AIASYB-2
Aplicación de la Inteligencia Artificial a los Sensores y Biosensores
PCI-AECID B/024393/09

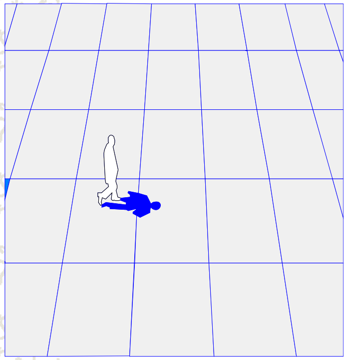
GENETIC ALGORITHMS

Matilde Santos Peñas and Jesús Manuel de la Cruz
msantos@dacya.ucm.es, jmcruz@fis.ucm.es
Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid, Spain

South Brazil Section Chapter of the
IEEE Computational Intelligence Society

Brazil, 27 October 2010

GENETIC ALGORITHM



"Genetic Algorithms are good at taking large, potentially huge search spaces and navigating them, looking for optimal combinations of things, solutions you might not otherwise find in a lifetime."

- Salvatore Mangano
Computer Design, May 1995

OUTLINE

- OVERVIEW
- CHARACTERISTICS
- SIMPLE GENETIC ALGORITHM (SGA)
- EXAMPLE
- OTHER OPERATORS
- EXAMPLE
- APPLICATIONS

EVOLUTIONARY COMPUTATION

- Evolutionary computation consists of machine learning optimization and classification paradigms that are roughly based on evolution mechanisms such as biological genetics and natural selection
- The EC field comprises four main areas:
  – genetic algorithms
  – evolutionary programming
  – evolution strategies
  – genetic programming.

EC PARADIGMS

- EC paradigms differ from traditional search and optimization ones in that EC paradigms:

1) Use a population of points in their search,
2) Use direct “fitness” information, instead of function derivatives or other related knowledge, and ,
3) Use probabilistic rather than deterministic transition rules.

EC QUICK OVERVIEW

- A.S. Fraser, 1950’s, Australia, biologist using computers to simulate natural genetic systems
- J.D. Bagley (first used term GA in his 1967 Ph.D)
- L.J. Fogel, Evolutionary programming, 1960’s
- I. Rechenberg, Evolution strategy, 1960’s
- Latane, Particle swarm optimization (Social impact theory)

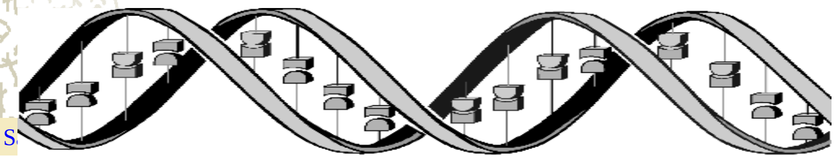
GA QUICK OVERVIEW

- J. Holland (1975), “Adaptation in natural and artificial systems”
- DeJong’s dissertation on GAs, 1975
- D. Goldberg, book “GA in search, optimization, and machine learning”, 1989
- Since 1985, interest explosion
  - International Conferences
  - Scientific Journals
  - Web resources
  - Widely-used today in business, scientific and engineering circles

GA MAIN IDEA

Directed search algorithms based on the mechanics of biological evolution

An initial set of individuals evolve along generations by reproduction and mutation, to become the best individuals, the ones who survive.



GA CHARACTERISTICS

- To understand the adaptive processes of natural systems
- To design artificial systems software that retains the robustness of nature system
- Typically applied to discrete optimization
- Attributed features:
  - not too fast
  - good heuristic for combinatorial problems
- Many variants, e.g., reproduction models, operators

ADVANTAGES

- Concept is easy to understand
- Modular, separate from application
- Supports multi-objective optimization
- Good for "noisy" environments
- Always gives an answer; answer gets better with time
  - An acceptable good solution in a reasonable time
  - At any stage there is a solution (maybe not the best but a good one)
- Inherently parallel; easily distributed

MORE BENEFITS

- Many ways to speed up and improve a GA-based application as knowledge about the problem domain is gained
- Easy to exploit previous or alternate solutions
- Flexible building blocks for hybrid applications (Fuzzy+GA, GA+NN, etc)
- Substantial history and range of use

DISADVANTAGES

- You may not find the optimal solution
- The solution space has to take into account only the feasible solutions
- Definition of the evaluation function that includes the knowledge of the problem
- They are not specialized algorithms
  - Application dependence
- The success depends on the designer

