Lecture 2: Simple Genetic Algorithms

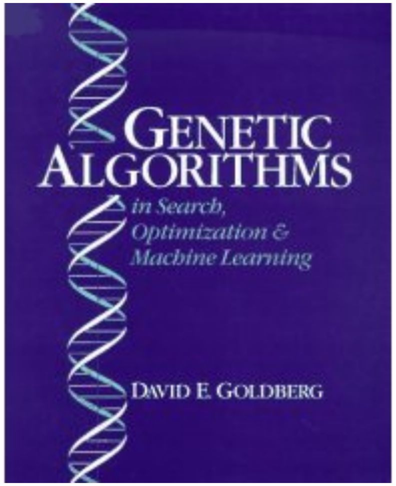
Genetic Algorithms and Other Evolutionary Techniques

POLITECNICO DI MILANO

Prof. Pier Luca Lanzi

References

- David E. Goldberg "Genetic Algorithms in Search, Optimization and Machine Learning" Addison Wesley, 1989.
- CS 5320 "Introduction to Evolutionary Computation" Prof. Martin Pelikan, University of Missouri, St. Louis, MO



Outline

- Basic Procedure
- Terminology
- Operators
- Initialization
- Evaluation
- Selection
- Variation
- Replacement
- Simulation

Basic Idea

Evolve a population (multiset) of candidate solutions using the concepts of survival of the fitness, variation, and inheritance

Genetic Algorithm

Generate an initial population

Repeat

- Select promising solutions from the population
- Create new solutions by applying variation
- Incorporate new solutions into original population

Until stop criterion met

A Simple Genetic Algorithm for Binary Strings


```
Genetic algorithm(n,N,f,pm,pc)
  P = generate(n,N);
  while (!done())
    fv = evaluate(P,f,n,N);
    S = selection(P,fv,n,N);
    O = variation(S,n,N,pm,pc);
    P = replacement(O,P,n,N);
```

Parameters

n	The number of bits
N	The population size
f	The objective function
pm	Probability of mutation
pc	Probability of crossover

Common Terminology in Genetic Algorithms

- Solution String
- String position
- Bit, feature value
- Objective Function
- Selected solutions
- New candidate solutions
- Iteration
- Structure
- Decoded structure
- Nonlinearity

- Individual, chromosome
- Locus
- Allele
- Fitness function, fitness
- Parents
- Offspring
- Generation
- Genotype
- Phenotype
- Epistasis

Operators

Initialization Randomly generates the initial population of strings.

Evaluation Evaluates the population of strings using the given fitness function.

Selection Selects promising solutions from the current population by making more copies of better solutions at the expense of the worse ones.

Variation Processes selected solutions to generate new candidate solutions that share similarities with selected solutions but are novel in some way.

Replacement Incorporates new candidate solutions into the original population.

Generating the Initial Population


```
generate(n,N)
  P = new population of size N;
  for i=1 to N
    P[i] = generate_random(n);
  return P;
```

Evaluation

```
evaluate(P,f,n,N)
  fv = new array of N real numbers;
  for i=1 to N
    fv[i]=f(P[i],n);
  return fv;
```

