Introduction to Genetic Algorithms

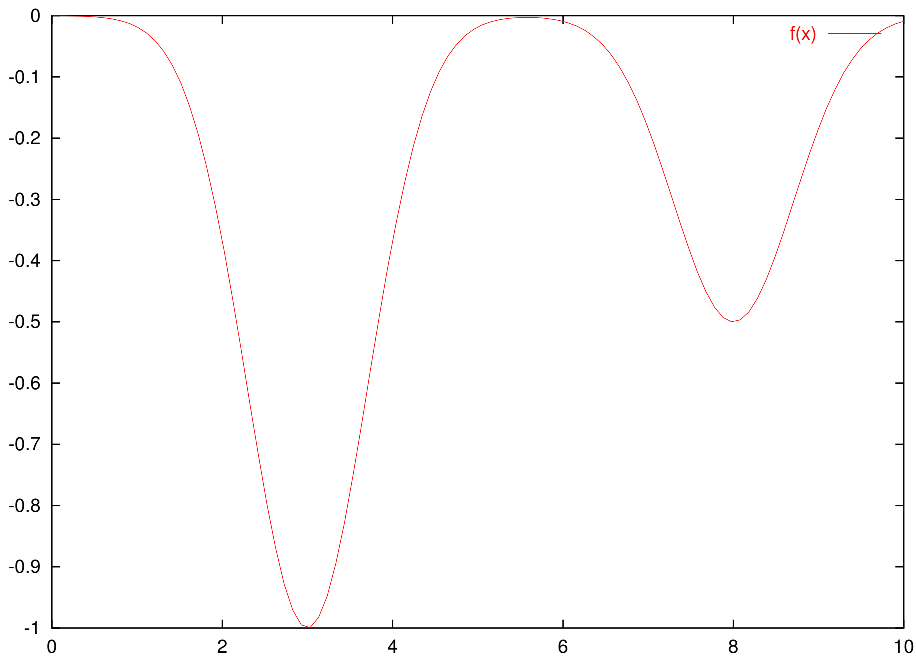
Dr. Paul A. Bromiley
Imaging Science and Biomedical Engineering

Overview

- Optimisation methods and robustness
- Simple three-operator GAs:
  - reproduction
  - crossover
  - mutation
- Why GAs work: schemata and building blocks
- Problems:
  - premature convergence
  - genetic drift
  - diversity preservation
- Multi-objective GAs

Optimisation Methods

- One aim: robustness
  - efficient + effective over many problems
- Two types of optima: local and global



- Three types of algorithm
  - calculus-based
  - enumerative
  - random

Calculus-Based Methods

- First derivative discrete: set equal to zero

$$f(x) = x^2$$

$$\frac{df(x)}{dx} = 2x = 0 \Rightarrow x = 0$$

- First derivative available at any point: hill-climb (many methods)
  - assumes meaningful derivatives (smoothness)
  - local
- Not robust (effectiveness)

Enumerative Methods

- Discretise the search space and test every point
  - not efficient
- Not robust (efficiency)

Random Methods

- True random methods e.g. random search
  - no better than enumerative methods
- Randomised methods e.g. simulated annealing
  - Boltzmann equation

$$P = exp(-\frac{E}{kT})$$

  - start with random state: energy = cost
  - random change with probability

$$p = exp(-\frac{(E_2 - E_1)}{kT})$$

  - decease temperature: annealing schedule
  - solves TSP but AS problem dependent
- Randomised methods $\neq$ random search
- Not robust (effectiveness)

Genetic Algorithms

- None of the previous methods are robust:
  - assume smoothness / local
  - inefficient
  - problem specific

- Nature optimises by evolution: cost function
  - discontinuous
  - multi-modal
  - high-dimensional
  - dynamic

- Copy evolution: Genetic Algorithms
  - investigate natural evolution
  - produce robust optimisation method

Genetic Algorithms: Outline

- Code parameters as a binary string
  - bit = gene
  - value (0,1) = allele
  - string = chromosome
- Create random population (typically ~ 100)
- Apply three operators:
  - reproduction
  - crossover
  - mutation

Genetic Algorithms: Example

- Example: optimise x^2 over x=0 to 31
- Code parameters as five bit unsigned integer
  - x=0 at 00000, x=31 at 11111
- Population=4

String	x	Fitness f(x)
01101	13	169
11000	24	576
01000	8	64
10011	19	361

- Average fitness = 293 : Max fitness = 576

Genetic Algorithms: Example 2

- Reproduction
  - copy strings with prob. $f_1/\bar{f}$
- Roulette wheel selection
  - choose strings to copy at random, weighted by their proportional fitness:

String	x	f(x)	$\frac{f_i}{f}$	Expected count	Actual count
01101	13	169	0.14	0.58	1
11000	24	576	0.49	1.97	2
01000	8	64	0.06	0.22	0
10011	19	361	0.31	1.23	1

- Note that RWS is noisy

Genetic Algorithms: Example 3

- Crossover
  - randomly pair the strings
  - randomly select a position and swap ends

MP	Mate	Crossover site	New pop	x	f(x)
0110—1	2	4	01100	12	144
1100—0	1	4	11001	25	625
11—000	4	2	11011	27	729
10—011	3	2	10000	16	256

Average fitness = 439: Max fitness = 729

- Mutation: flip bits at random
  - very low probability

Schemata

- Why does this work?