WHEN TO USE A GA

- Alternative solutions are too slow or overly complicated
- Need an exploratory tool to examine new approaches
- Problem is similar to one that has already been successfully solved by using a GA
- Want to hybridize with an existing solution
- Benefits of the GA technology meet key problem requirements

APPLICATIONS

Complex systems (N-P problems) for which an acceptable solution is enough

- Optimization
- Searching
- Parameters identification
- Pattern recognition
- Machine learning
- …

FIELDS

- Robotics
- Planning
- Optimization (circuits, controllers, neural networks, etc.)
- Simulation
- Hardware design and implementation
- Data mining
- Identification
- etc

SOME GA APPLICATION TYPES

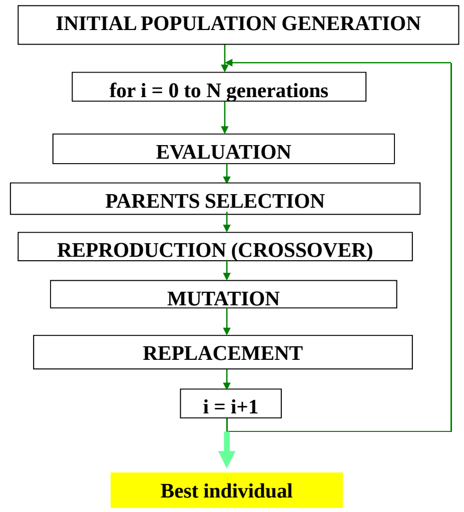
Domain	Application Types
Control	gas pipeline, pole balancing, missile evasion, pursuit
Design	semiconductor layout, aircraft design, keyboard configuration, communication networks
Scheduling	manufacturing, facility scheduling, resource allocation
Robotics	trajectory planning
Machine Learning	designing neural networks, improving classification algorithms, classifier systems
Signal Processing	filter design
Game Playing	poker, checkers, prisoner's dilemma
Combinatorial Optimization	set covering, travelling salesman, routing, bin packing, graph colouring and partitioning

SIMPLE GENETIC ALGORITHM

- Holland's original GA is now known as the simple genetic algorithm (SGA)
- Other GAs use different:
  - Representations
  - Mutations
  - Crossovers
  - Selection mechanisms

SIMPLE GENETIC ALGORITHM

It is just an algorithm to solve a problem



GA COMPONENTS

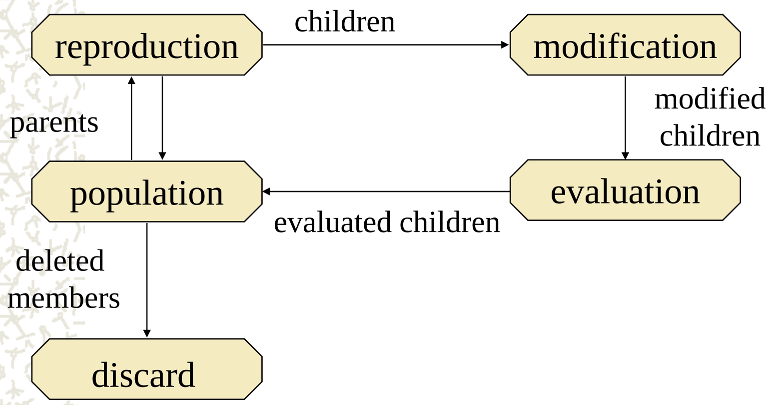
A problem to solve, and ...

- Encoding technique *(gene, chromosome)*
- Initialization procedure *(creation)*
- Evaluation function *(environment)*
- Selection of parents *(reproduction)*
- Genetic operators *(mutation, recombination)*
- Parameter settings *(practice and art)*

SIMPLE GA PSEUDO-CODE

```
{
    initialize population;
    evaluate population;
    while TerminationCriteriaNotSatisfied
    {
        select parents for reproduction;
        perform recombination and mutation;
        evaluate population;
    }
}
```