Selection

- Fitness proportionate selection
- Tournament selection
- Truncation selection

Tournament selection

Parameters

Tournament size k

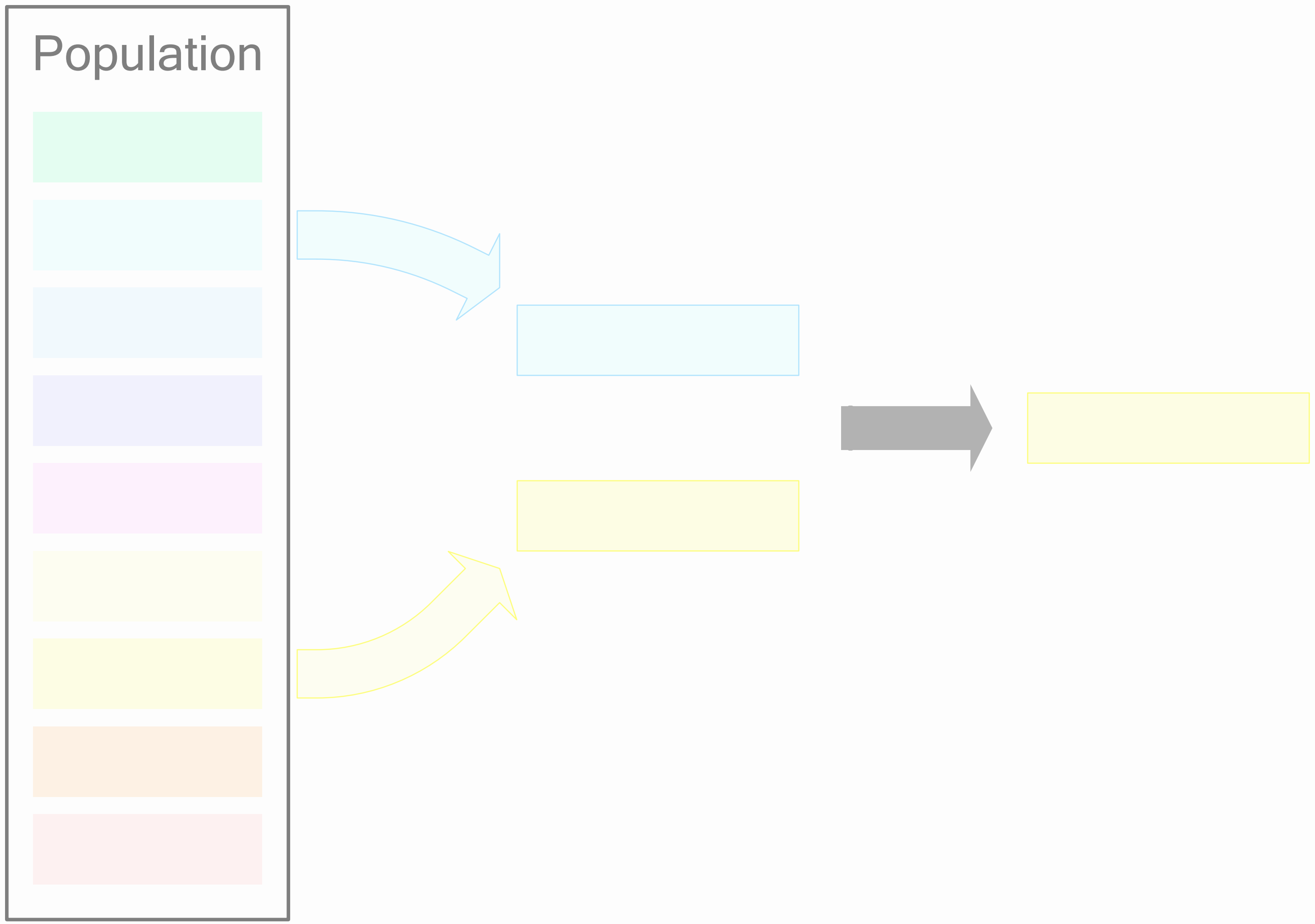
Procedure

To select each candidate solution, select a random subset of k solutions from the original population and then select the best solution out of this subset.

Comments

- Tournament selection is among the most popular selection methods in genetic algorithms.
- Binary tournament selection ($k = 2$) is probably most popular.

Binary Tournament Selection



Binary Tournament Selection


```
binary_tournament_selection(P,fv,n,N)
  S = new population of size N;
  for i=1 to n
    a=rand(1,n);
    b=rand(1,n-1);
    if (b>=a) b++;
    if (fv[a]>fv[b])
      S[i]=P[a];
    else
      S[i]=P[b];
  return S;
```

Truncation Selection

Parameters

Truncation threshold τ 2 (0,1)

Procedure

Select top $\tau \times N$ candidate solutions from the population (based on the value of the objective function).

Comments

- Very intuitive selection method (select best guys).
- Usually, $\tau = 0.5$ (top 50%) or $\tau = 0.3$ (top 30%).

