3
From *in vivo* to *in silico*

Cedric Gondro

The Centre for Genetic Analysis and Applications
University of New England

University of
New England

Evolution and population genetics 101

- Evolution
  - organisms survive and reproduce in an environment
- Dynamic
  - Organisms adapt to environmental changes
- Opportunistic
  - No big picture
  - Just the next available position that's good enough

variability + selection pressure -> evolution

better organisms -> more offspring -> more of particular genotype

EC common features

- Population
- Selection
- Search operators

General EC structure

1. Create an initial population – randomly or based on prior information
2. Assign a fitness value to all organisms (also referred to as chromosomes)
3. Select organisms for reproduction based on their fitness and a selection scheme
4. Create descendants from the selected parents
5. Modify the descendants with the search operators
6. Evaluate the fitness of the descendants
7. Cull organisms from the parental population and replace them with the descendants according to the selection scheme
8. Repeat from step 3 until a termination criterion is met, for example, a specified number of iterations or a predefined fitness value is reached

Interlude – representation and implementation

- Binary, integers, reals
- Vectors, matrices
- Lists, stacks
- Parse-trees
- Finite state machines

Example: Population is a multidimensional array

Org1 [Val1][Val2][Val3][Val4][Val5] …

Org2 [Val1][Val2][Val3][Val4][Val5] …

…

Each line in array is an organism

Each position in line is an allele (parameter)

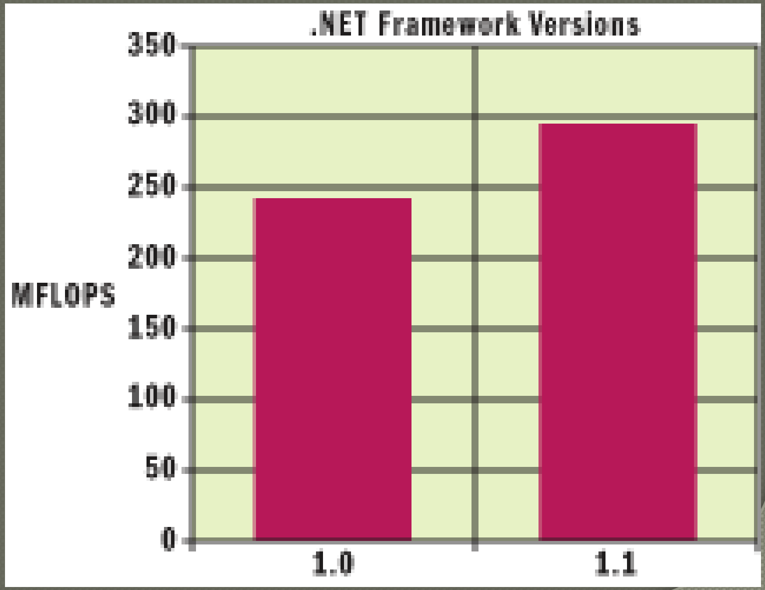
.NET Common Language Runtime

- Connects code to OS and hardware – CPU and OS independence.
- Eliminates need for low level plumbing.
- Use of components developed in other languages – specially inheritance.
- Strongly typed and 100% object-oriented.
- Garbage collection.
- Code is run in native machine language using just-in-time compiling.

Just-in-time Compiling (JIT)

- Better optimization based on individual hardware architecture.
- JIT improvements are reflected in the code without need for the developer to spend time in optimizing the code to target a specific architecture.
- JIT compiling creates a certain overhead when compared to precompiled code.

SciMark measurements



C# Attributes

- Developed specifically to target the .NET framework, compiling to the Microsoft Intermediate Language.
- Mix of C++, Java and Visual Basic.
- Syntax similar to C++.
- Consistent syntax – everything is treated as an object.
- Garbage collection and automatic memory management.
- Error Handling.
- Type safe and object-oriented.
- High-precision floating-point operations (80bit double)
- Allows creation of complex number structures (imaginary numbers – using struct, a primitive data type).
- Multidimensional arrays – jagged arrays.
- Large and helpful community.

