Code

- A simple `UniformRandomBitGenerator` wrapper for the xoshiro256\*\*\* and xoroshiro128+ PRNGs  
<https://github.com/morinim/xoshiro256ss>
- Symbolic regression with ECF  
<http://ecf.zemris.fer.hr/tutorial.html#tree>