Initial pop	Final pop
01101	01100
11000	11001
01000	11011
10011	10000

  - sub-string 11*** confers high fitness
- Template = *schema* (pl. *schemata*)
  - 3^l possible schemata for string length l
  - single string contains 2^l schemata
  - population of size n contains from 2^l to $n2^l$ schemata, depending on diversity.
- Schemata have two important properties:
  - order o(H)
  - defining length $\delta(H)$
- e.g. H=1**0* : o(H)=2, $\delta(H) = 3$

Schemata 2: Reproduction

- m examples of schema H at time t
- $f(H)$ = average fitness of strings containing H

$$m(H, t{+}1) = m(H,t).n.\frac{f(H)}{\Sigma f_j} = m(H,t)\frac{f(H)}{\bar{f}}$$

- If H retains fitness $c\bar{f}$

$$m(H, t+1) = m(H,t)\frac{\bar{f} + c\bar{f}}{\bar{f}} = (1+c)m(H,t)$$

$$m(H,t) = m(H,0)(1+c)^t$$

- Above (below) average schemata grow (decline) exponentially in proportion to the ratio of their fitness to the population average

Schemata 3: Crossover and Mutation

- Crossover may disrupt a schema if the crossover site falls within the schema
  - there are $\delta(h)$ such sites
  - there are $(l-1)$ possible sites
  - apply crossover with probability p_c
  - probability of survival is:

$$p_s \geq 1 - p_c \frac{\delta(H)}{(l-1)}$$

- Mutation may disrupt a schema if a defined bit is flipped
  - there are o(H) defined bits
  - each is flipped with probability p_m.
  - probability of survival is:

$$(1-p_m)^{0(H)} \approx (1-o(H))p_m \;\; for \;\; p_m << 1$$

Schemata 4: The Fundamental Theorem of Genetic Algorithms

- Collecting terms gives

$$m(H, t{+}1) \geq m(H, t)\frac{f(H)}{\bar{f}}[1{-}p_c\frac{d(H)}{l-1}{-}o(H)p_m]$$

Fundamental Theorem of Genetic Algorithms

- Above average fitness, low-order, short defining length schemata grow exponentially
- Call these schemata *building blocks*
- Is this a good thing? k-armed bandit problem (Holland, 1975)

How Many Schemata are Usefully Processed?

- How many survive crossover with probability p_s i.e. error rate $\epsilon < (1 - p_s)$?

$$p_s = 1 - \frac{\delta(H)}{l-1}$$

$$l_s < \epsilon(l-1) + 1$$

where $\delta(H) = l_s - 1$.

- Number of schemata of length l_s or less:

