Genetic Algorithms and Genetic Programming

Lecture 10: (27/10/09)

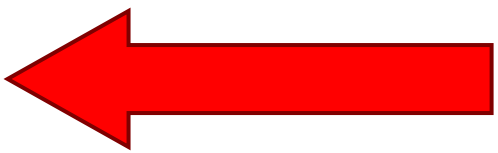
More recent trends and Practical issues in Evolutionary Computation

Michael Herrmann

michael.herrmann@ed.ac.uk, phone: 0131 6 517177, Informatics Forum 1.42

Overview

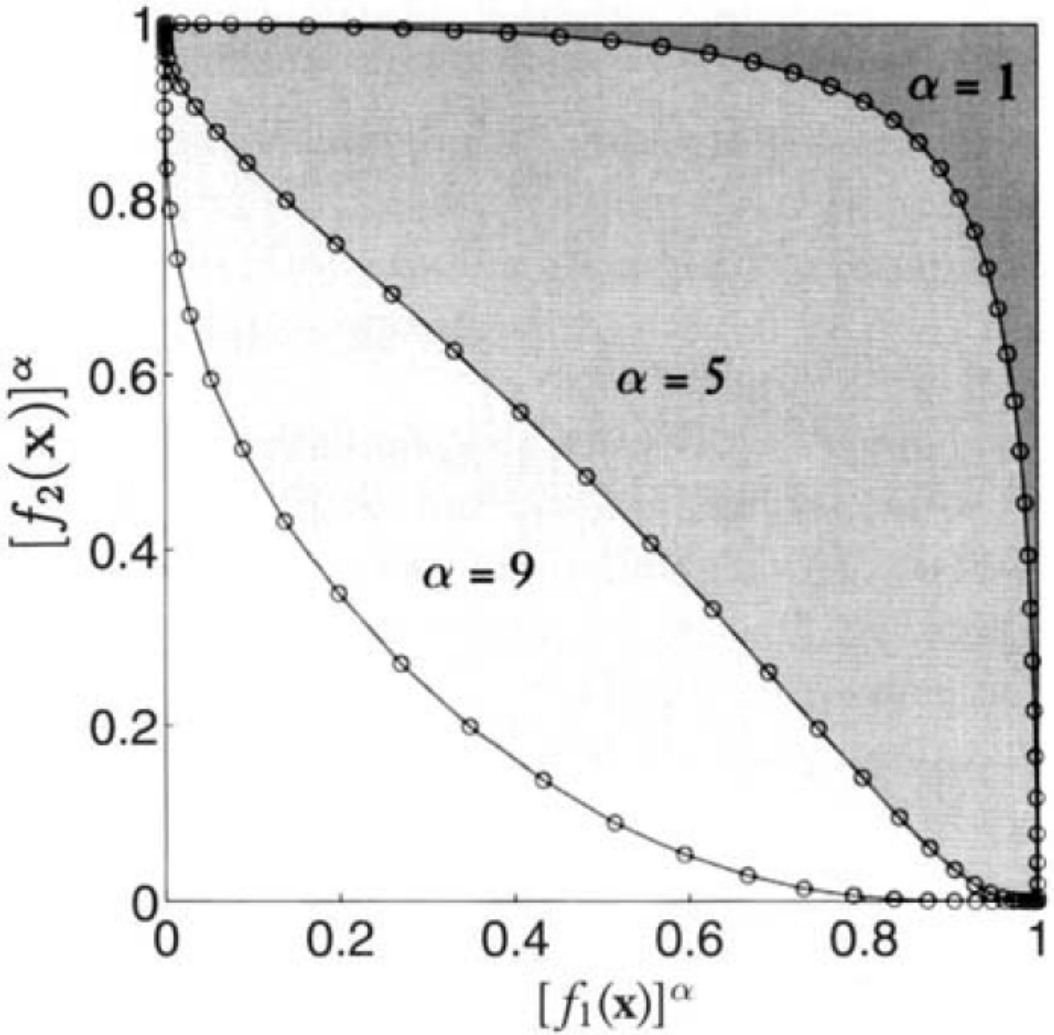
1. Introduction: History
2. The genetic code
3. The canonical genetic algorithm
4. Examples & Variants of GA
5. The schema theorem
6. Hybrid algorithms
7. Evolutionary robotics
8. Genetic Programming
9. Genetic Programming II
10. Recent and Practical issues



Overview

- Multiobjective Optimization
- Fitness-distance correlations
- No-free-lunch theorem for GA
- Practical issues

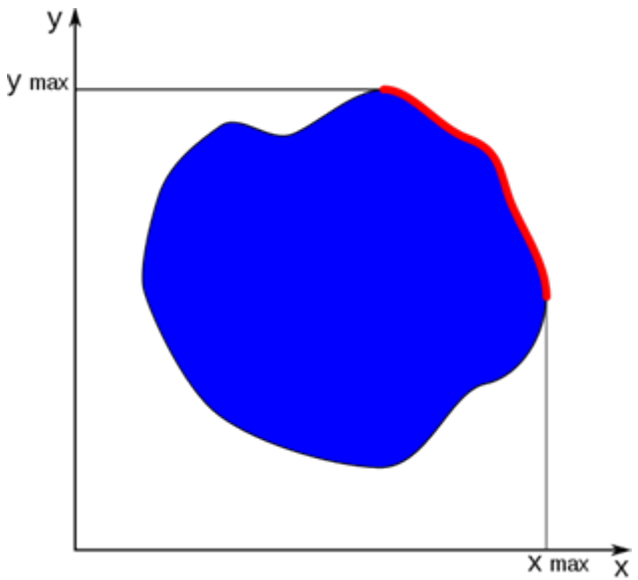
GA for Multiobjective Optimization: Combination of fitness functions



C. M. Fonseca &P. J. Fleming (1995) An Overview of Evolutionary Algorithms in Multiobjective Optimization

Evolutionary Computation 3(1): 1–16

Multiobjective optimization

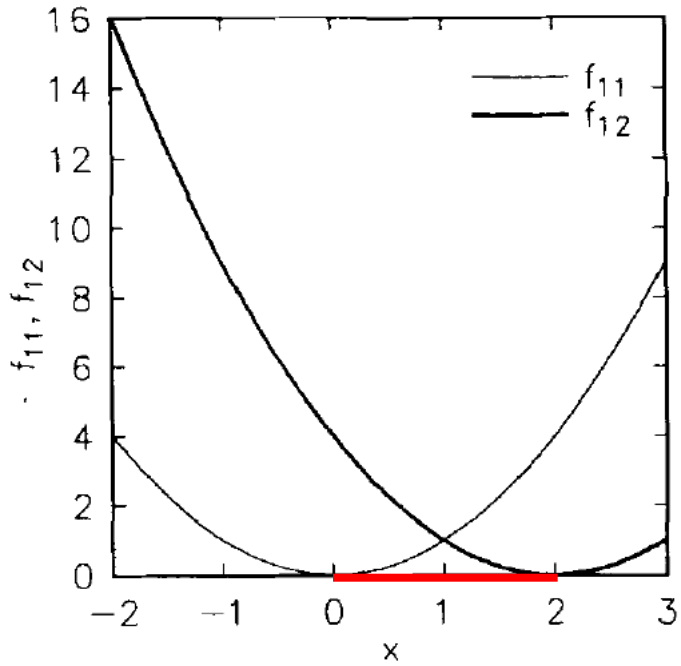


x^* is Pareto optimal for a class of fitness functions f_i if there exists no x with

$$f_i(x) \geq f_i(x^*) \qquad \text{for all } i$$

or, equivalently, x^* is not dominated by any other x.