Fitness Proportionate Selection

Parameters

None

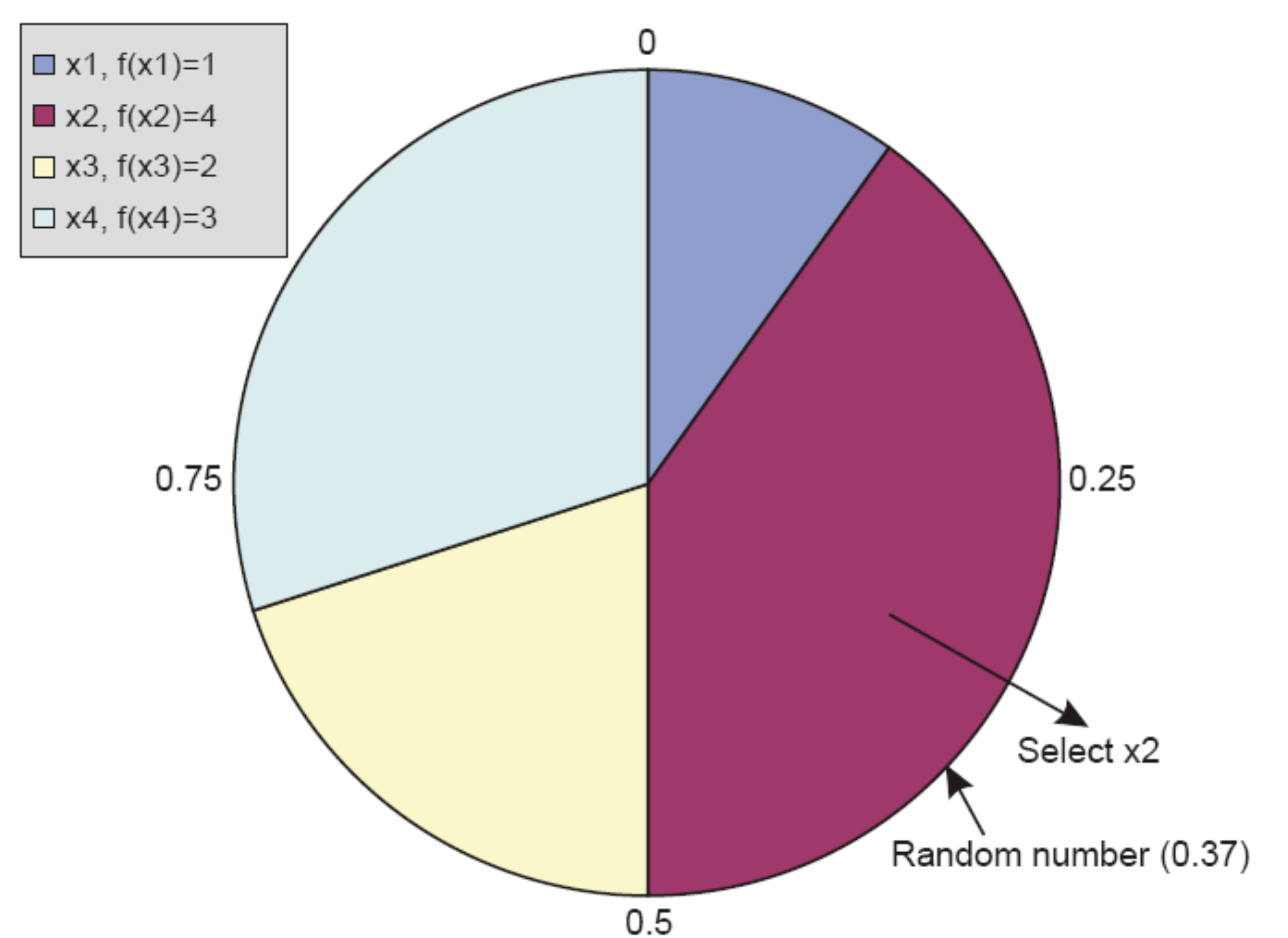
Procedure

Select each candidate solution x with probability proportional to its fitness f (x) (proportionate selection assumes that fitness is positive for all solutions).

Comments

- Also called proportionate selection.
- Selection is usually implemented using a roulette wheel generator.
- If fitness values are scaled exponentially with a temperature parameter, we have Boltzmann selection.