$$2^{(l_s - 1)}(l - l_s + 1)$$

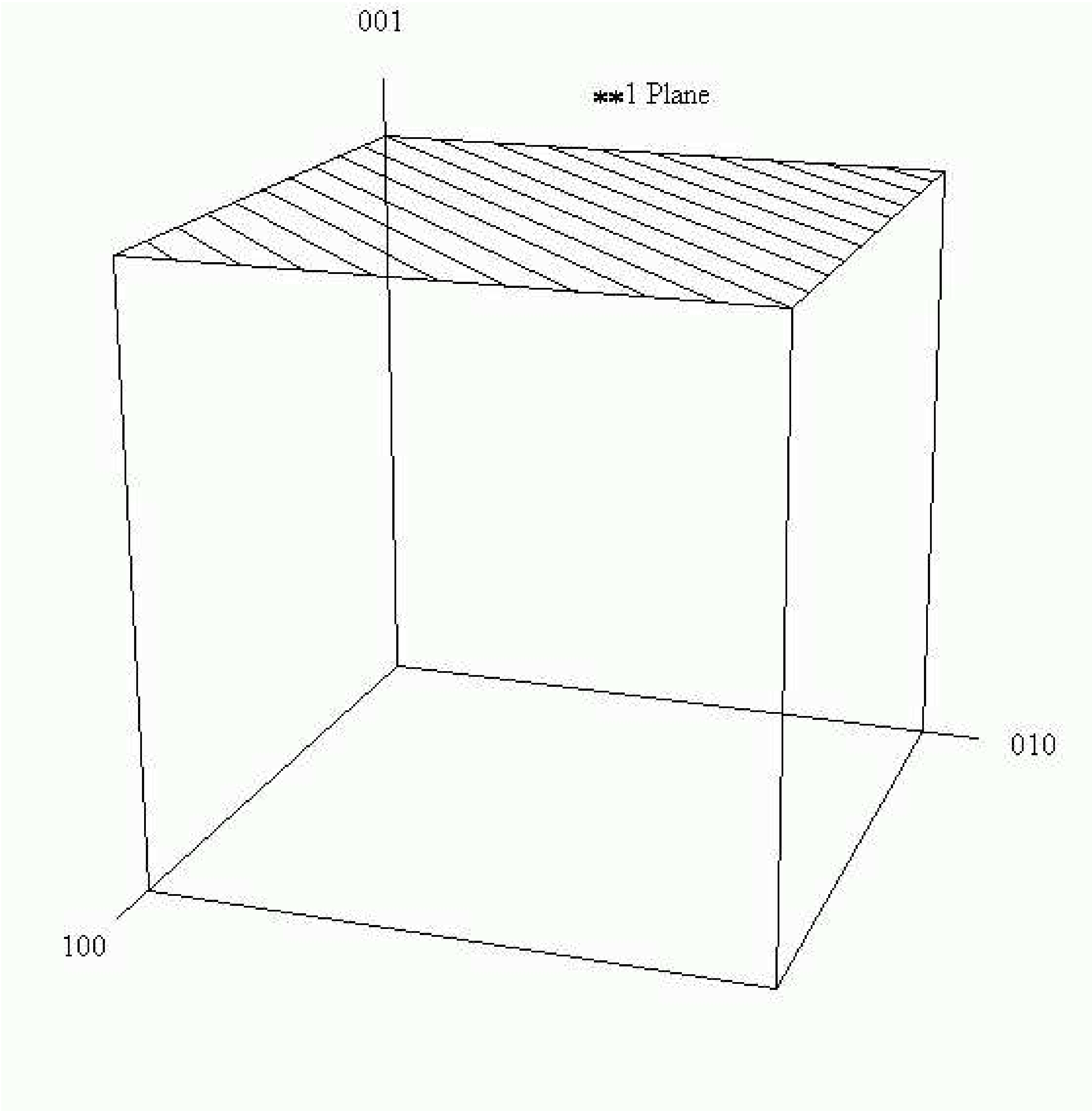
  - pick population size of $n = 2^{l_s/2}$: $\leq$1 of each schema of length $l_s/2$ or more.
  - half shorter, half longer: pick longer half

$$n_s \geq \frac{n(l - l_s + 1)2^{l_s}}{4} = \frac{(l - l_s + 1)n^3}{4}$$

- $O(n^3)$: many more schemata than strings are processed: *implicit parallelism*

Schemata as Hyperplanes

- A schema represents a hyperplane in the search space:



Codings

- Most important factor
- Coding should generate as many building blocks as possible
  - minimum cardinality alphabet ⇒ binary
- Coding should allow effective manipulation of building blocks
  - building blocks should be relevant to problem and relatively independent

