Machine Learning

Genetic Algorithms

Genetic Algorithms

- Developed: USA in the 1970's
- Early names: J. Holland, K. DeJong, D. Goldberg
- Typically applied to:
  - discrete parameter optimization
- Attributed features:
  - not too fast
  - good for combinatorial problems
- Special Features:
  - Emphasizes combining information from good parents (crossover)
  - many variants, e.g., reproduction models, operators

Oversimplified description of evolution

- There is a group of organisms in an environment
- At some point, each organism dies
- Before it dies each organism may reproduce
- The offspring are (mostly) like the parents
  - Combining multiple parents makes for variation
  - Mutation makes for variation
- Successes have more kids than failures
  - Success = suited to the environment = lives to reproduce
- Over many generations, the population will resemble the successes more than the failures

Genotypes and phenotypes

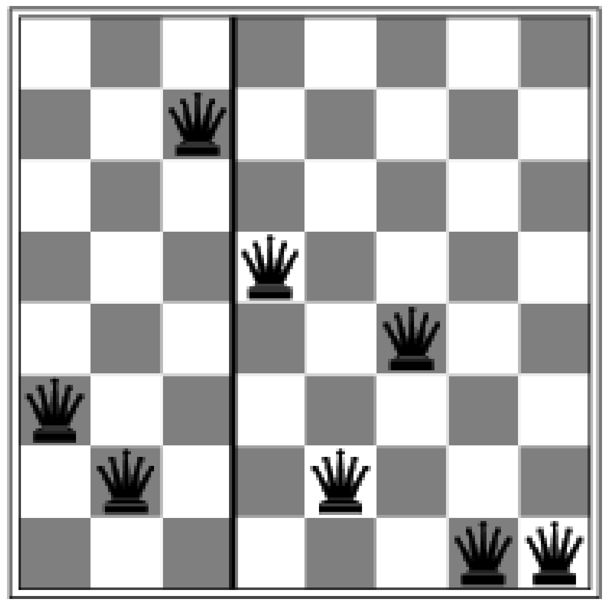
- ***Genes***: the basic instructions for building an organism
- A ***chromosome*** is a sequence of genes
- Biologists distinguish between an organism's
  - ***genotype*** (the genes and chromosomes)
  - ***phenotype*** (the actual organism)
  - Example: You might have genes to be muscle-bound, but not grow to be so for other reasons (such as poor diet)
- Genotype->Phenotype mapping can be complex
  - Can involve "development," etc.

Genotype & Phenotype (1)

Genotype

the encoding operated on by mutation and inheritance

3, 2, 7, 5, 2, 4, 1, 1



Phenotype

the “real” thing

Genotype & Phenotype (2)