Fitness Proportionate Selection



Fitness Proportionate Selection


```
proportionate_selection(P,fv,n,N)
  S=new population of size N;
  pc=new array of N real values;
  pc[1]=fv[1];
  for i=2 to N pc[i]=pc[i-1]+fv[i];
  for i=1 to N-1 pc[i]=pc[i]/pc[N];
  pc[N]=1;
  for i=1 to N
    r=rand01();
    j=1;
    while (r>=pc[j])
      j=j+1;
    S[i]=P[j]; \\ select individual P[j]
  remove pc;
  return S;
```

Discussion About Selection

Observations

- Selection methods behave differently in the way they put pressure on quality of candidate solutions.
- Some selection methods are stochastic, some are deterministic.
- Some selection methods ensure that the overall best is selected, some don't.
- Strength of some selection methods (selection pressure) can be tuned.

Comments

- How do these methods behave in terms of selection pressure?
- We will look at this issue later and study selection methods theoretically

Variation

Purpose
- Process selected promising solutions.
- Create new solutions that share features with selected solutions but are new in some way.
- Two basic principles:
  - variation (introducing novelty), and
  - inheritance (reusing the old).

Variation in genetic algorithms

Two components
- Crossover: Combines bits and pieces of promising solutions.
- Mutation: Makes small perturbations to promising solutions.

Variation: Basic Procedure


```
variation(S,n,N,pm,pc)
  O = new population of size N;
  shuffle(S,n,N);
  for i=1 to N with step 2
    O[i] = mutation(S[i],n,pm);
    O[i+1] = mutation(S[i+1],n,pm);
    if (rand01()<pc)
      crossover(O[i],O[i+1],n);
  return O;
```

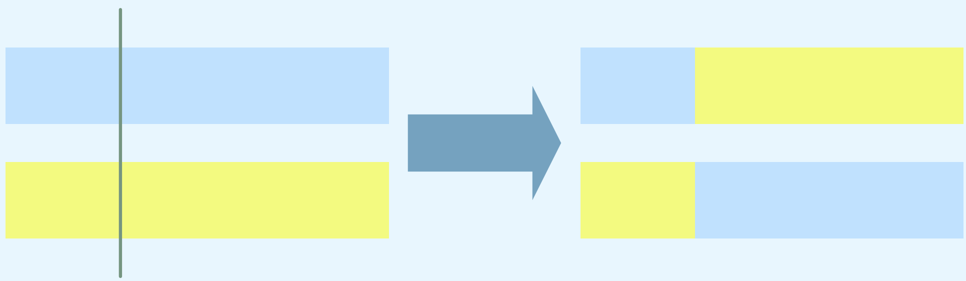
Comments

- The call shuffle(S,n,N) randomly reorders strings in the population P

One-point Crossover

Basic idea

- Randomly select one string position called crossing point.
- Exchange all bits after this position.



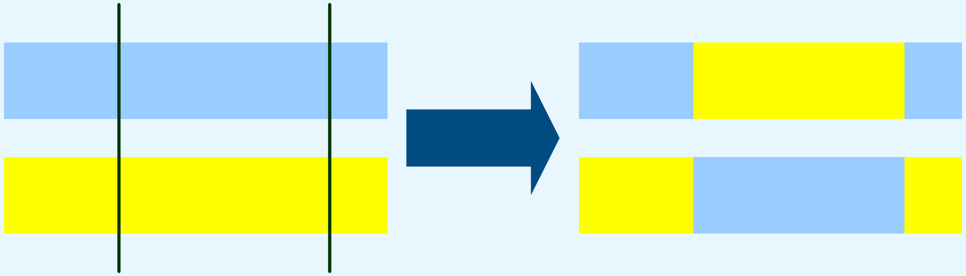
Pseudocode

```
onepoint_crossover(x,y,n)
  cp=random(2,n);
  for i=cp to n
    exchange(x[i],y[i]);
```