Example: A machine is characterized by power and torque. A machine is *better* if -- at equal torque -- its power is higher. Unless a comparison between the two quanitities is possible, all machines along the red margin are (Pareto) optimal



GA for Multiobjective optimization

Benefits

- Collective search required for sampling the Pareto set
- Non-connected Pareto sets are OK
- Incorporation of constraints in fitness function

Problems:

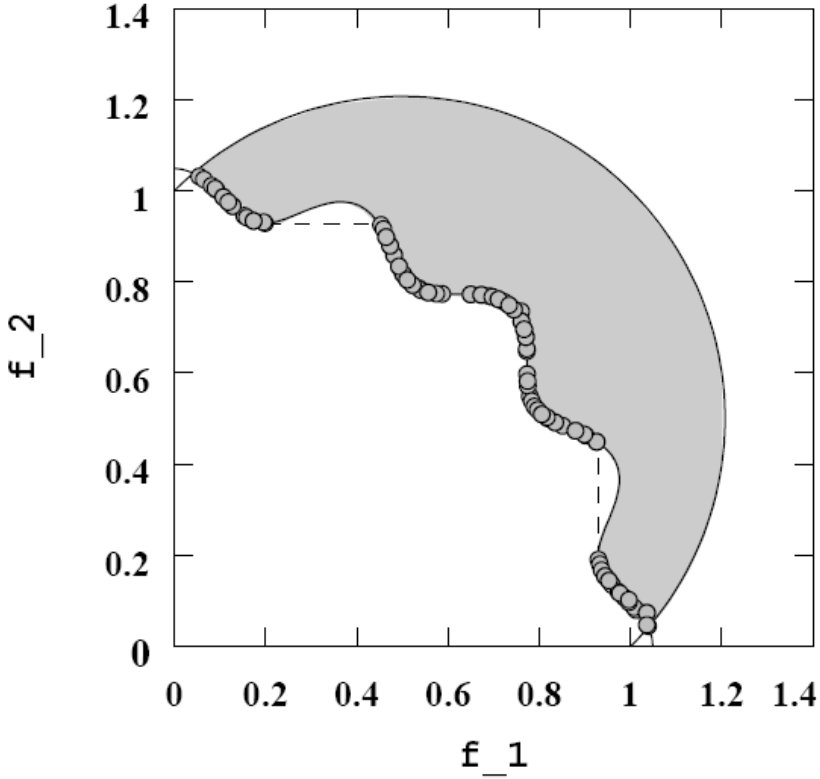
- Selection of fit individuals?
- Elitism?
- Pareto-optimal diversity?
- Speed?

$$f_1(\mathbf{x}) = x_1$$

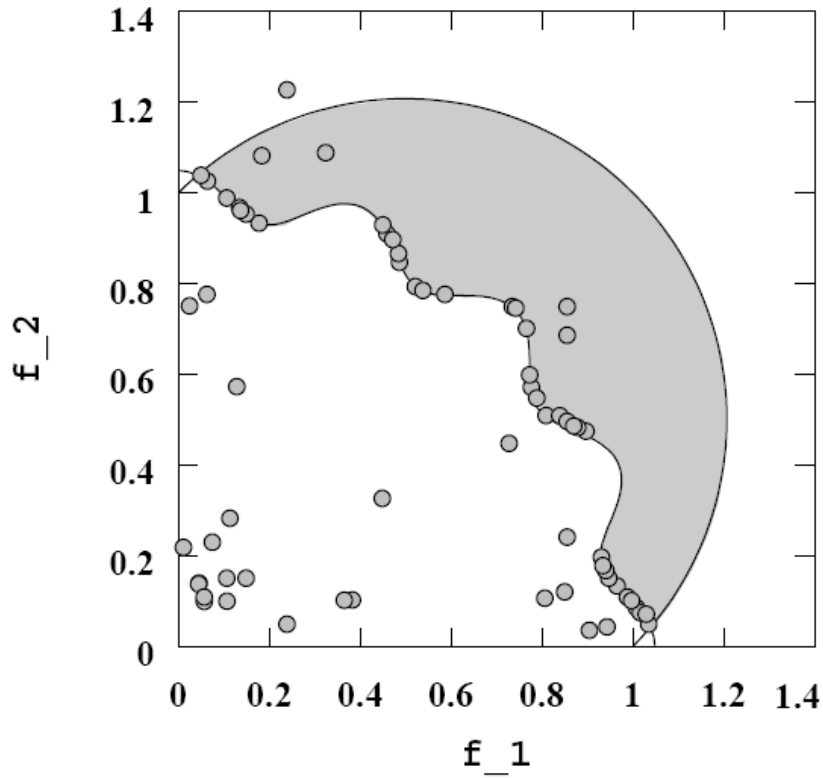
$$f_2(\mathbf{x}) = x_2$$

$$g_1(\mathbf{x}) = -x_1^2 - x_2^2 + 1 + 0.1\cos(16\arctan x/y) \leq 0$$

$$g_2(\mathbf{x}) = (x - 0.5)^2 + (x_2 - 0.5)^2 \leq 0.5$$



NSGA-II



conventional algorithm
(also GA-style)

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal and T. Meyarivan (2000) A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II, IEEE Transact. Evolutionary Computation 6,182-197.

