POLITECNICO DI MILANO

Genetic Algorithms

Data Mining and Text Mining

Prof. Pierluca Lanzi

Syllabus

- A.E. Eiben & J.E. Smith Introduction to Evolutionary Computing (Natural Computing Series) 2nd ed. 2015
- Sean Luke Essentials of Metaheuristics https://cs.gmu.edu/~sean/book/metaheuristics/
- David E. Goldberg "The Design of Innovation" Springer 2002
- Martin Pelikan "Hierarchical Bayesian optimization algorithm: Toward a new generation of evolutionary algorithms", Springer 2005
- Zbigniew Michalewicz "Genetic Algorithms + Data Structures = Evolution Programs" Springer 1998

What's the problem we are trying to solve?

Given a certain problem and a target function over the domain find the maximum/minimum

If we knew the analytic form, ...

Algorithm 1 *Gradient Ascent*

1: $\vec{x} \leftarrow$ random initial vector
2: **repeat**
3: $\quad \vec{x} \leftarrow \vec{x} + \alpha \nabla f(\vec{x})$
4: **until** $\vec{x}$ is the ideal solution or we have run out of time
5: **return** $\vec{x}$

If we knew the analytic form, …

Algorithm 1 *Gradient Ascent*

```
x⃗ ← random initial vector
repeat
    x⃗ ← x⃗ + α∇f(x⃗)
until x⃗ is the ideal solution or we have run out of time
return x⃗
```

If we do not know the analytic form, …

$$x \Rightarrow \boxed{problem} \Rightarrow f(x)$$

Basic Black Box Optimization

- Random search
- Hill Climbing
- Tabu Search
- Simulated Annealing
- …

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

```
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	Individual, chromosome
String position	Locus
Bit, feature value	Allele
Objective Function	Fitness function, fitness
Selected solutions	Parents
New candidate solutions	Offspring
Iteration	Generation
Structure	Genotype
Decoded structure	Phenotype
Nonlinearity	Epistasis

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;
```