Genotype: Settings for decision tree learner

Attribute_Selection = InfoGain

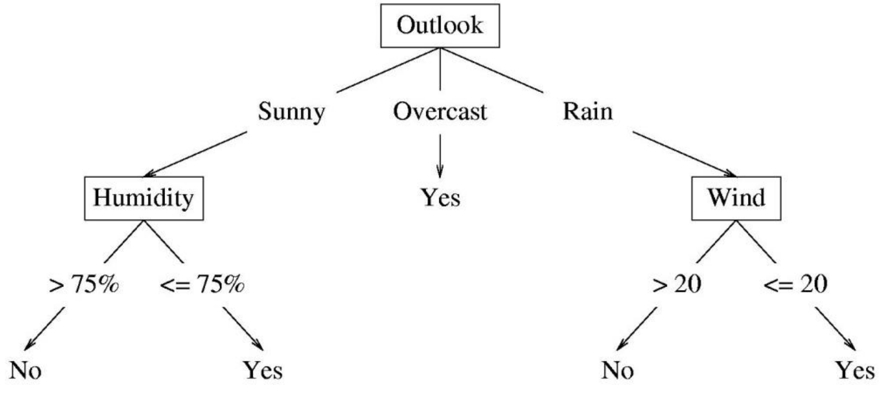
LaplacePrior = 0.2

LaplaceStrength = 2 examples

Pruning = Off

Phenotype: Decision Tree

Trained on a dataset using the settings given in genotype



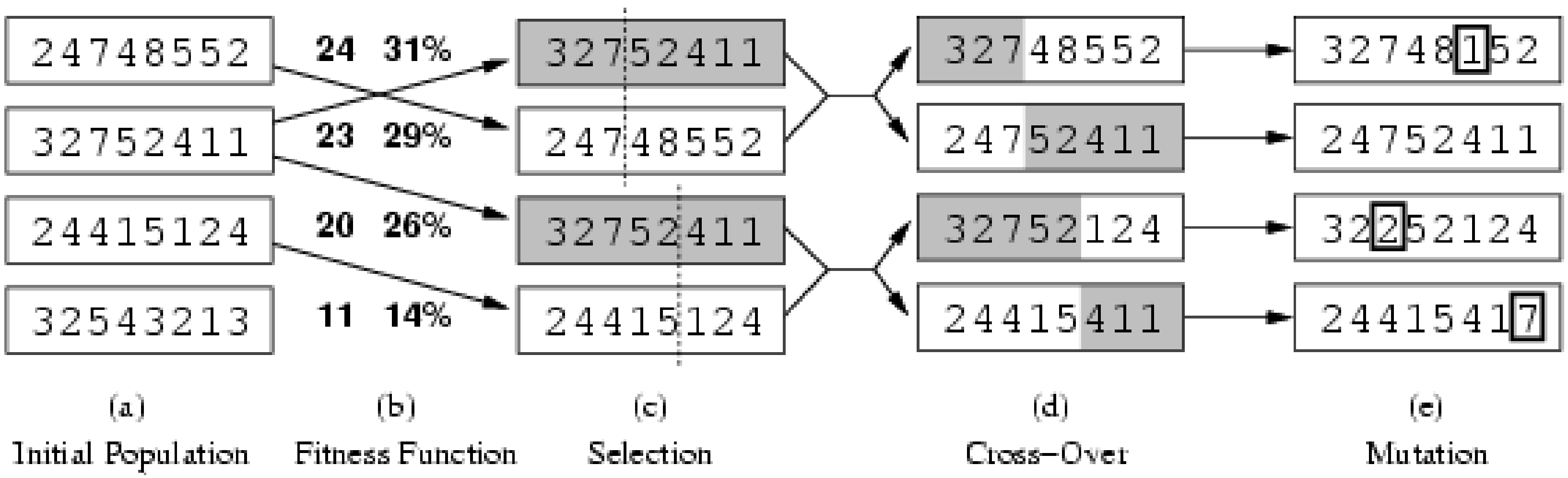
The basic genetic algorithm

- Start with a large population of randomly generated "attempted solutions" to a problem
- Repeatedly do the following:
  - Evaluate each of the attempted solutions
  - Keep a subset of these solutions (the "best" ones)
  - Use these solutions to generate a new population
- Quit when you have a satisfactory solution (or you run out of time)

Simple Genetic Algorithm (SGA)

- Define an optimization problem
  - N queens
- Define a solution encoding as a string (genotype)
  - A sequence of digits: the ith digit is the row of the queen in column i.
- Define a fitness function
  - Fitness = How many queen-pairs can attack each other (lower is better)
- Define how mutation works
  - Each digit in the gene has prob. p of changing from the parent
- Define how inheritance works
  - Chances to be a parent determined by fitness
  - Two parents, one split-point.
- Define lifespan
  - All parents die before new generation reproduces