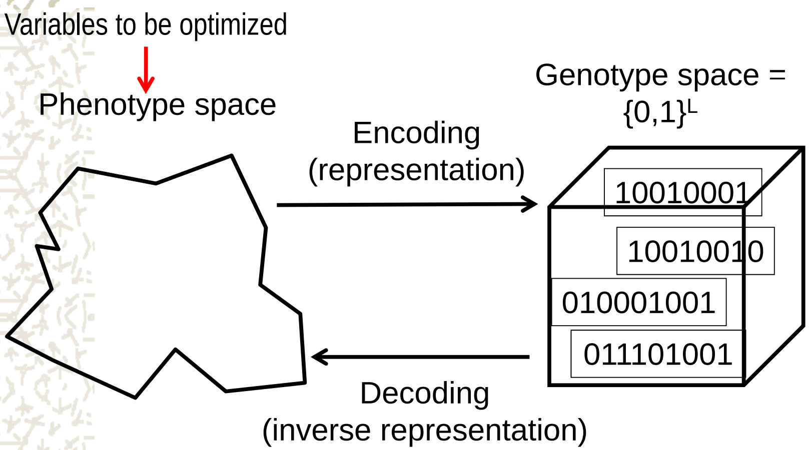
THE GA CYCLE OF REPRODUCTION



INITIAL POPULATION

- Population of *n* individuals (potential solutions)
- Individual = data structure (string of characters or chromosomes from an alphabet Φ)
  - CROMOSOME = $a_1\ a_2\ a_3 \ldots a_m$, $a_i \in \Phi$
  - Chromosome: <u>parameters to be optimized</u>
- Each element of the chromosome, a_i, a gene
  - Allele: value
  - Locus: position in the string
- Size: enough to cover the solution space

REPRESENTATION



REPRESENTATION

Chromosomes could be:

- Bit strings (0101 ... 1100)
- Real numbers (43.2 -33.1 ... 0.0 89.2)
- Permutations of elements (E11 E3 E7 ... E1 E15)
- Lists of rules (R1 R2 R3 ... R22 R23)
- Program elements (genetic programming)
- ... any data structure ...

REPRESENTATION

- Binary string
  - Easy operations
  - Lowest cardinality of the alphabet (simples searching)
  - Algorithms convergence proved
  - Variants (BCD, Gray code, etc)
- Other types of representations
  - Different cardinality alphabets

POPULATION INITIALIZATION

- SIZE:
  - Start with moderate sized population (50-500)
  - Population size tends to increase linearly with individual string length (not exponentially)
- RANDOMLY:
  - To cover all the space
  - To prove the algorithm
- HEURISTICALLY (include promising values):
  - Assure the variety of solutions (do not skew population significantly)
  - Avoid the premature convergence of the algorithm
  - Include constraints

EVALUATION

- FITNESS FUNCTION
  - Each individual is assigned a fitness measure
    - How good it is as solution
    - More chances of surviving
  - The link between the GA and the problem it is solving
    - Specific
  - Normalization (scale fitness values)
    - Better discrimination

REPRODUCTION

- PARENT SELECTION
- CHROMOSOME MODIFICATION
  - Genetic operators:
    - Crossover (recombination)
    - Mutation

Genetic operators significantly enhance parallel search capabilities

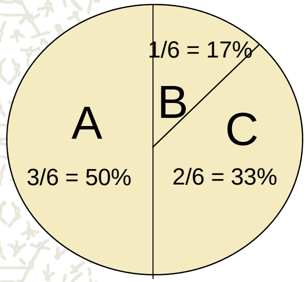
PARENT SELECTION

- Parents are selected at random with selection chances biased in relation to chromosome evaluation

  – Better individuals get higher chance
    - Chances proportional to fitness

SELECTION

- Implementation: **roulette wheel technique**
  - Assign to each individual a part of the roulette wheel
  - Spin the wheel n times to select n individuals



fitness(A) = 3

fitness(B) = 1

fitness(C) = 2

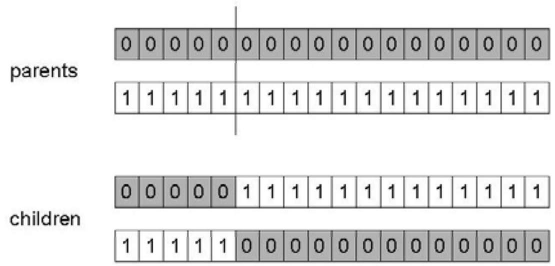
REPRODUCTION CYCLE

1. Select parents for the mating pool
   (size of mating pool = population size)
2. Shuffle the mating pool
3. For each consecutive pair apply crossover with probability $\mathbf{p_c}$, otherwise copy parents
4. For each offspring apply mutation (bit-flip with probability $\mathbf{p_m}$ independently for each bit)
5. Replace the whole population with the resulting offspring