Gray Codes

- Notable coding: Gray codes

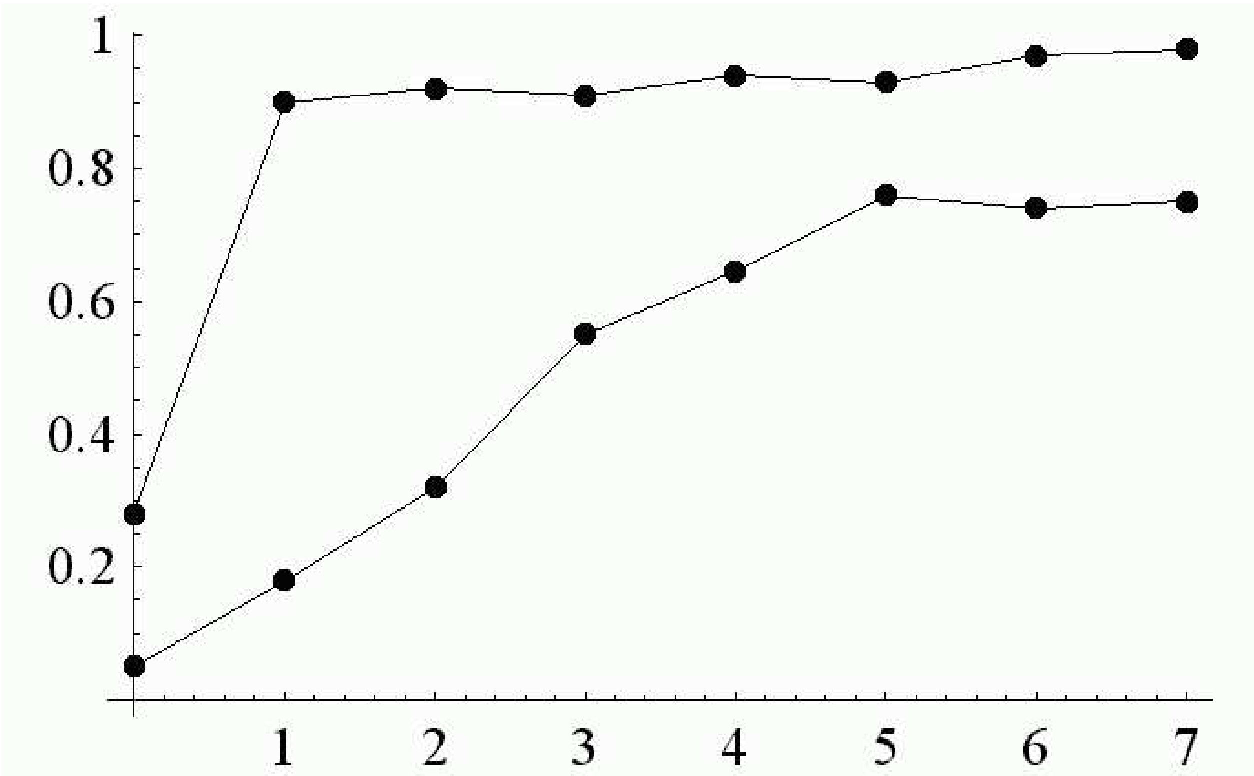
Integer	Binary	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

  – Adjacent integers differ by a single bit

- Holstien, 1971: Gray Codes
- Janikow and Michalewicz, 1991; Wright, 1991: real-valued
- Optimal coding problem dependent

Premature Convergence

- Simple example: $f(x) = (x/c)^{10}$
  - range x = 0 to 1: 30-bit binary encoding
  - $p_m = 0.03$, $p_c = 0.6$, $n = 30$ (De Jong, 1975)



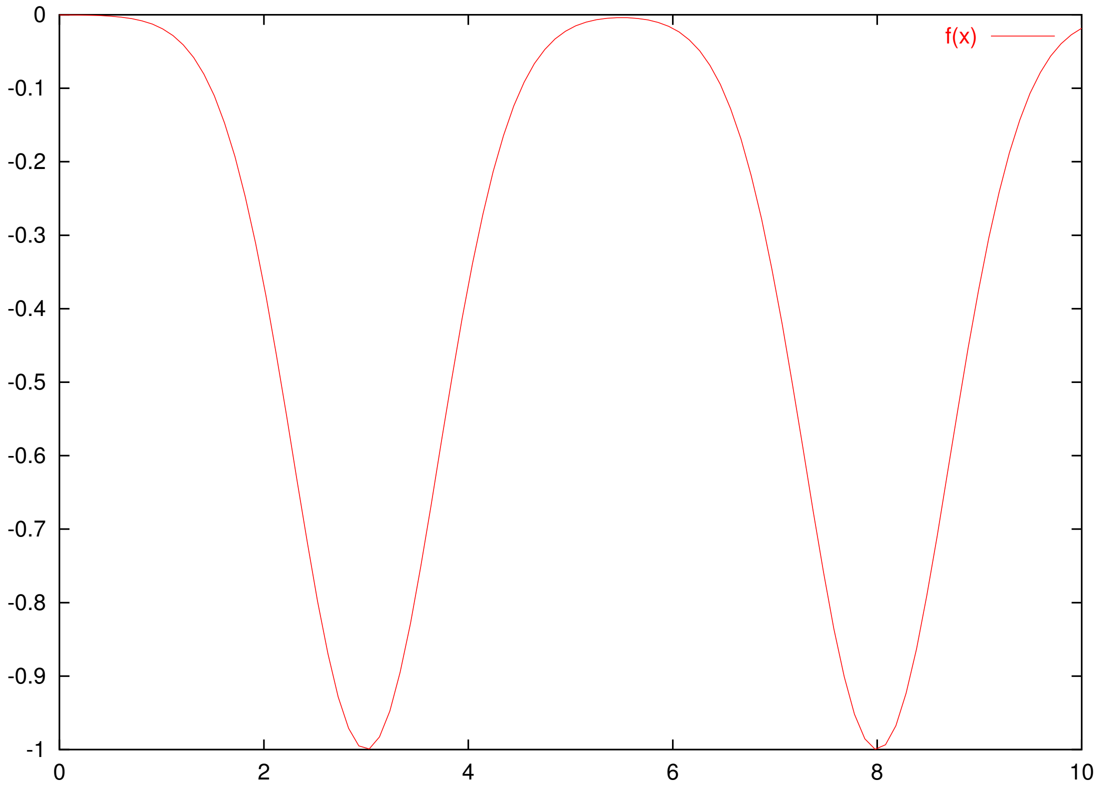
Fitness vs. Generation

- Approaches optimum, but does not reach it
  - population is degenerate by generation 7
  - diversity loss
  - premature convergence