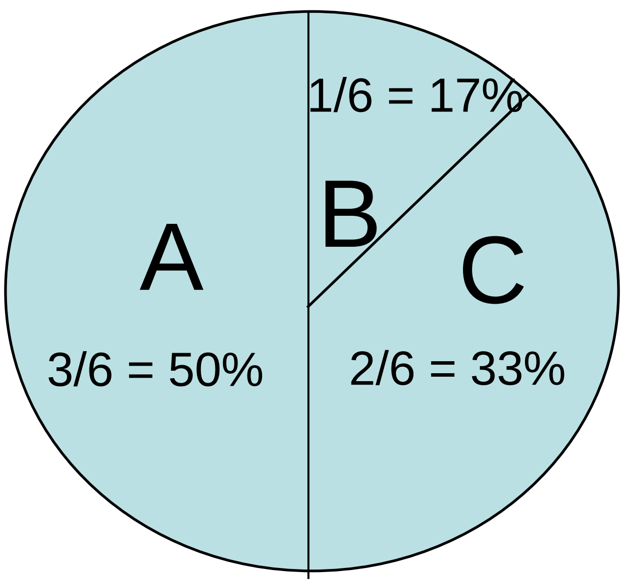
Genetic algorithms



- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc

SGA operators: Selection

- Main idea: better individuals get higher chance
  - Chances proportional to fitness
  - Implementation: roulette wheel technique
    - » Assign to each individual a part of the roulette wheel
    - » Spin the wheel n times to select n individuals



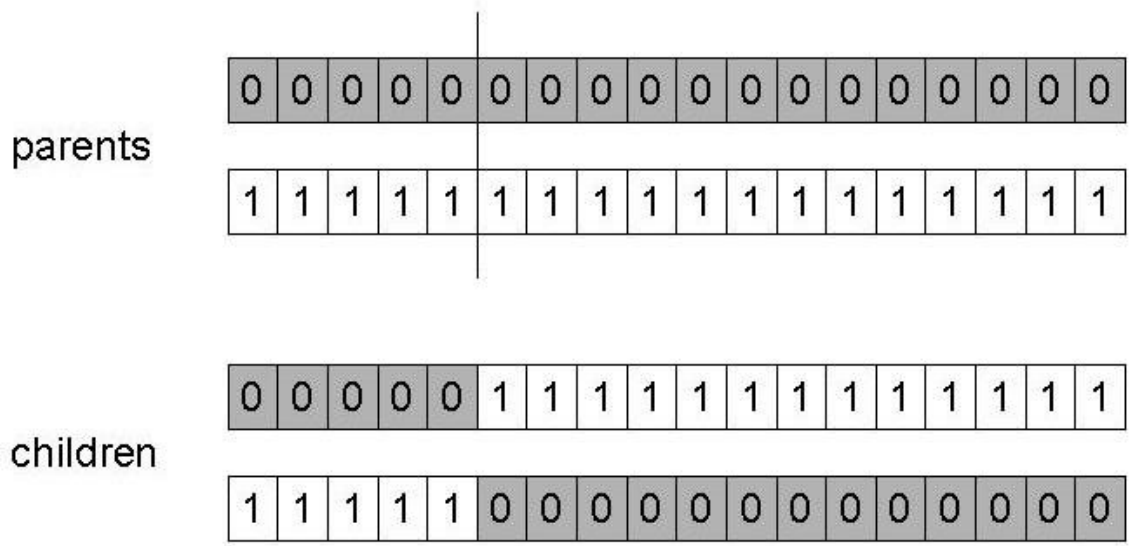
fitness(A) = 3

fitness(B) = 1

fitness(C) = 2

SGA operators: 1-point crossover

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



SGA operators: mutation

- 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

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

The simple GA (SGA)