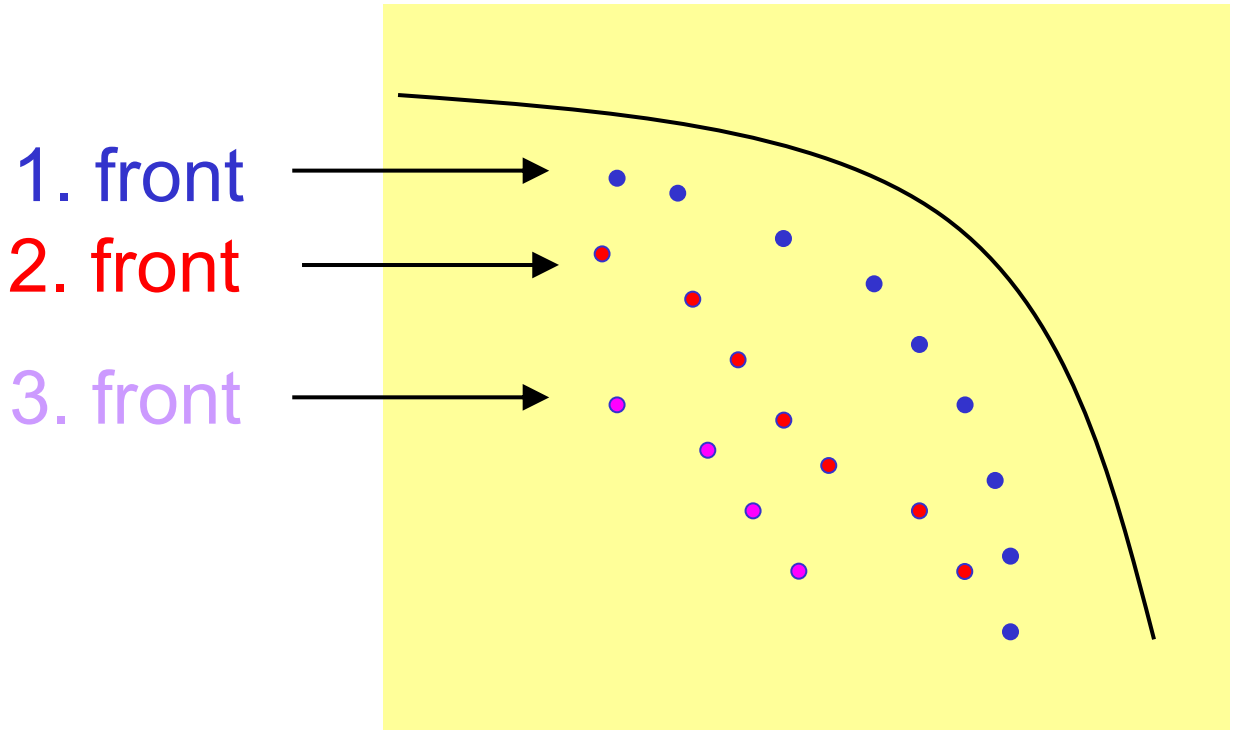
How does it work?

- Non-dominated sorting genetic algorithm (NSGA)
- Selection by non-dominated sorting
- Preserving diversity along the non-dominated front
- Use two populations *P* and *P'*
- $\prec$ "being dominated by", denotes a partial order induced by a set of fitness functions

$P' =$ `find-nondominated-front(`P`)`	
$P' = \{1\}$	include first member in P'
for each $p \in P \wedge p \notin P'$	take one solution at a time
$P' = P' \cup \{p\}$	include p in P' temporarily
for each $q \in P' \wedge q \neq p$	compare p with other members of P'
if $p \prec q$, then $P' = P' \backslash \{q\}$	if p dominates a member of P', delete it
else if $q \prec p$, then $P' = P' \backslash \{p\}$	if p is dominated by other members of P', do not include p in P'

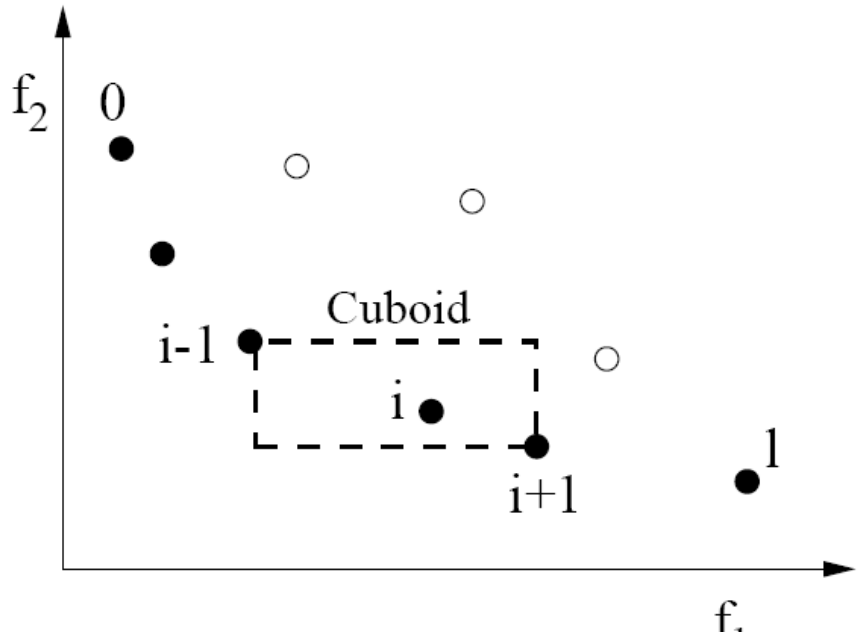
Complexity per step: $O(MN^2)$

“Ranking”



$\mathcal{F} = \texttt{fast-non-dominated-sort}(P)$	$\mathcal{F}$ is a set of non-dominated fronts
$i = 1$	i is the front counter and is initialized to one
until $P \neq \emptyset$	
$\mathcal{F}_i = \texttt{find-nondominated-front}(P)$	find the non-dominated front
$P = P \backslash \mathcal{F}_i$	remove non-dominated solutions from P
$i = i + 1$	increment the front counter

Preserving diversity




```
crowding-distance-assignment(I)
```

$l = |\mathcal{I}|$ — number of solutions in $\mathcal{I}$

for each i, set $\mathcal{I}[i]_{distance} = 0$ — initialize distance

for each objective m

  $\mathcal{I} = \text{sort}(\mathcal{I}, m)$ — sort using each objective value

  $\mathcal{I}[1]_{distance} = \mathcal{I}[l]_{distance} = \infty$ — so that boundary points are always selected