Sources of Diversity Loss

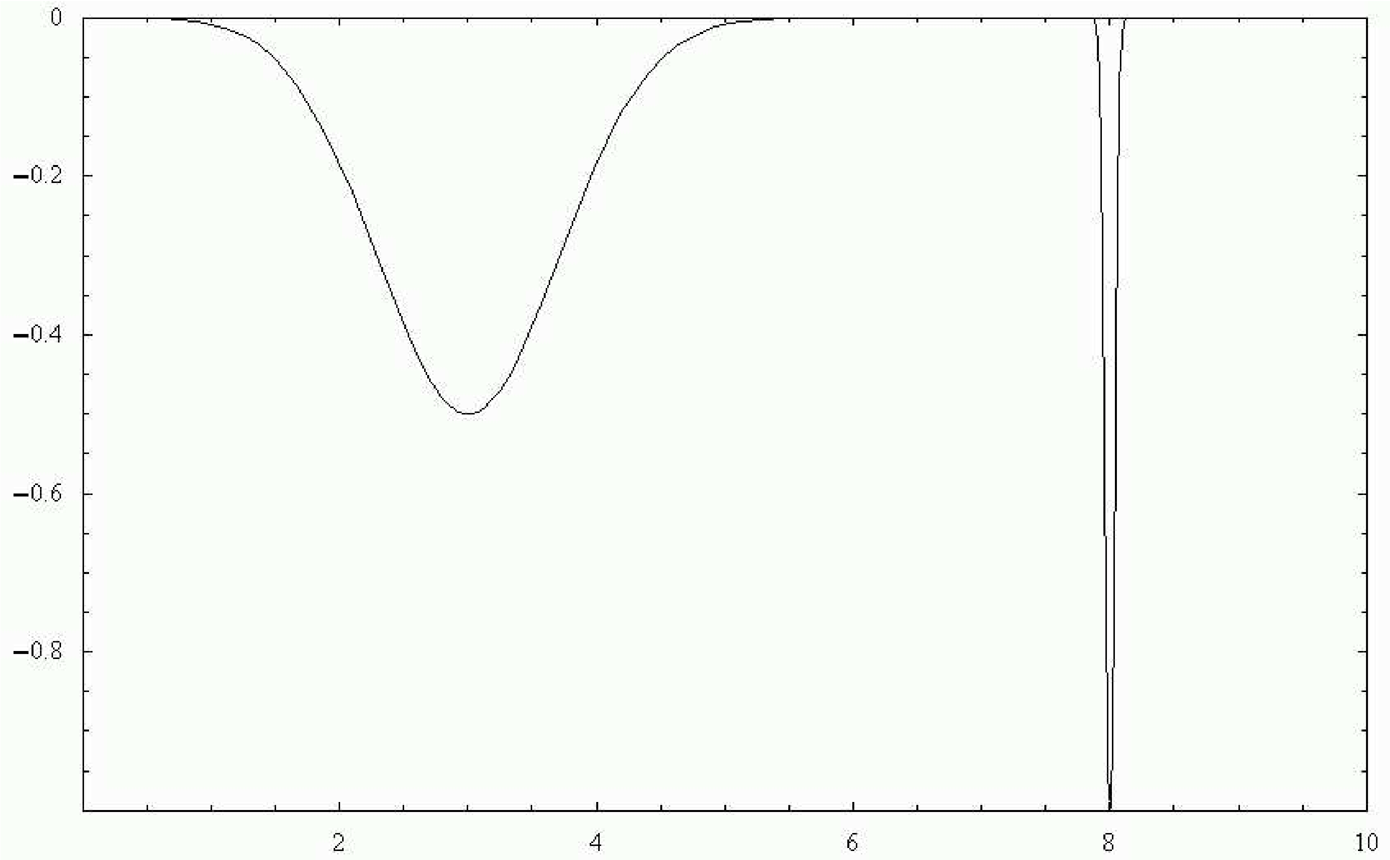
- Selection noise
  - De Jong (1975): schemata fitness from finite sample, stochastic errors in roulette wheel
- Selection pressure
  - lower fitness schemata eliminated
- Operator disruption
  - crossover and mutation destroy schemata