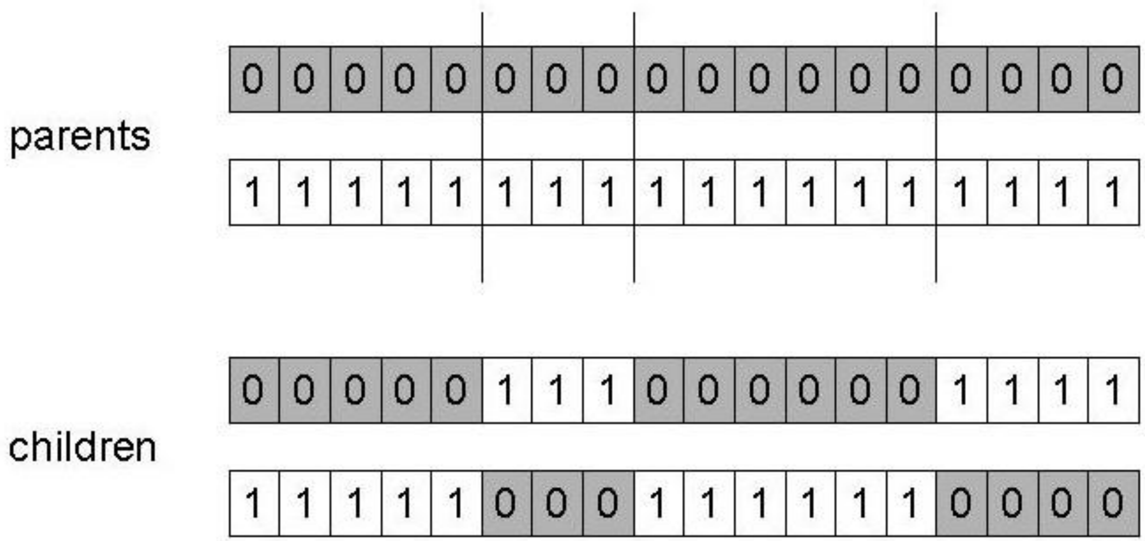
- Has been subject of many (early) studies
  - still often used as benchmark for novel GAs
- Shows many shortcomings, e.g.
  - Representation (bit strings) is restrictive
  - Selection mechanism:
    - insensitive to converging populations
    - sensitive to absolute value of fitness function
  - Generational population model can be improved with explicit survivor selection