Tournament selection

Parameters

Tournament size k

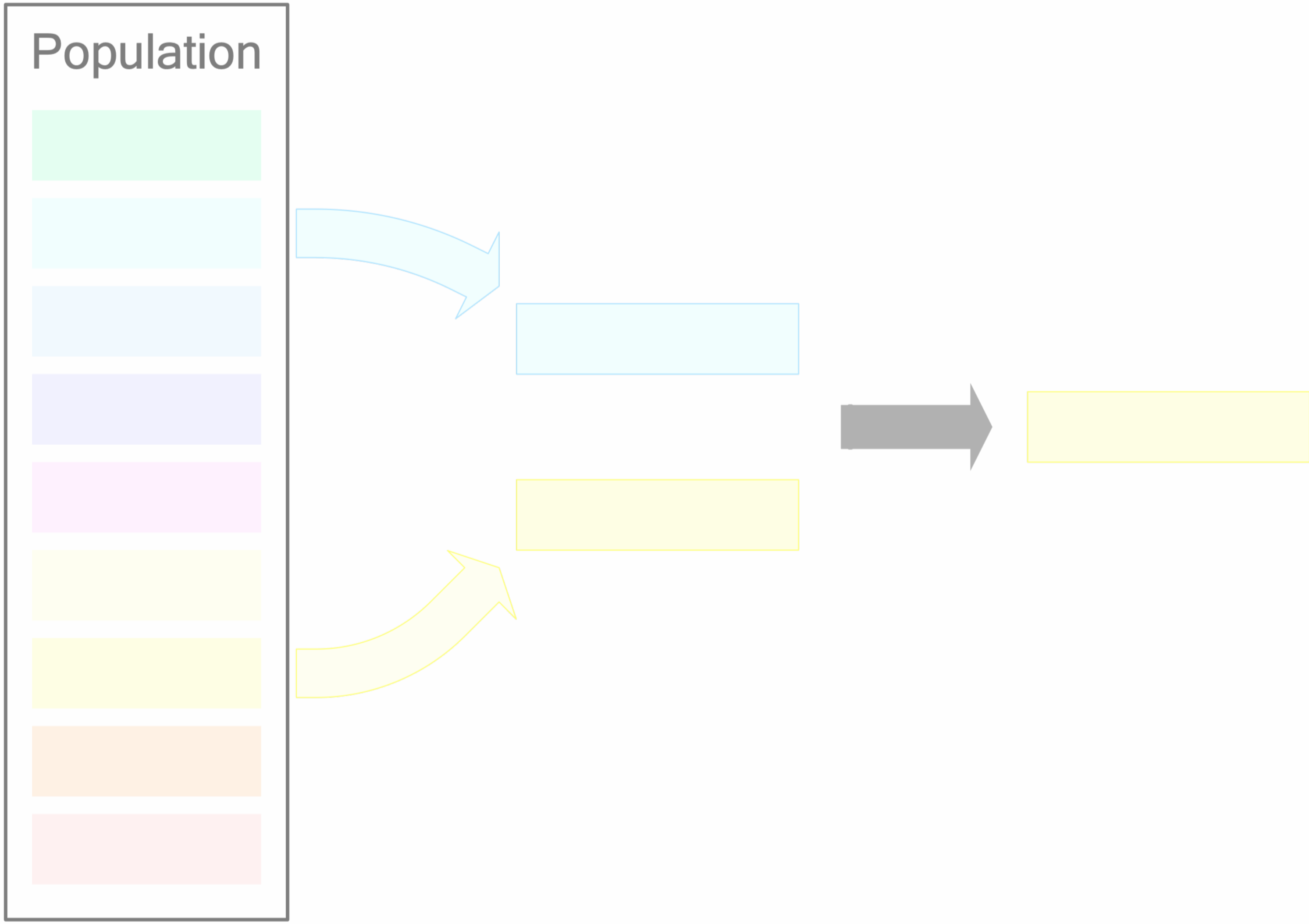
Procedure

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;
```

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;
```

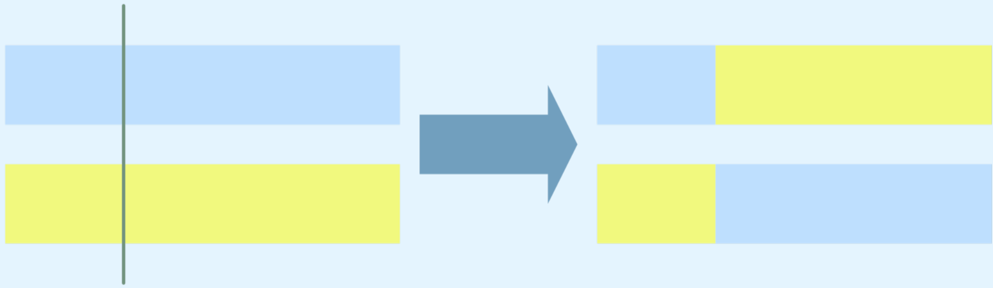
Comments

- 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.



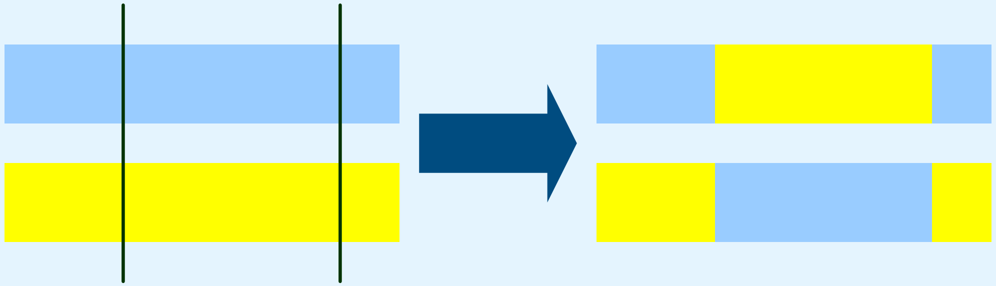
Pseudocode

```
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



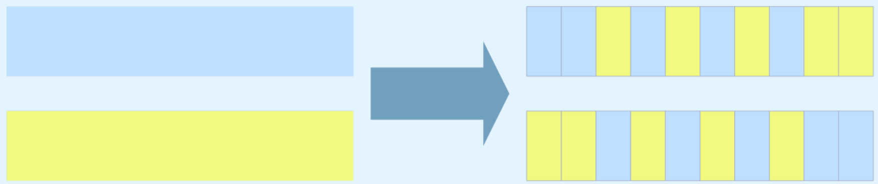
Pseudocode