Sources of Diversity Loss: Selection Noise



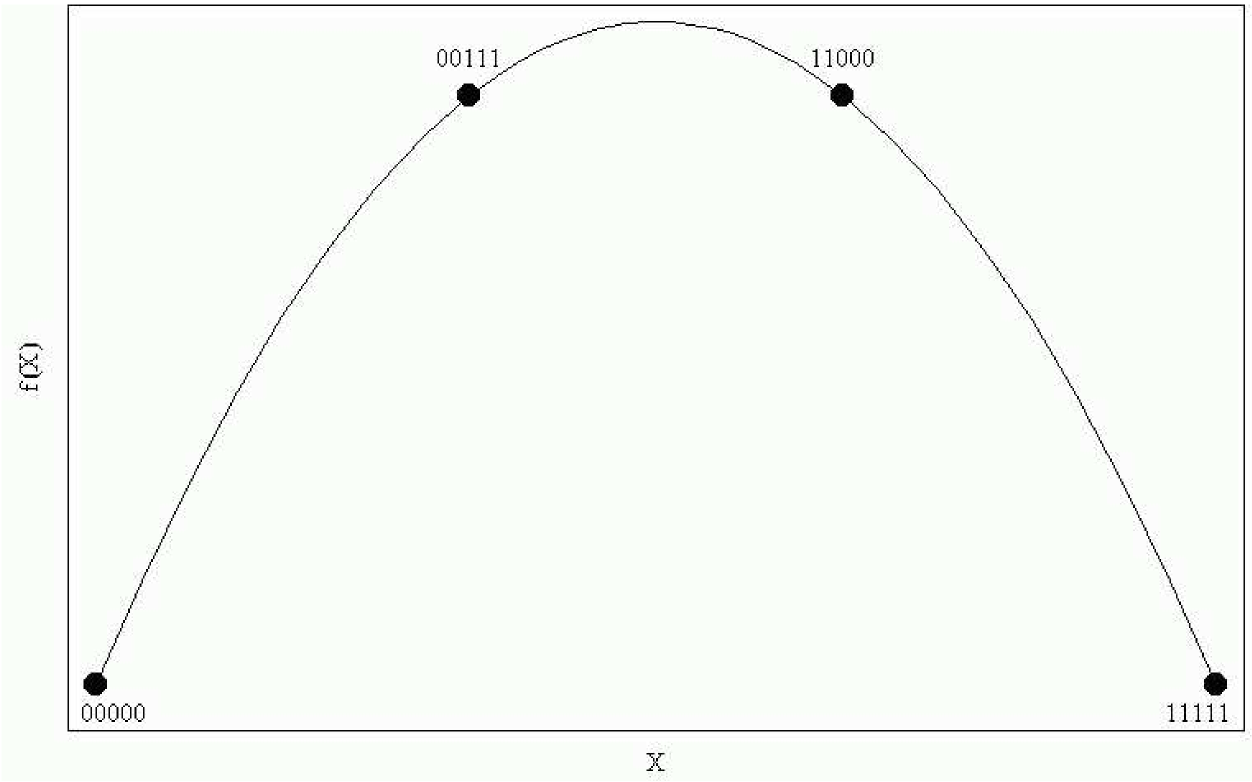
- Multi-modal functions
  - gambler's ruin
  - selection noise *will* eliminate one peak

Sources of Diversity Loss: Selection Pressure



- Broad local peak, narrow global peak
  - schemata near global peak get eliminated
  - global peak never found (deceptive function)
- Less-fit schemata eliminated, even if they provide partial solutions for global optima
  - *super-fit* individuals

Sources of Diversity Loss: Operator Disruption



- Crossover between individuals optimising different local optima is unhelpful.

Diversity Preservation

- Most GA research focuses on diversity preservation
- Many schemes:
  - alternative selection schemes
  - fitness scaling
  - crowding and preselection
  - niching and speciation
  - mating restriction

Alternative Selection Schemes

- From the simple example:

String	Expected count	Actual count
01101	0.58	1
11000	1.97	2
01000	0.22	0
10011	1.23	1

- De Jong (1975): variance of roulette wheel is main source of allele loss
- Expected value model:
  - calculate offspring count as usual $\frac{f_i}{f}$
  - reduce by 0.5 every time string is selected
  - if offspring count ≤ 0, string is never selected
  - total offspring $\leq \frac{f_i}{f} + 1$
  - influence of super-fit individuals reduced

Fitness scaling

- Start: few super-fit individuals dominate
- End: all individuals roughly same fitness: random search
- Fitness scaling: control this competition
- E.g. linear: scale fitness so that
  - $f'_{avg} = f_{avg}$
  - $f'_{max} = C_m f_{avg}$
  - $C_m = 1.2 \ \ to \ \ 2$ for $n = 50 \ \ to \ \ 100$
- Level of competition fixed
- Other scaling functions:
  - sigma
  - power law
  - review: Forrest (1985)