  for $i = 2$ to $(l - 1)$ — for all other points

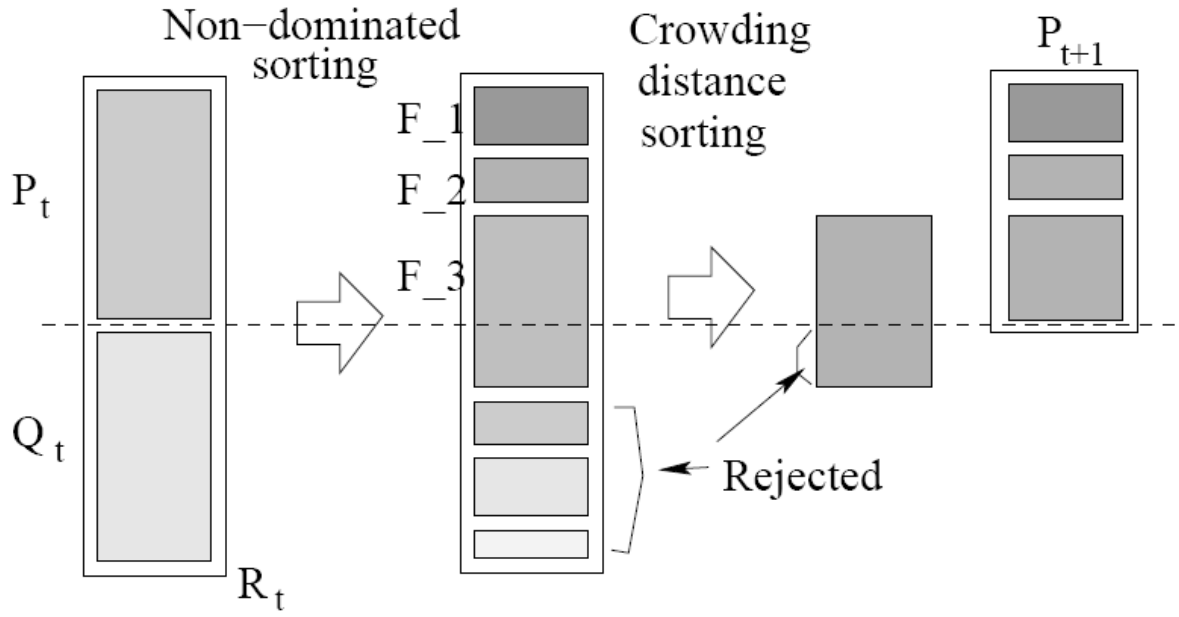
    $\mathcal{I}[i]_{distance} = \mathcal{I}[i]_{distance} + (\mathcal{I}[i+1].m - \mathcal{I}[i-1].m)$ — values of m-th fitness function

New distance measure: first rank, then lowest density:

$$i \prec_n j \quad \text{if } (i_{rank} < j_{rank}) \text{ or } ((i_{rank} = j_{rank}) \text{ and } (i_{distance} > j_{distance}))$$