Positional Bias & 1 Point Crossover

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

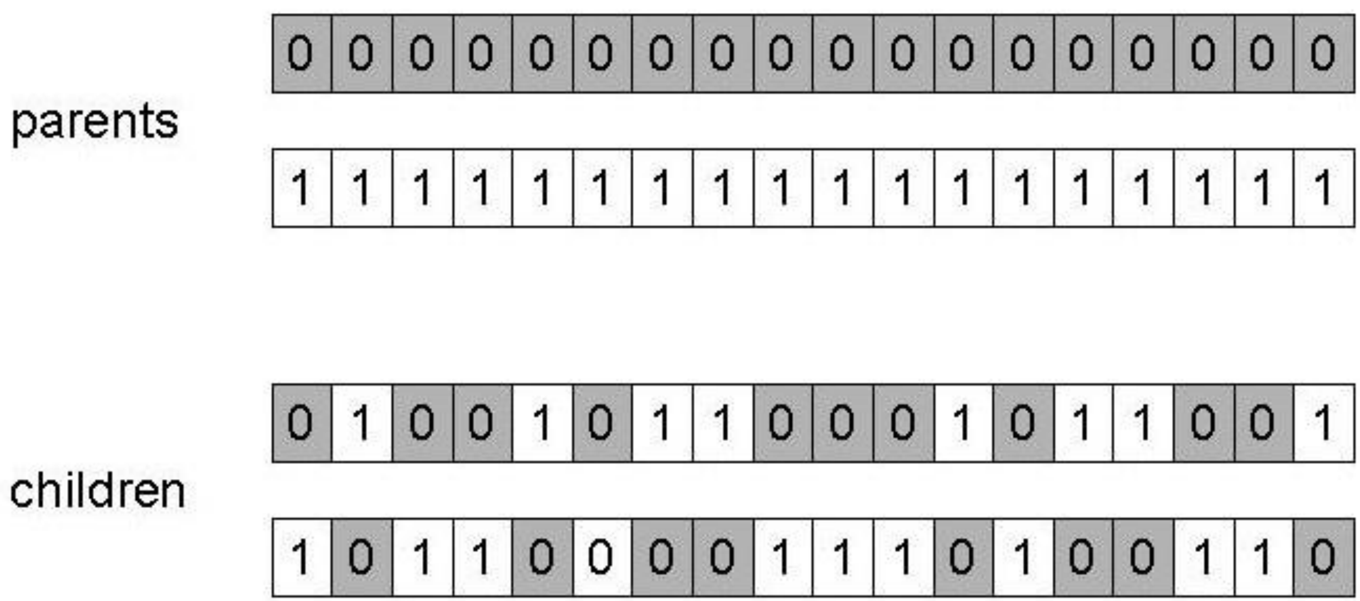
n-point crossover

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



Uniform crossover

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



Crossover OR mutation?

- Only crossover can combine information from two parents
- Only mutation can introduce new information (alleles)
- To hit the optimum you often need a ‘lucky’ mutation

Multiparent recombination

- Note that we are not restricted by nature
- Mutation uses 1 parent
- "traditional" crossover uses 2 parents
- Why not 3 or more parents?
  - Based on allele frequencies
    - p-sexual voting generalising uniform crossover
  - Based on segmentation and recombination of the parents
    - diagonal crossover generalising n-point crossover
  - Based on numerical operations on real-valued alleles
    - center of mass crossover,
    - generalising arithmetic recombination operators

Permutation Representations

- Task is (or can be solved by) arranging some objects in a certain order
  - Example: sort algorithm:
    - important thing is which elements occur before others (order)
  - Example: Travelling Salesman Problem (TSP)
    - important thing is which elements occur next to each other (adjacency)
- These problems are generally expressed as a ***permutation***:
  - if there are *n* variables then the representation is as a list of *n* integers, each of which occurs exactly once
- How can we search this representation with a GA?