CROSSOVER: RECOMBINATION

1-POINT CROSSOVER

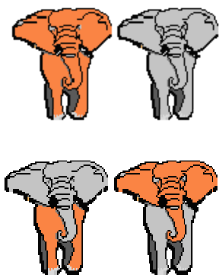
- Choose a random point on the two parents
- Split parents at this crossover point
- Create children by exchanging tails
- P_c typically in range (0.6, 0.9)



CROSSOVER: RECOMBINATION

Crossover is a critical feature of genetic algorithms:

- It greatly accelerates search early in evolution of a population
- It leads to effective combination of schemata (subsolutions on different chromosomes)
- Often start with relatively high xover rate, and reduce it during the run



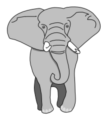
MUTATION: LOCAL MODIFICATION

- Alter each gene independently with a probability p_m
- p_m is called the mutation rate
  - Typically between 1/pop_size and 1/ chromosome_length
  - Usually held constant or increased during run (when fitness variability drops below some threshold)

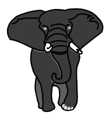
parent	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
child	0	1	0	0	1	0	1	1	0	0	0	1	0	1	1	0	0	1

MUTATION: LOCAL MODIFICATION

Before:	(1	0	1	1	0	1	1	0)
After:	(0	1	1	0	0	1	1	0)



Before:	(1.38	-69.4	326.44	0.1)
After:	(1.38	-67.5	326.44	0.1)



- Causes movement in the search space (local or global)
- Restores lost information to the population

SURVIVOR SELECTION

- A new population is generated each generation)
- Population replacement
  - Generational GA
    - Entire populations replaced with each iteration
  - Steady state GA
    - A few members replaced each epoch
      - Elitism: the best individual is copied into the next generation
      - New individuals randomly generated
      - Generational gap: replace x percent (worst individuals)

Population typically remains the same size

TERMINATION CRITERIA

- Computational time
- Number of generations
  – Depends on the complexity of the problem
- When the solution converges to a enough good value (if known)
  – Population member(s) with > specified fitness
- No change in max fitness in *m* generation

ISSUES

- Choosing basic implementation issues:
  – representation
  – population size, mutation rate, ...
  – selection, deletion policies
  – crossover, mutation operators
- Termination Criteria
- Performance, scalability
- Solution is only as good as the evaluation function (often hardest part)

SGA SUMMARY

Representation	Binary strings
Recombination	N-point or uniform
Mutation	Bitwise bit-flipping with fixed probability
Parent selection	Fitness-Proportionate
Survivor selection	All children replace parents
Speciality	Emphasis on crossover

SUMMARY OF GA PROCESS

1. Select the initial population (usually randomly).
2. Select percent probability of crossover (often .6-.8) and of mutation (often about .001).
3. Calculate the fitness value for each population member.
4. Normalize fitness values and use to determine probabilities for reproduction.
5. Reproduce new generation with the same number of members, using probabilities from 3.
6. Pair off strings to cross over randomly.
7. Select crossing sites (often 2) randomly for each pair.
8. Mutate on a bit-by-bit basis.
9. If more generations, go to step 2.
10. If completed, stop and output results.

AN EXAMPLE AFTER GOLDBERG '89

- Simple problem: max x^2 over {0,1,…,31}
- GA approach:
  - Representation: binary code, e.g. 01101 ↔ 13 (2^5)
  - Population size: 4 individuals
  - 1-point xover, bitwise mutation
  - Roulette wheel selection
  - Random initialization
- We show one generational cycle done by hand

X^2 EXAMPLE: SELECTION

String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

X^2 EXAMPLE: CROSSOVER

String no.	Mating pool	Crossover point	Offspring after xover	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 | 1	4	0 1 1 0 0	12	144
2	1 1 0 0 | 0	4	1 1 0 0 1	25	625
2	1 1 | 0 0 0	2	1 1 0 1 1	27	729
4	1 0 | 0 1 1	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

X^2 EXAMPLE: MUTATION

String no.	Offspring after xover	Offspring after mutation	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	$\boxed{1}$ 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 $\boxed{1}$ 0 0	18	324
Sum				2354
Average				588.5
Max				729

THE SIMPLE GA

- Has been subject of many (early) studies
  - still often used as benchmark for novel GAs
- Shows many shortcomings, e.g.
  - Representation is too restrictive
  - Mutation & crossovers only applicable for bit-string & integer representations
  - Selection mechanism sensitive for converging populations with close fitness values
  - Generational population model can be improved with explicit survivor selection