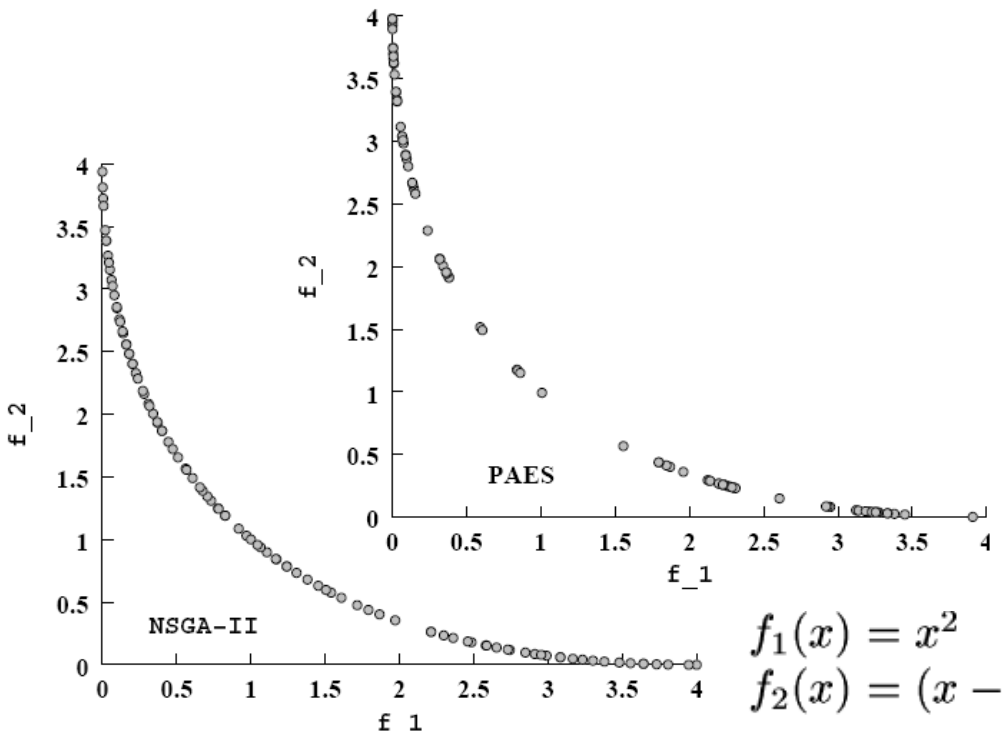
NSGA-II

main loop

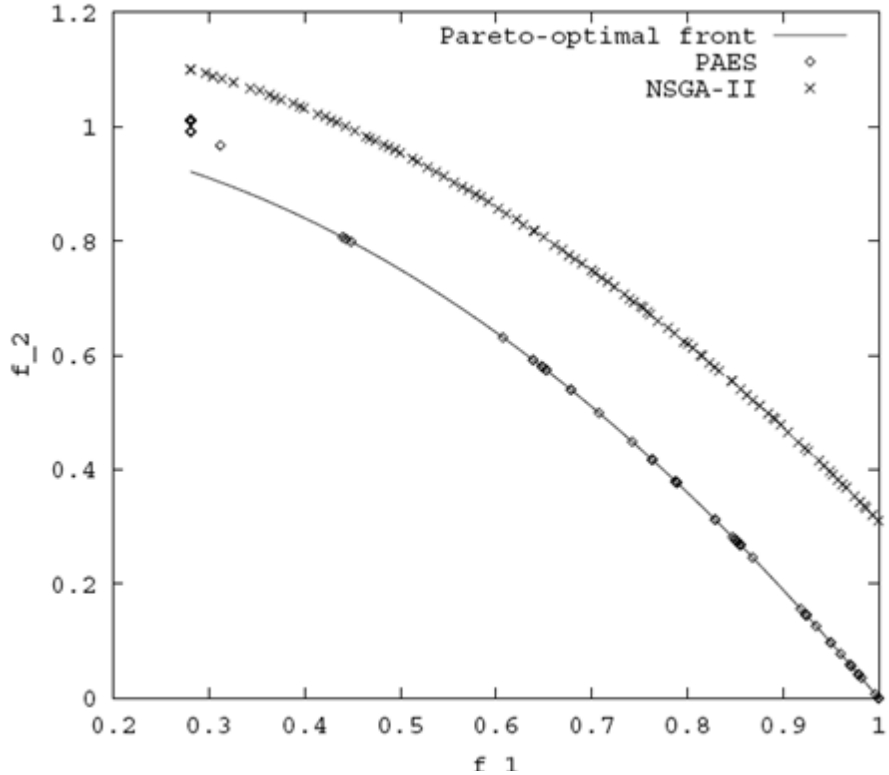


$R_t = P_t \cup Q_t$	combine parent and children population
$\mathcal{F} = \texttt{fast-nondominated-sort}(R_t)$	$\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2, \ldots)$, all non-dominated fronts of R_t
$P_{t+1} = \emptyset$ and $i = 1$	
until $\lvert P_{t+1}\rvert + \lvert\mathcal{F}_i\rvert \leq N$	till the parent population is filled
$\quad\texttt{crowding-distance-assignment}(\mathcal{F}_i)$	calculate crowding distance in $\mathcal{F}_i$
$\quad P_{t+1} = P_{t+1} \cup \mathcal{F}_i$	include i-th non-dominated front in the parent pop
$\quad i = i + 1$	check the next front for inclusion
$\text{Sort}(\mathcal{F}_i, \prec_n)$	sort in descending order using $\prec_n$
$P_{t+1} = P_{t+1} \cup \mathcal{F}_i[1 : (N - \lvert P_{t+1}\rvert)]$	choose the first $(N - \lvert P_{t+1}\rvert)$ elements of $calF_i$
$Q_{t+1} = \texttt{make-new-pop}(P_{t+1})$	use selection, crossover and mutation to create a new population Q_{t+1}
$t = t + 1$	increment the generation counter

Performance



$$f_1(x) = x^2$$
$$f_2(x) = (x-2)^2$$



$$f_1(\mathbf{x}) = x_1$$
$$f_2(\mathbf{x}) = g(\mathbf{x})\left[1 - \sqrt{x_1/g(\mathbf{x})}\right]$$
$$g(\mathbf{x}) = 1 + 10(n-1) + \sum_{i=2}^{n}\left[x_i^2 - 10\cos(4\pi x_i)\right]$$

left: Performance similar, NSGA-II has better distribution
right: Even spread of the solution is a further goal that here compromises Pareto optimality of NSGA-II. (optimality is towards down and left)

For comparison: (1 parent, 1 child) Pareto-archived Evolution Strategy (PAES by Knowles and Corne)

Fitness Distance Correlation

Assume you have a set of fitnesses $F = (f_1, ..., f_n)$ and a set of known distances to the global optimum $D = (d_1, ..., d_n)$

$$\text{FDC} = \frac{C}{SF \times SD}$$

SF and SD are the standard deviation of F and D respectively and

$$C = \frac{1}{n} \sum_{i=1}^{n} (f_i - \bar{f})(d_i - \bar{d})$$

where $\bar{f}$ and $\bar{d}$ are the means of F and D. C is the covariance of F and D.

Ideally, FDC = -1 , i.e. the smaller the distance the higher the fitness

Maximally **deceptive** fitness functions: FDC = 1.

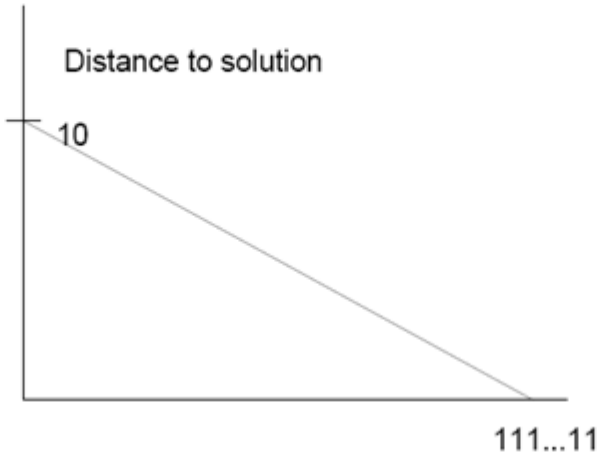
FDC Examples

Max-Ones: fitness is the number of ones in the chromosome. FDC = -1

Royal Road R1: 64-bit, reward groups of 8 contiguous ones. FDC $\sim$ -0.18

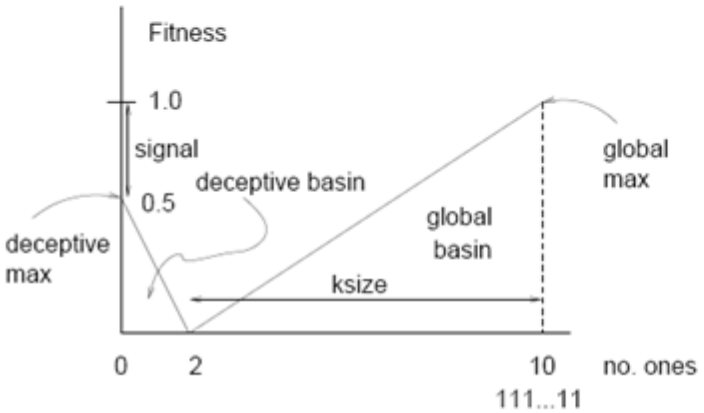
Royal Road R2: 64-bit, reward groups of 8, 16 and 32 contiguous ones. FDC $\sim$ -0.13

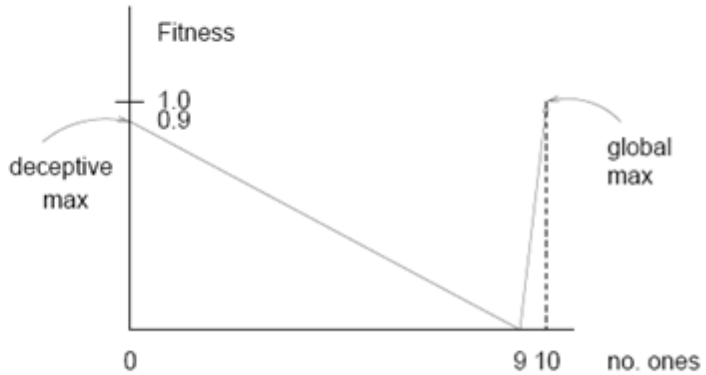
Gray codes vs. binary codes: for large numbers of bits, Gray coding usually has a better FDC



Deceptive Fitness Functions

Trap function FDC = -0.87





Highly deceptive trap function FDC = 0.73

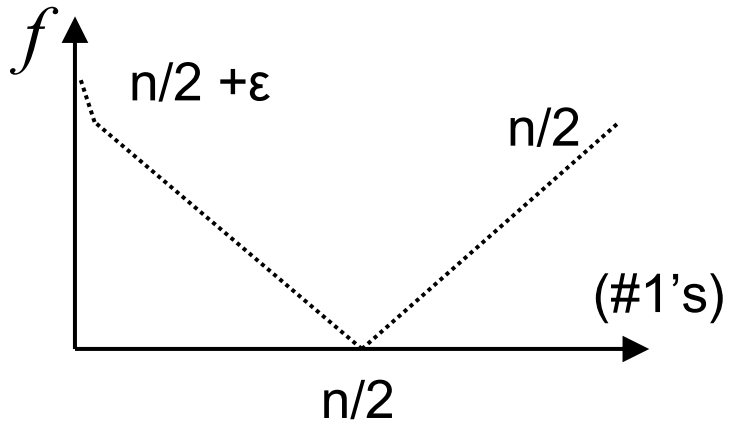
Deceptive fitness functions increase the **further** we get from the global optimum