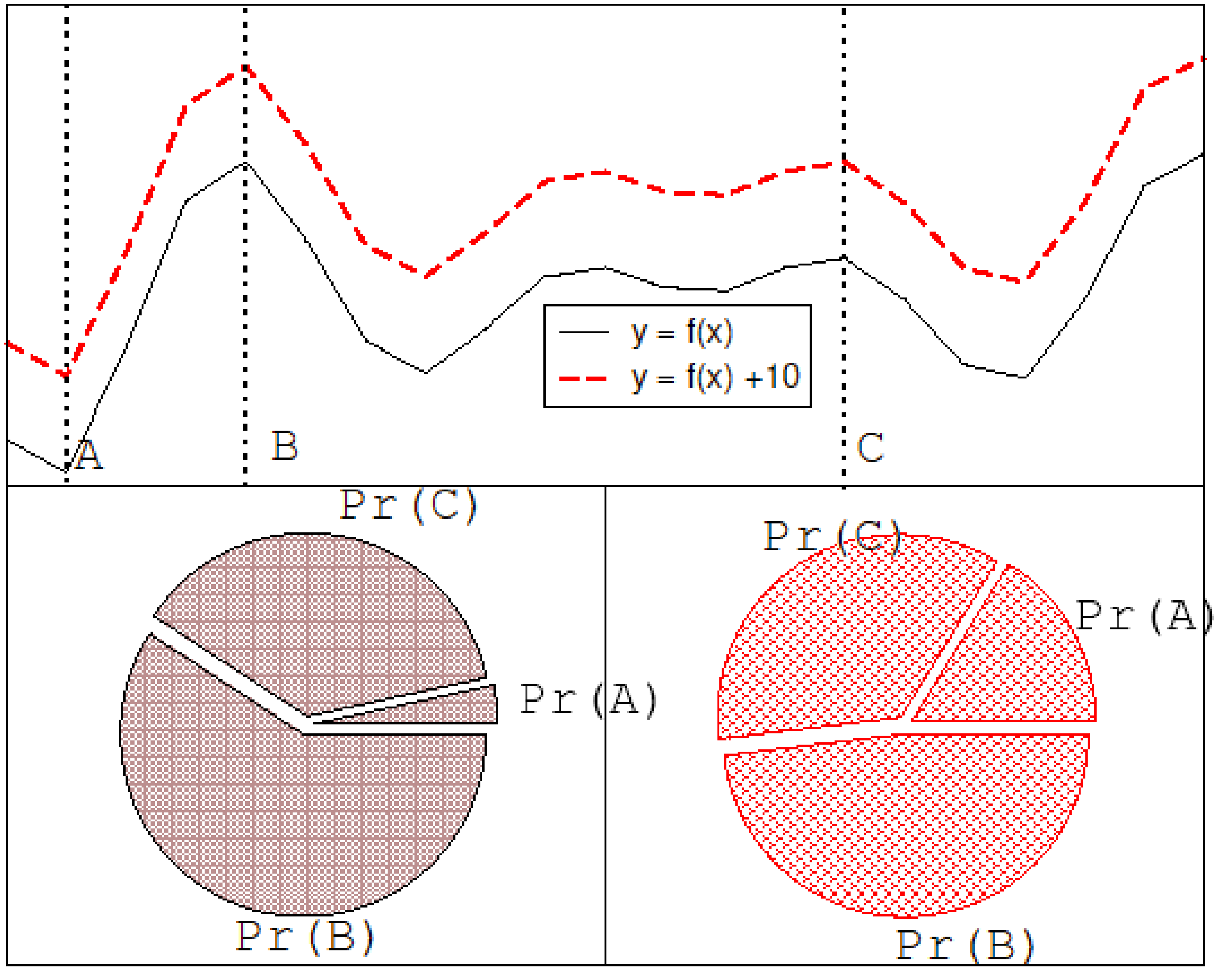
Population Models

- SGA uses a Generational model:
  - each individual survives for exactly one generation
  - the entire set of parents is replaced by the offspring
- At the other end of the scale are “Steady State” models (SSGA):
  - one offspring is generated per generation,
  - one member of population replaced,
- Generation Gap
  - the proportion of the population replaced
  - 1.0 for SGA, 1/pop_size for SSGA

Fitness-Proportionate Selection

- Premature Convergence
  - One highly fit member can rapidly take over if rest of population is much less fit
- Loss of “selection pressure”
  - At end of runs when fitness values are similar
- Highly susceptible to function transposition
- Scaling can help with last two problems
  - Windowing: $f'(i) = f(i) - \beta^{t}$
    - where β is worst fitness in this generation (or last n gen.)
  - Sigma Scaling: $f'(i) = (f(i) - \langle f \rangle)/(c \bullet \sigma_f)$
    - where c is a constant, usually 2.0

Function transposition for FPS



Rank – Based Selection

- Attempt to remove problems of FPS by basing selection probabilities on *relative* rather than *absolute* fitness
- Rank population according to fitness and then base selection probabilities on rank where fittest has rank μ and worst rank 1
- This imposes a sorting overhead on the algorithm, but this is usually negligible compared to the fitness evaluation time

Tournament Selection

- Rank based selection relies on global population statistics
  - Could be a bottleneck esp. on parallel machines
  - Relies on presence of absolute fitness function which might not exist: e.g. evolving game players
- Informal Procedure:
  - Pick *k* members at random then select the best of these
  - Repeat to select more individuals

Tournament Selection 2

- Probability of selecting *i* will depend on:
  - Rank of *i*
  - Size of sample *k*
    - higher *k* increases selection pressure
  - Whether contestants are picked with replacement
    - Picking without replacement increases selection pressure
  - Whether fittest contestant always wins (deterministic) or this happens with probability *p*
- For $k = 2$, time for fittest individual to take over population is the same as linear ranking with $s = 2 \bullet p$

Concluding remarks

- Genetic algorithms are—
  - Fun!
    - Probably why they are a subject of active research
  - Slow
    - They look at a LOT of solutions
  - Challenging to code appropriately
    - ½ the work is finding the right representations
  - A bit overhyped (at least in the 90's)
    - Though genetic algorithms *can* sometimes come up with a solution when you can see no other way of tackling the problem