C# versus C: SciMark Benchmark

Fast Fourier Transformations (FFT)

Successive Over-Relaxation Iterations (SOR)

Monte Carlo

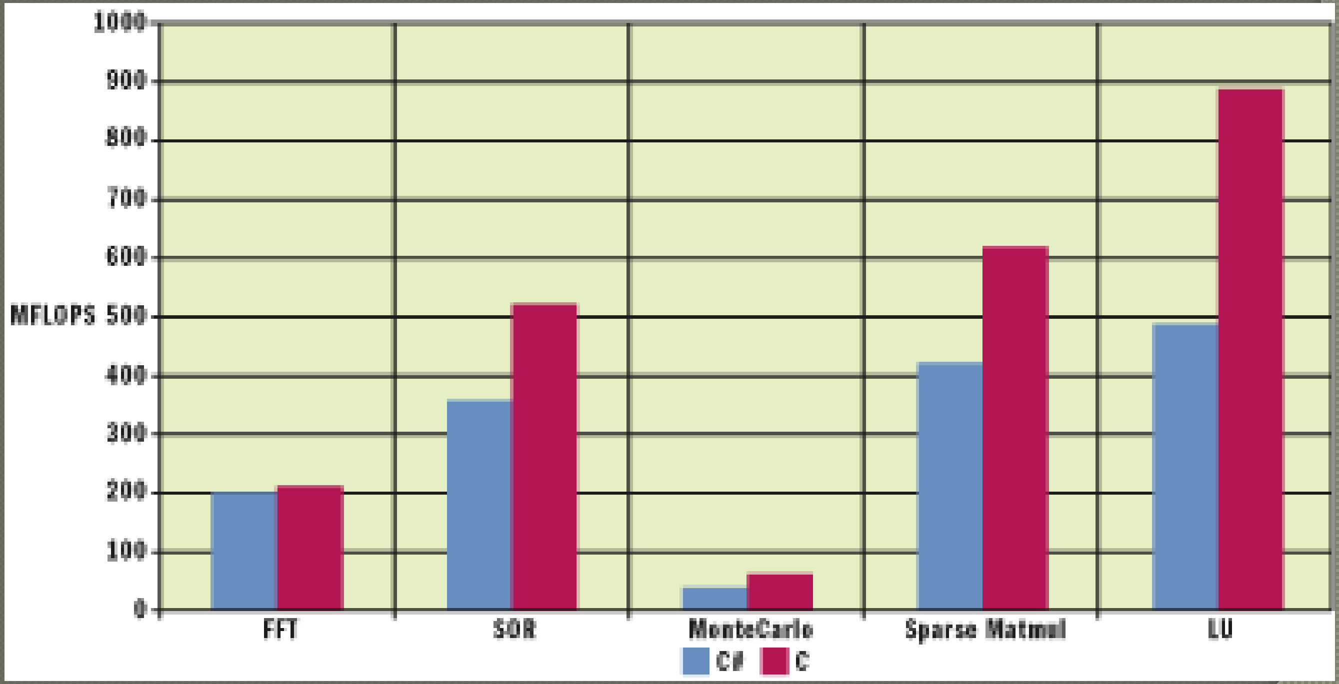
Quadrature

Sparse matrix

multiplications

Dense matrix

factorization (LU)



Language Speed Comparison

Loop 10000 x
u =Sqrt(q) * Exp(-1 * r /Pow(s, 2)) - Pow(x(i), t)

VB in internal class	771.1088
VB in external class	771.1088
C#	761.0944
C++ Managed	761.0944
Fortran Managed	761.0944
C++ Unmanaged	370.5328
Fortran Unmanaged	150.2160

End of interlude – back to EAs

- Features
  - Population
  - Selection
  - Search operators

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

- Illustrate with canonical GA
  - Binary bitstring

1	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**0**

Population

- Initial population
  - Random
  - Seeded

- Population size
  - No fixed rules
  - Interactions between parameters (mutation, crossover, selection)
  - DE smaller populations
  - GA larger populations

Selection

- Keep better, remove worse

- Selection strategies
  - proportionate -> roulette wheel
  - rank-based -> selection probability based on ranking
  - Boltzmann -> changes over time, from uniform to proportionate
  - Tournament -> competition based
  - Truncation -> very high selection pressure

Basically it's population wide or not!

Generation structure

- Continuous or discrete (generational)
  - Slower evolution
  - More robust
  - Less chance of entrapment
  - All solutions die
- Steady-state or overlapping
  - Faster evolution/convergence (specially with tournament)
  - Higher variance
  - Solutions can be immortal

Fitness

- The most important aspect of EAs
- Fitness is not the same as objective function
  - Fitness -> EA
  - Objective function -> problem

Bottom line is that selection can be linked to the problem or independent.