It's the GA designer's job to design a fitness function that leads towards the global optimum – as best they can (we don't usually know where the global optimum is)

Deceptive fitness functions

- A sub-schema is more specific then a schema (i.e. as a subset it is smaller then the superset)
- A schema is deceptive if the sub-schema that contains the deceptive attractor is no worse than all other sub-schemata in the schema
- A fitness function is fully deceptive if all schemas are deceptive (e.g. $f(x)$=(#1's) if $x \neq 0$ and $f(0)=n+1$)
- FDC may not be a good indicator of deceptiveness for the following functions:

$f(x)=\max\{(\#1\text{'s}), (\#0\text{'s})\}+\varepsilon I(x=0)$



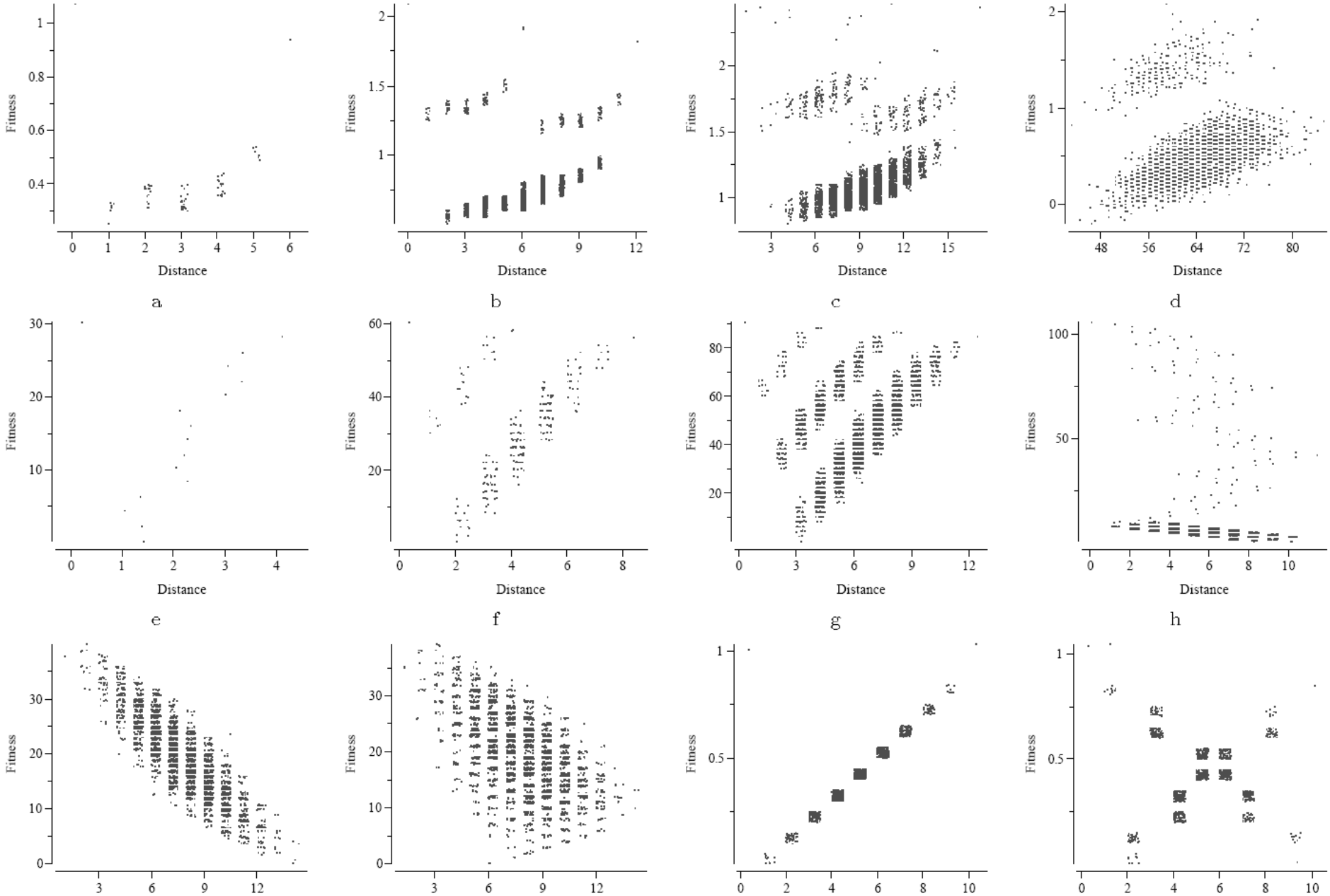


Figure 2: A sample of fitness distance scatter plots. Function sources are given in Table 1. FDC values for functions on more than 12 bits are computed from a random sample of 4000 points. The functions are as follows: (a-c) one, two and three copies of Deb & Goldberg's fully deceptive 6-bit problem ($r = 0.30$). Notice the additive effect. (d) Holland's royal road on 128 bits ($b = 8$, $k = 4$ and $g = 0$), ($r = 0.27$). (e-g) one, two and three copies of Whitley's F2, a fully deceptive 4-bit problem ($r = 0.51$). (h) Horn, Goldberg & Deb's long path problem with 11 bits ($r = -0.12$). Notice the path. (i,j) De Jong's F3 binary and Gray coded with 15 bits as a maximization problem ($r = -0.86$ and -0.57). (k) Liepins and Vose's fully deceptive problem on 10 bits ($r = 0.98$) and (l) their transformed problem ($r = -0.02$). Correlation cannot detect the X.

Fitness-Distance Correlation

Objectives:

- Visualization and prediction of difficulty of a problem based on the prediction of GA behaviour on a number of reasonably well-studied problems
- Are the predicted results reproducible?
- “Surprises” when using royal road fitness functions
- Effect of differences in coding and representation
- Connection between GAs and heuristic search

Previous slide: T Jones, S Forrest (1995) Fitness distance correlation as a measure of problem difficulty for genetic algorithms. Neural Information Processing Systems.