REVIEW OF GA OPERATIONS

- Representation of variables
- Population size
- Population initialization
- Fitness calculation
- Reproduction
- Crossover
- Inversion
- Mutation
- Selecting number of generations

OTHER REPRESENTATIONS

Consider example problem,
where 127 is 01111111 and 128 is 10000000

The smallest fitness change requires change in every bit

- Gray coding of integers (still binary chromosomes)
  - Gray coding is a mapping that means that small changes in the genotype cause small changes in the phenotype (unlike binary coding). "Smoother" genotype-phenotype mapping makes life easier for the GA

OTHER REPRESENTATIONS

Nowadays it is generally accepted that it is better to encode numerical variables directly as

- Integers
- Floating point variables

- Some software converts dynamic range and resolution into appropriate bit strings

- Different alphabets possible

INTEGER REPRESENTATIONS

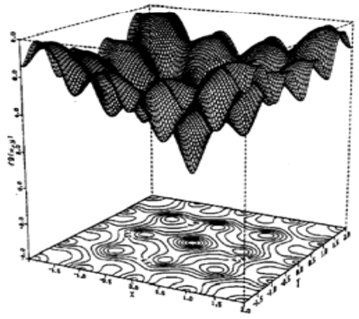
- Some problems naturally have integer variables, e.g. image processing parameters
- Others take *categorical* values from a fixed set e.g. {blue, green, yellow, pink}
- N-point / uniform crossover operators work
- Extend bit-flipping mutation to make
  - "creep" i.e. more likely to move to similar value
  - Random choice (esp. categorical variables)
  - For ordinal problems, it is hard to know correct range for creep, so often use two mutation operators in tandem

REAL VALUED PROBLEMS

- Many problems occur as real valued problems, e.g. continuous parameter optimization $f: \Re^n \rightarrow \Re$
- Illustration: Ackley's function (often used in EC)

$$f(\overline{x}) = -c_1 \cdot exp\left(-c_2 \cdot \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - exp\left(\frac{1}{n} \cdot \sum_{i=1}^{n} cos(c_3 \cdot x_i)\right) + c_1 + 1$$

$$c_1 = 20,\ c_2 = 0.2,\ c_3 = 2\pi$$



MAPPING REAL VALUES ON BIT STRINGS

$z \in [x,y] \subseteq \Re$ represented by $\{a_1,\ldots,a_L\} \in \{0,1\}^L$

- $[x,y] \rightarrow \{0,1\}^L$ must be invertible (one phenotype per genotype)
- $\Gamma: \{0,1\}^L \rightarrow [x,y]$ defines the representation

$$\Gamma(a_1,\ldots,a_L) = x + \frac{y-x}{2^L - 1} \cdot \left(\sum_{j=0}^{L-1} a_{L-j} \cdot 2^j\right) \in [x,y]$$

- Only 2^L values out of infinite are represented
- L determines possible maximum precision of solution
- High precision → long chromosomes (slow evolution)

ALTERNATIVE CROSSOVER OPERATORS

- Performance with 1 Point Crossover depends on the order that variables occur in the representation
  - more likely to keep together genes that are near each other
  - Can never keep together genes from opposite ends of string
  - This is known as *Positional Bias*
  - Can be exploited if we know about the structure of our problem, but this is not usually the case

N-POINT CROSSOVER

- Choose n random crossover points
- Split along those points
- Glue parts, alternating between parents
- Generalisation of 1 point (still some positional bias)

parents	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
children	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	1	1	1
	1	1	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	0

UNIFORM CROSSOVER

- Assign 'heads' to one parent, 'tails' to the other
- Flip a coin for each gene of the first child
- Make an inverse copy of the gene for the second child
- Inheritance is independent of position

parents	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
children	0	1	0	0	1	0	1	1	0	0	0	1	0	1	1	0	0	1
	1	0	1	1	0	0	0	0	1	1	1	0	1	0	0	1	1	0

CROSSOVER OR MUTATION?

- Decade long debate: which one is better / necessary

- Answer (at least, rather wide agreement):
  – it depends on the problem, but in general, it is good to have both
  – both have another role
  – mutation-only-EA is possible, xover-only-EA would not work

CROSSOVER OR MUTATION?

Exploration: Discovering promising areas in the search space, i.e. gaining information on the problem

Exploitation: Optimising within a promising area, i.e. using information

➢There is co-operation AND competition between them

- Crossover is explorative, it makes a *big* jump to an area somewhere "in between" two (parent) areas
- Mutation is exploitative, it creates random *small* diversions, thereby staying near (in the area of) the parent

CROSSOVER OR MUTATION?