Two-point Crossover

Basic idea

- Exchange all bits between two randomly chosen points



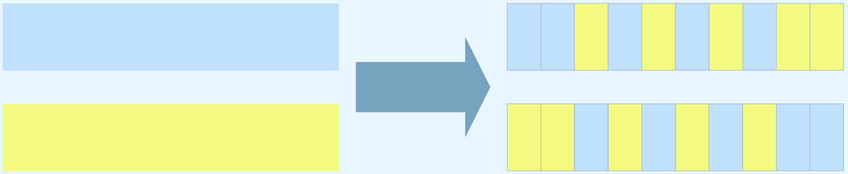
Pseudocode

```
twopoint_crossover(x,y,n)
  cp1=random(1,n);
  cp2=random(1,n);
  if (cp1>cp2)
    exchange(cp1,cp2);
  for i=cp1 to cp2
    exchange(x[i],y[i]);
```

Uniform Crossover

Basic idea

- Exchange every bit with probability 0.5



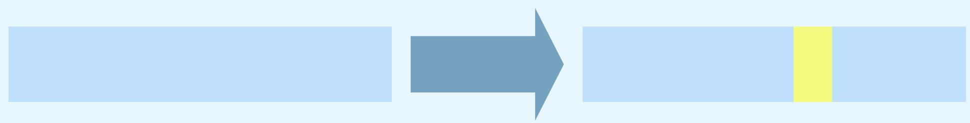
Pseudocode

```
uniform_crossover(x,y,n)
  for i=1 to n
    if (rand01()<0.5)
      exchange(x[i],y[i]);
```

Bit-Flip Mutation

Basic idea

- Flip every bit with a specified probability
- Usually one or only few bits should be mutated
- Typically pm=1/n



Pseudocode

```
mutation(x,n)
  y=new binary string of n bits;
  for i=1 to n
    if (rand01()<pm)
      y[i]=1-x[i];
    else
      y[i]=x[i];
```

Replace All

Basic idea

- Assume that the offspring population is of the same size as the original population.
- Replace the entire original population with offspring.

Pseudocode

```
replace_all(O,P,n,N)
  new_P = new population of N strings;
  for i=1 to N
    new_P[i]=O[i];
  return new_P;
```

Replace Worst (Elitism)

Basic idea

- Assume that the offspring population is smaller than the original population.
- Replace worst strings in the original population by new offspring.

Elitism

Elitist genetic algorithms preserve best solutions found so far.
This is one of elitist schemes.

Discussion About Operators

- Are there other operators?
  - There are many other operators that we will look at later; for now we need only what we have looked at so far.

- Things to think about
  - What are the differences between different selection methods?
  - What are the differences between different variation operators?
  - How would GA operators perform by themselves?
  - Why would the particular combination of operators work?
  - How will GA work on onemax?
  - How will GA work on other problems?

Simulation by Hand

- Parameters
  - number of bits — n = 5
  - population size — N = 4
  - fitness function — onemax (count ones)
  - selection binary — tournament selection
  - variation — one-point crossover and bit-flip mutation
  - replacement — replace all
  - prob. of crossover — pc = 1.0
  - prob. of flip — pm = 1/5

Initialization and Evaluation

String No.	Initial population (generated randomly)	$f(x)$ (onemax)
1	11000	2
2	10111	4
3	01011	3
4	00100	1

Selection (Binary Tournament)

String No.	Initial population	$f(x)$ (onemax)	Tournaments	Selected population
1	11000	2	00100 vs. 11000	11000
2	10111	4	01011 vs. 10111	10111
3	01011	3	10111 vs. 00100	10111
4	00100	1	11000 vs. 01011	01011

Crossover + Mutation

String No.	Selected population	Shuffled parents	After mutation	Crossing point	After crossover
1	11000	10111	10011	4	10010
2	10111	11000	11010		11011
3	10111	01011	11001	2	10111
4	01011	10111	10111		11001