GA and GP

- Genetic Algorithms and Genetic Programming are related fields
- GAs use a fixed length linear representation
  GP uses a variable size tree representation
  (variable length means variable length up to some *fixed* limit)
- Representations and genetic operators of GA and GP appear different, (ultimately they are populations of bit strings in the computer's memory)
- The important difference lies in the interpretation of the representation: 1-to-1 mapping between the description of an object and the object itself (GA) or a many-to-1 mapping (GP)
- No Free Lunch theorem is valid for 1-to-1 mappings but not for many-to-1 mappings
- Many of the empirical results discovered in one field apply to the other field e.g. maintaining high diversity in a population improves performance

Woodward (2003)

No-Free-Lunch Theorems

- Statement:
  - Averaged over all problems
  - For any performance metric related to number of distinct data points
  - All non-revisiting black-box algorithms will display the same performance
- Implications
  - If a new black box algorithm is good for one problem → it is probably poor for another one
  - There are as many deceptive as easy fitness functions
  - Makes sense not to use “black-box algorithms”
- Ongoing work showing counterexamples given specific constraints

Wolpert & Macready (1997) No Free Lunch Theorems for Optimization *IEEE Transact Evolutionary Computation* **1**, 67

Pragmatics of GA Design

- Selection methods
- Crossover
- Mutation
- Population model and elitism
- Spatial separation
- Maintaining diversity
- Parameter setting
- Evaluating the system – experimental methodology

Next (fifth) tutorials: Pragmatics of GA design
Work yourself through slides 21-44. At the tutorials some exam-style questions will be available for you to be answered in class.

Selection Method Considerations

What do we want selection to do?

- Provide selection pressure – AND avoid premature convergence, maintain diversity, exploration vs. exploitation
  How would we detect premature convergence?

A measure:

- Takeover time – till best individual replaces all others
  Is the best individual good enough? (If yes, convergence wasn't premature!)

Do we want the optimal solution, a good solution, any solution, a solution within some specific time, a good (on average) set of solutions?

Selection Methods

Aim: choose parents. Emphasise fitter ones. Balance exploitation and exploration.

- Fitness-proportionate selection (FPS)
  Often leads to premature convergence; has high selection pressure
- Rank-based selection
  Fitness(i) = Min + (Max - Min)(Rank(i) - 1)/(N-1) then do FPS
  Max and Min are chosen by you.
  This is linear scaling, but can do exponential scaling or any other scaling that...
  ...Preserves diversity, slows selection pressure
- Tournament selection
  Select k individuals. (Copies of) Fittest m go into intermediate population (perhaps with some probability)
  Less computationally expensive (don't evaluate all chroms.)

- Uniform selection (FUSS – Fitness Uniform Selection Scheme)

  Lowest/highest fitness in current generation is Min, Max. Select a fitness f uniformly in [Min, Max]. Individual with fitness closest to f is chosen. Maintains genetic diversity – we only want **one** solution of maximal fitness. Favours those solutions that are different from all others and encourages search

- Elitism

  Copy some number of **fittest** individuals into intermediate or next-generation population

  Don't lose good solutions when we've found them until we find better solutions

- and others, e.g. combinations of the above. See Mitchell Sect. 5.4.

- Fitness sharing

  Fitness + raw fitness / some measure of how many others are similar

  Reward difference. Get speciation, can explore several local maxima, avoid premature convergence

- May wish to change the selection method or rate parameters as a function of generation or diversity or fitness: balancing exploration at start of process with exploitation later on, or to avoid premature convergence, or to provide killer instinct

Crossover

- Single-point, Two-point (as seen previously)
- Try to preserve building blocks (but avoid hitch-hiking): cf. Montana-Davis
- Uniform: choose each child gene with probability p from parent 1, else 2 – so no linkage between genes. Often p $= 0.5$ or a bit higher.

Attempts to make crossover less disruptive (or at least mitigate disruptive effects):

– Brood crossover: 2 parents produce several offspring, fittest 2 chosen

– Elite crossover: put offspring into pool with parents, select fittest 2

– Intelligent crossover: crossover hotspots – a template for crossover points that is also evolved

Mutation

- Original aim: to preserve diversity
- Can end up solving the problem
- Allele (point) mutation probability of mutation $p_m \sim 1/l$ where l is the chromosome length (ca. 1 mutation per chromosome)
- Reordering mutations – inversion or cycling – like natural evolution
- Swap mutations – swap 2 random positions
- Intelligent mutations
  - encode p_m onto chromosome and allow GA to evolve it
  - use a schedule to change p_m as a function of time
  - or as a function of fitness
- Choose to suit the problem – chromosome length, population size

Population Model and Elitism

- What do we do with the new candidates?
- **Generational** model: place children from generation n into generation $n+1$, continue until generation $n+1$ is full
- Generational model with **elitism**: keep some proportion of the best candidates from generation n and give them a free ride into generation $n+1$
- **Steady state** model: newly created children are inserted into the $current$ generation and replace the worst candidates

Elitism and steady state can lead to very high selection pressure (why?)

Spatial Separation

- **Island** model: evolve independent populations, sometimes allowing a good candidate to migrate across islands
- Simple island model:
  - P independent populations
  - every M generations select one (or best) candidate from one population
  - insert it into all other populations (replace the worst)
- Advantages: encourages diversity, can run in parallel, find alternative solutions
- Can use different fitness functions on the islands – multicriterion optimisation
- **Cellular** model: candidates are arranged on a Grid and can only mate with nearby candidates; tune selection pressure via grid shape (square grid, short takeover time, high selection pressure; narrow grid, long takeover time, low selection pressure)

Maintaining Diversity

- Restrict which individuals can mate with each other, e.g. must be sufficiently different.
- New offspring replace those most similar to themselves
- Fitness sharing (see earlier slide)
- Mate selection – keep a label on each chromosome. It can only mate with chromosomes with same label. Evolve labels
- Island and cellular models
- High p_m

Setting Parameters

- Usual approach:
  - Try a wide range of parameters (logarithmic? linear search?)
  - Using the best, alter one at a time
  - Continue until no improvement possible
- but the parameters interact non-linearly
- Systematic approach: use a lot of CPU time
- Self-adaptation? Fitness of parameters based on how many highly fit individuals they contribute to producing (so we need to run a GA to get the fitness of a GA)
- Starting point: Small population 20–50, crossover rate 0.75–0.95, mutation rate 0.005–0.01 per bit ($0.01 \times 50 \simeq$ half a mutation per generation)

Evaluating the System

- No. generations vs. no. evaluations
- Cpu time vs. real time (fitness function takes most time)
- Benchmarks: a range of real problems
- Compare with alternative algorithms
- Time to reach solution
- Quality of solution

- *Real* fitness of solution (to your actual purpose) – did the GA exploit the fitness function or did it do what you wanted it to do? E.g. spotting tanks (all the pictures of the tanks were taken in bright sunlight – detecting overall intensity level was sufficient – the GA did not recognise the tank, only the sunshine)

- Acceptability to users

- When is one parameter set better than another? Statistics, many experiments with each parameter set (see Lecture notes from previous years Appendix A for notes on statistics and data analysis)

What do Genetic Algorithms Offer?