- Only crossover can combine information from two parents
- Only mutation can introduce new information (alleles)
- Crossover does not change the allele frequencies of the population (thought experiment: 50% 0's on first bit in the population, ?% after performing *n* crossovers)
- To hit the optimum you often need a 'lucky' mutation

A SIMPLE EXAMPLE

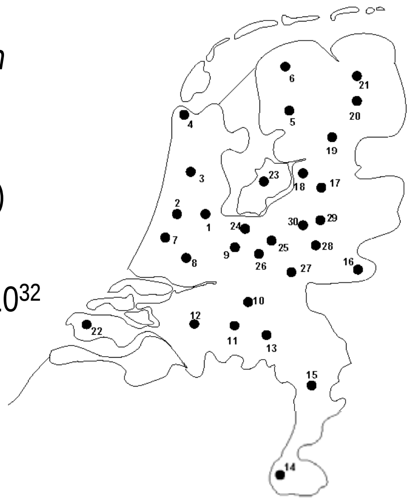
The Traveling Salesman Problem:

Find a tour of a given set of cities so that
- each city is visited only once
- the total distance traveled is minimized

PERMUTATION REPRESENTATION

- Encoding:
  - Label the cities 1, 2, … , *n*
  - One complete tour is one permutation (e.g. for n =4 [1,2,3,4], [3,4,2,1] are OK)
- Search space is BIG:

  for 30 cities there are 30! ≈ 10^{32} possible tours



REPRESENTATION

Representation is an ordered list of city numbers known as an *order-based* GA.

1) London	3) Dunedin	5) Beijing	7) Tokyo
2) Venice	4) Singapore	6) Phoenix	8) Victoria

CityList1 (3 5 7 2 1 6 4 8)

CityList2 (2 5 7 6 8 1 3 4)

CROSSOVER OPERATORS FOR PERMUTATIONS

- "Normal" crossover operators will often lead to inadmissible solutions

```
1 2 3 | 4 5          1 2 3 2 1
5 4 3 | 2 1   --->   5 4 3 4 5
```

- Many specialised operators have been devised which focus on combining order or adjacency information from the two parents

CROSSOVER

Crossover combines inversion and recombination:

```
                 *           *
Parent1   (3  5 [7  2  1  6] 4  8)
Parent2   (2  5  7  6  8  1  3  4)
------------------------------------
Child     (5  8 [7  2  1  6] 3  4)
```

This operator is called the *Order1* crossover.

MUTATION FOR PERMUTATIONS

- Normal mutation operators lead to inadmissible solutions
  - e.g. bit-wise mutation : let gene *i* have value *j*
  - changing to some other value *k* would mean that *k* occurred twice and *j* no longer occurred
- Therefore must change at least two values
- Mutation parameter now reflects the probability that some operator is applied once to the whole string, rather than individually in each position

MUTATION

Mutation involves reordering of the list:

```
                 *           *
Before:   (5  8 [7] 2  1 [6] 3  4)
                  <--------->
After:    (5  8 [6] 2  1 [7] 3  4)
```

Number of cities is fixed

REFERENCES

- Santos M., de la Cruz JM., Algoritmos genéticos. Reverté, 2005
- Goldberg, D.E. Genetic algorithms in search, optimizacion and machine learning, Adisson-Wesley, MA, 1989
- D. Beasley, D.R. Bull, Ralph R. Martin, An overview of genetic algorithms: Part I and II. University Computing, 15, 1993 (ralph.cs.cf.ac.uk/pub/papers/Gas/ga_overview1 y 2.ps)
- Haupt RL, Haupt SE, Practical genetic algorithms. Wiley, 2004
- Holland, JH, Adpatation in natural and artificial systems. U Michigan Press, 1975
- Michalewicz Z, Genetic Algorithms + Data Structures = Evolution Programs. Springer Verlag, 1992
- Mitchell, M, An introduction to genetic algorithms. MIT Press, 1998
- www.cs.cmu.edu/Groups/AI/html/repository.html

SOFTWARE REFERENCES

- MATLAB Genetic Algorithm and Direct Search Toolbox
- EVOCOM (http://pc-isa2.dacya.ucm.es/evocom
- GOAL (freeware). Visual Basic (www.geocities.com/geneticoptimization)
- Free software (C, C++, Visual Basic, Perl, …) (www.geneticprogramming.com/ga/GAsofware.html)
- www.cs.cmu.edu/Groups/AI/html/repository.html