```
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]);
```

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.

Real-Coded Genetic Algorithms

Real-Valued Operators

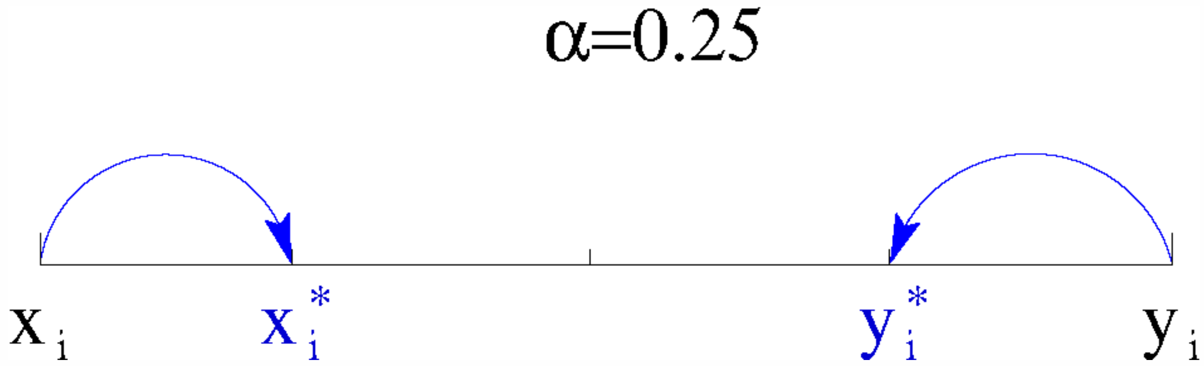
- Candidate solutions are vectors of real values.
- Random initialization generates a random number from an interval for each variable.
- Operators must deal with real-valued parameters.
- There are many ways to do this.
- We only look at some simple methods.

Arithmetic Crossover

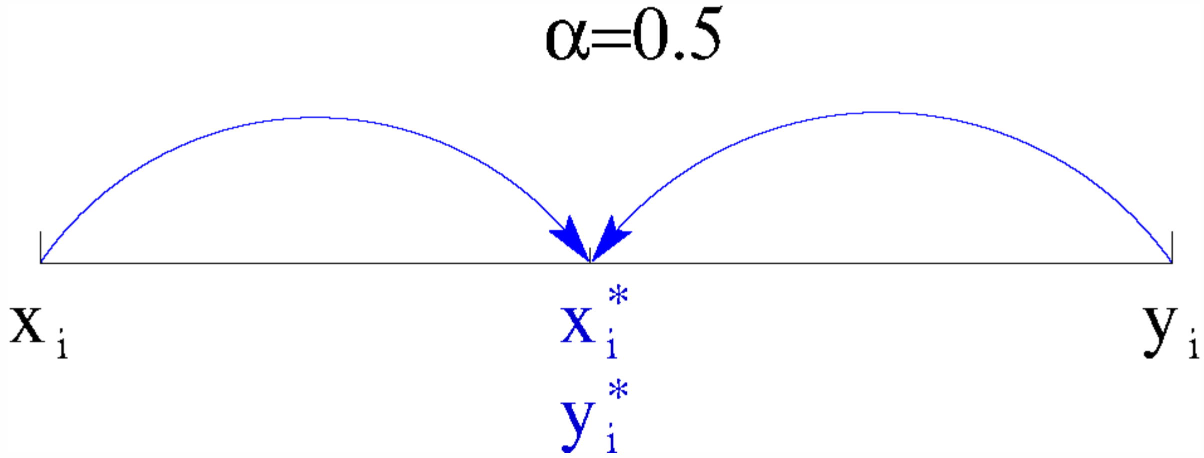
- Two real-valued parents
  - $x = (x_1, \dots, x_n)$
  - $y = (y_1, \dots, y_n)$
- Choose random variable $i \in \{1, 2, \dots, n\}$.
- Randomly generate $\alpha \in [0, 1]$.
- The first child is
  - $o_1 = (x_1, \dots, x_{i-1}, \alpha y_i + (1-\alpha)x_i, x_{i+1}, \dots, x_n)$
- The second child is
  - $o_2 = (y_1, \dots, y_{i-1}, \alpha x_i + (1-\alpha)y_i, y_{i+1}, \dots, y_n)$

Arithmetic Crossover: Example

- Example 1



- Example 2



- Simple mutation
  - Change each variable with a fixed probability.
  - To change a variable, start by generating a random number from $[-\delta, \delta]$ where $\delta>0$ is a small number.
  - Add the generated number to the variable.
- Gaussian mutation
  - The change is generated according to the Gaussian distribution $N(0, \sigma^2)$ where σ^2 is the variance of the mutation steps, which is a small number.
- Another idea
  - Choose a random variable for each solution and mutate that one.