- Robust problem-solving ability
- Search, optimisation, machine learning
- Good performance on dynamic as well as static problems (e.g. job-shop scheduling)
- Ease of implementation
- Hybridisation with other methods (e.g. genetic planning, as optimiser for initial hill-climbing solutions)
- Anytime problem solving behaviour
- Easy to run on parallel machines
- A competitive solution for many hard problems

Reading: Mitchell Chapter 5 – all of it
Section 10 of Whitley tutorial (notes Chapter 2)

Representation: Encoding the candidates

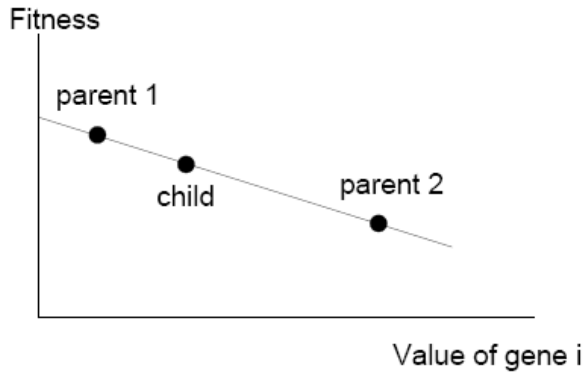
How shall we represent it?

- Fixed-length linear binary encodings
  Unnatural. Unnatural orderings. Hamming cliff. Gray codes?
  Theory exists
- Fixed-length linear non-binary encodings
  Real values or characters. NN weights or exam timetables
- Variable length linear non-binary encodings
  Plans, Prisoner's Dilemma
- Tree-based chromosomes
  GP. Open-ended search space. But unwieldy trees, much junk
- Implicit, explicit
  Chromosome of parameters vs. grammar

Intuition: encode solution in the most natural way possible, then create genetic operators to make it work.

Operators for Non-binary Encodings: Crossover

- We can use single- or multi-point crossover on all linear encodings.
- If we use binary representation of non-binary numbers, need to decide whether to do crossover at the points *between* the *numbers* or between any of the *bits*. (The latter will not maintain "number-sized" building blocks.)
- Montana-Davis crossover for neural nets – maintain semantics
- "Semantics-based" crossover for plans – crossover inbetween plan actions (see also evolving policies – subsequent lecture)
- Subtree crossover for tree-based chromosomes
- (also use) Averaging crossover for real-valued chromosomes: value of gene i in child is (first-parent $+ \alpha \cdot$ second-parent) – will tend to interpolate fitness between parents



(What if fitness surface varies hugely between parent1 and parent2?)

- Uniform crossover: choose each child gene with probability p from parent1, $(1-p)$ from parent2
- Brood crossover, elite crossover: aim to mitigate disruptive effects of crossover
- Intelligent crossover: evolve crossover point(s) on chromosome

Operators for Non-binary Encodings: Mutation

- **Binary**:point mutation (bit-flipping)
- **Non-binary, discrete**: Replace allele with another randomly chosen from the alphabet
- **Non-binary, real**: Add small random number, keep value within some desired range
- **Plans**: add in a random action, mutate the parameter(s) to some action, delete an action
- **Trees**: add a random subtree, promote a subtree (hoist), remove/shrink a subtree (replace by a terminal)
- Reordering, swap mutation, encode mutation parameters onto chromosome

Evaluating the Candidates

- Need
  - A set of configurations C – the chromosomes
  - A fitness function $f : C \rightarrow \Re$
  - An additional geometrical/topological/algebraic structure N on C that allows us to define which chromosomes are neighbours – i.e. what says that chromosome A should be arrayed next to chromosome B on our picture of the fitness landscape? How similar are two chromosomes? (Stadler: landscape theory)
- Single candidate fitness function, $f(c_i)$
  - The more fine-grained, the better
  - Should push towards better solutions
- Fitness function, neighbourhood structure, operators all interact

Searching the Space 1

Crossover and mutation cause the GA to search the fitness landscape.

Must maintain genetic diversity in order to escape from local optima

Fitness Uniform Selection Scheme – FUSS (M. Hutter)

Suppose the lowest and highest fitness values in the population are f_{min} and f_{max}. Select a fitness value f uniformly in $[f_{min}, f_{max}]$. The individual whose fitness is closest to f is chosen for intermediate population (then crossover and mutation).

Uses a distribution over fitness values rather than individual chromosomes

So if there's a high density of unfit individuals and a low density of fit individuals, we'll favour the fitter ones. (We'll also favour low fitness individuals if there are only a few of them $\Rightarrow$ extra search)

Searching the Space 2

Fitness Uniform Deletion Scheme – FUDS (Hutter/Legg IDSIA)

Use with a steady-state GA

When we need to delete an individual, only delete one with a very common fitness value (divide fitness range into bins; delete an individual from the bin with the most individuals)

Does better than random deletion on some common and deceptive problems

Searching the Space 3

Scouting-Inspired Evolutionary Algorithm – SEA (Pfaffmann et al.)

Change the mutation amount according to how "surprising" a chromosome's fitness is (amount to be added to real-valued gene given by Gaussian distribution with width σ)

- Keep a database of chromosomes and their fitness values
- Estimate the fitness of a new chromosome from database values (e.g. use nearest neighbours)
- Surprise = actual measured fitness - estimated fitness
- If surprise is large, make σ small; we are in part of search space not seen before, so explore some more
- If surprise is small, make σ large; we are in part of search space that is well-modelled, so try to jump elsewhere

Searching the Space 4

- We might use fitness sharing for multiple solutions – prevent premature convergence
  - Fitness = Raw fitness/(Some measure of how many others are similar)
  - Reward difference. Speciation. Explore several local maxima

Searching in Phenotype Fitness Space

- Round Robin competitions for strategies
- Decode genotype into phenotype and evaluate its behaviour: controllers for agents, neural networks

When should a GA be used?

- Large or very large search space
  Noughts and crosses vs. protein folding
- A sufficiently good solution is good enough
  Exam timetabling
- Fitness landscape is not smooth and unimodal
  Optimal headphone loudness vs. setting value on a mixing desk
- Fitness landscape is poorly understood
  Find Flatiron building in Manhattan vs. Paris Left Bank bistro
- Fitness function is noisy and/or complex
  Sensory input or performance in noisy/unpredictable world

- No good algorithms exist to solve the problem
  Timetabling?
- Good local search operators exist
  Building a plan
- The problem is weakly compositional
  TSP vs. Lottery Extra

♠ Linux kernel tuning using a GA (Moilanen): chromosome is string of Linux kernel internal settings, fitness function is performance under some workload (benchmark workloads)

♠ TSP, knapsack, bin-packing, design of concert-hall acoustics, (simulated) F1 cars

Next (fifth) tutorials: Pragmatics of GA design
Work yourself through slides 21-44. At the tutorials some exam-style questions will be available for you to be answered in class.