Crowding and preselection

Generational GA:
- replace whole population at each iteration

Steady-state GA:
- replace only a portion of the population

- Preselection:
  - Cavicchio, 1970
  - fit offspring replace their own parents
- Crowding:
  - De Jong, 1975
  - crowding: offspring replace similar individuals from subset of population
  - similarity: bitwise distance in Hamming space
- compare to speciation in natural evolution
- Mahfoud (1992): stochastic errors still lead to diversity loss

Niching and Speciation

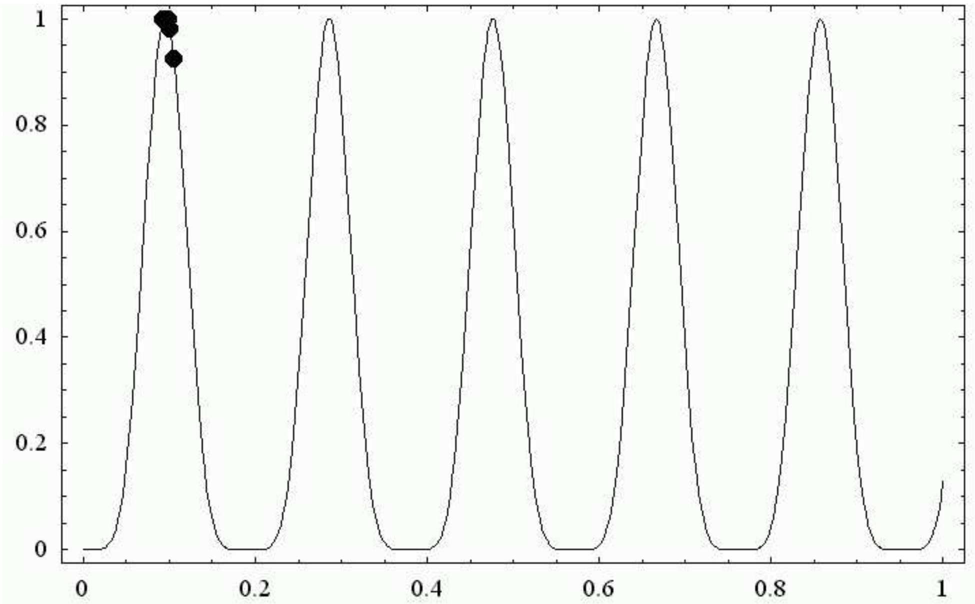
- Crowding and preselection examples of niching:
  - impose competition between like individuals
  - generate species on each local optimum
- Fitness sharing
  - Goldberg and Richardson (1987)
  - impose competition directly
  - fitness scaled by

$$f_s(x_i) = \frac{f(x_i)}{\Sigma_{j=1}^{n} s(d(x_i, x_j))}$$

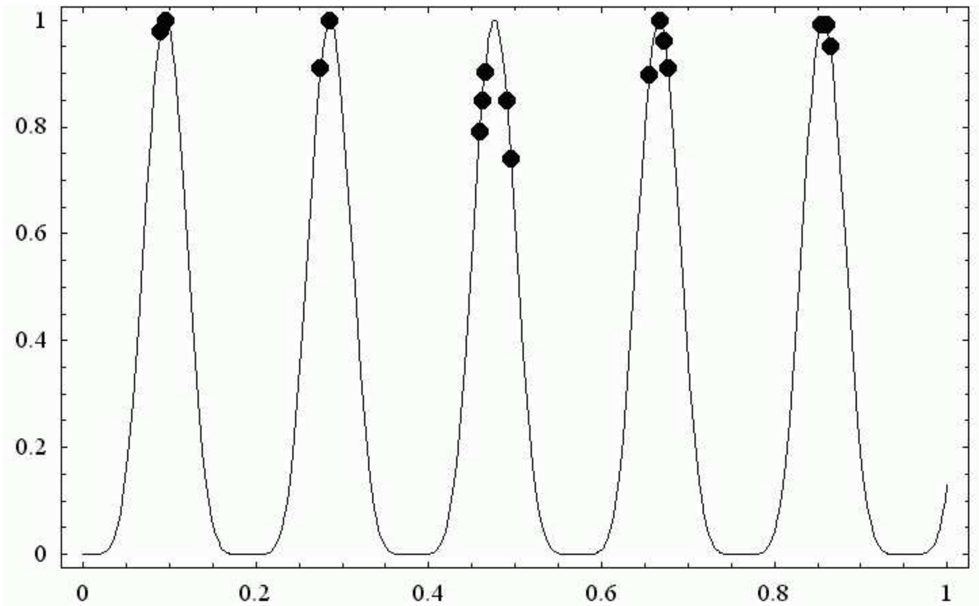
  - d = distance, s = sharing function
- Example: triangular sharing function