- All or nothing fitness = random search!
- Smooth gives better chances for exploring the solution space

Search operators

- Mutation – new variability
- Crossover – combine variability
- Notes:
  - High mutation – no chance to explore the solution space
  - Low mutation – not enough variability
  - High crossover – break up good solutions
  - Low crossover – no exploration
  - There are more variants than stars in the sky!
- EA = selection + population size + crossover + mutation

Crossover

One-point crossover breakpoint

1	1	0	0	1	1	1	1	0	0	1	Parent A
0	0	1	1	1	1	0	1	0	1	0	Parent B
1	1	1	1	1	1	0	1	0	1	0	Offspring

Mutation

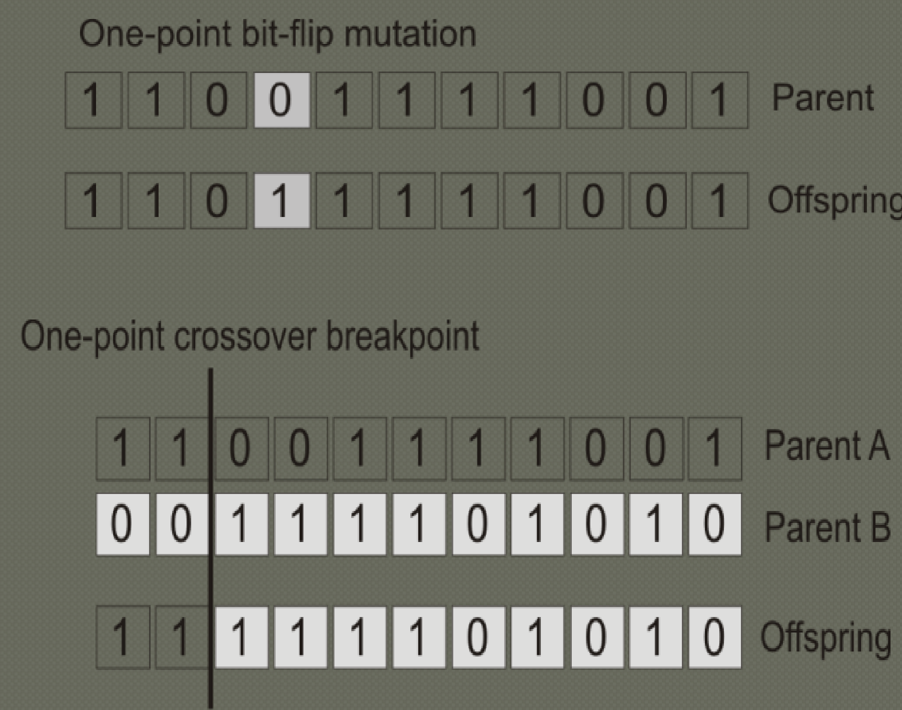
One-point bit-flip mutation

1	1	0	0	1	1	1	1	0	0	1	Parent
1	1	0	1	1	1	1	1	0	0	1	Offspring

GAs are not ideal for numerical optimization

- Why not?

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001



Summary - general framework for EC

- Nature of the problem
- Modeling of the problem
- Objective function
- Development of an evolutionary algorithm

Example

An example – x from 1 – 100.
Objective – find value of constants

Functions
y1=x*v1
y2=x/v2
y3=x-v3
y4=v4/x

Constants (unknown!)
5.43
7.66
45
28.2

dataset

1	5.43	0.130548303	-44	28.2
2	10.86	0.261096606	-43	14.1
3	16.29	0.391644909	-42	9.4
4	21.72	0.522193211	-41	7.05
5	27.15	0.652741514	-40	5.64
6	32.58	0.783289817	-39	4.7
7	38.01	0.91383812	-38	4.028571429
8	43.44	1.044386423	-37	3.525
9	48.87	1.174934726	-36	3.133333333
10	54.3	1.305483029	-35	2.82
11	59.73	1.436031332	-34	2.563636364
12	65.16	1.566579634	-33	2.35
13	70.59	1.697127937	-32	2.169230769
14	76.02	1.82767624	-31	2.014285714
…				

Which method is best?
None! NFL still applies, but…

Test Case: 30 values for y1,y2,y3,y4 (20000 evaluations)

$y1=(1+k1*\underline{y4}*P)/(1+(\underline{y4}*P)-(B*\underline{y1}))$

$y2=\underline{y1}-(b2*\underline{y2})$

$y3=(R*\underline{y1})-(b3*\underline{y3})$

$y4=(S*\underline{y2})-(\underline{y3}*\underline{y4})$

Method	SSE
Random Search	-0.01248
Greedy Hill Climbing	-0.00098
Simple Genetic Algorithm	-3.91E-16
Differential Evolution	12.91E-27

University of New England