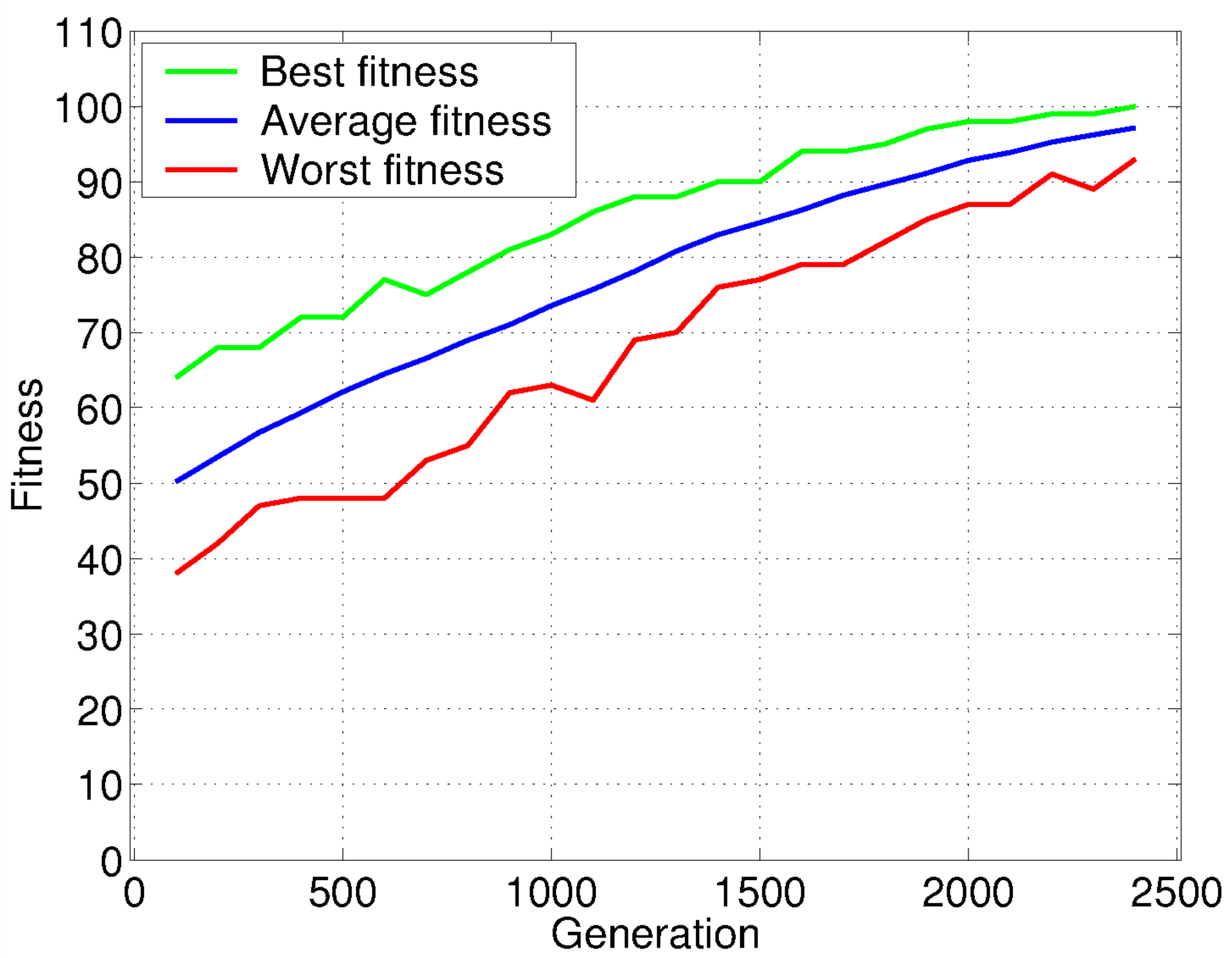
Replacement and Evaluation

String No.	Old population	$f(x)$	Offspring population	New population	$f(x)$
1	11000	2	10010	10010	2
2	10111	4	11011	11011	4
3	01011	3	10111	10111	4
4	00100	1	11001	11001	3
min		1			2
avg		2.5			3.25
max		4			4

Discussion

- The good
  - Average fitness increased
  - Minimum fitness decreased
  - Maximum fitness did not decrease
  - We obtained new high quality solutions
- Reality
  - Not always everything goes well, but if things work, this should happen most of the time

Actual Run for 100-bit Onemax



Summary

- Description of the simple genetic algorithm
- Population of binary strings
- Three basic selection methods
  - Tournament
  - Truncation
  - Roulette-wheel
- Variation
  - One-point/Two-point/Uniform crossover
  - Bit-flip mutation
- Very simple, but how does it work?