Niching and Speciation 2

- Results from Goldberg and Richardson



Generation 100: no mutation, no sharing

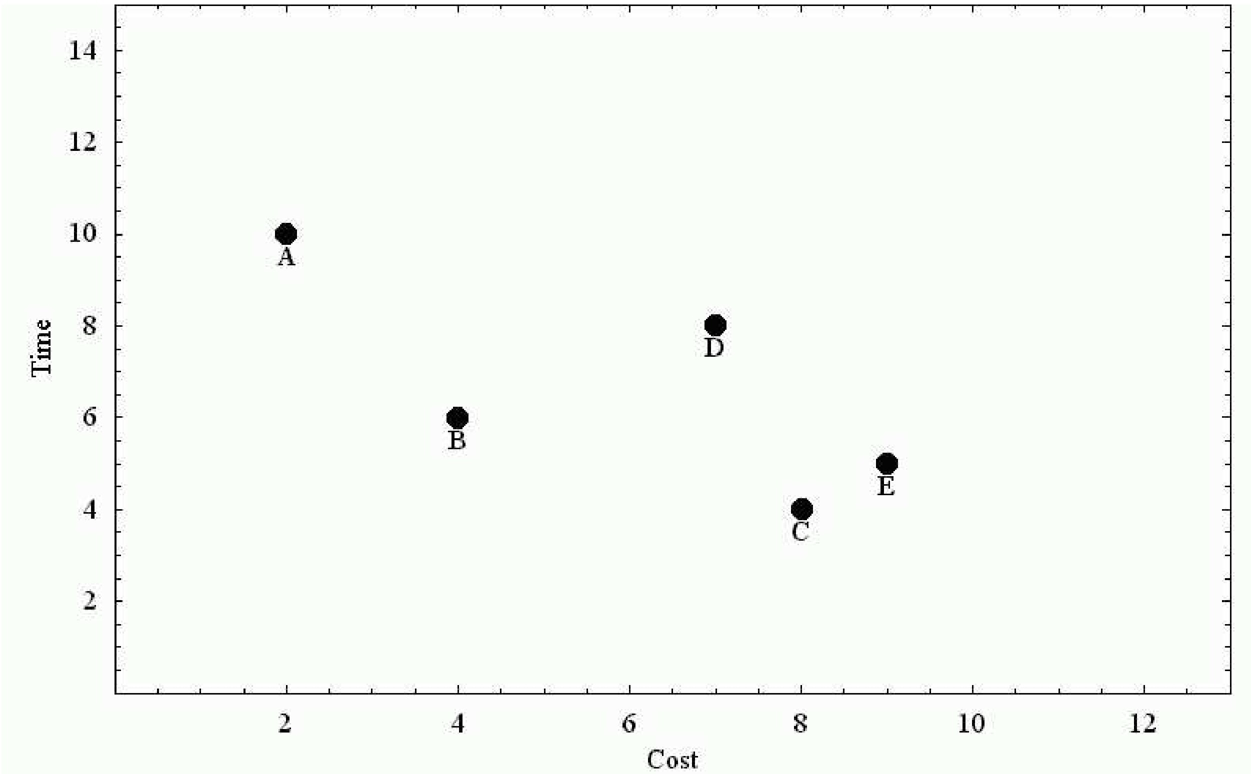


Generation 100: no mutation, sharing

Mating Restriction

- Impose niching by restricting mating
- Hollstien (1971)
  - traditional farming practises
- Line-breeding: champion individual repeatedly bred with others
  - good for unimodal cost functions
- Inbreeding with intermittent cross-breeding
  - close individuals mate if fitness goes up
  - if not, mate outside family
  - improvement for multi-modal functions

Multi-objective Optimisation



- D, E: *dominated* solutions
- A, B, C: *non-dominated* solutions
- Non-dominated solutions form the *Pareto Front*
- GAs ideal:
  - multiple individuals in population
  - entire Pareto Front in a single run
  - diversity must be preserved

Conclusions

- GAs implementation
  - code parameters as binary string
  - initialise multiple random strings
  - reproduction, crossover and mutation
- GA theory
  - schemata
  - Fundamental Theorem of Genetic Algorithms
  - implicit parallelism
- GA problems
  - selection noise, selection pressure, operator disruption
  - loss of diversity
  - hence diversity preservation methods
- GA advantages
  - robust
  - multi-objective optimisation
  - suitable for parallel architectures