EC Preliminaries

- Dr Karsten Øster Lundqvist
  - k.o.lundqvist@reading.ac.uk
  - Otherwise often found in SSE Room 178

Preliminaries

- Assessment
  - 1.5 hr examination (50%)
  - Coursework (50%), can be found on BB, and in your hands
- Course Text
  - Introduction to Evolutionary Computing, A.E. Eiben, J.E. Smith, ISBN 3-540-40184-9, Springer-Verlag, 2003
  - Website

  www.cs.vu.nl/~gusz/ecbook/ecbook.html

  www.cems.uwe.ac.uk/~jsmith/ecbook/ecbook.html

Preliminaries

- This used to be a self-study module with only few lectures. This has changed, but self-study is still expected!
- So you must support by additional reading!
  - You can find much inspiration on Blackboard…

Preliminaries

- Suggestions:
  - Clever Algorithms: Nature-Inspired Programming Recipes, Jason Brownlee (Free ebook on lulu)
  - Handbook of Evolutionary Computation, Thomas Bäck et al. Oxford University Press. £553.00 on Amazon (yesterday), but available in library.
  - The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation, Gary W. Flake (£25.70 on Amazon)
  - Evolutionary Computation: A Unified Approach. Kenneth A De Jong (£35.10)

Preliminaries

- More Books
- T Back, Evolutionary Algorithms in Theory and Practice, Oxford University Press, New York, 1996.
- T Back, D.B. Fogel, Z. Micahelwicz, eds, Evolutionary Computation 1: Basic Algorithms and Operators, Institute of Physics Publishing 2000.
- T Back, D.B. Fogel, and Z. Michalewicz, eds, Evolutionary Computation 2:Advanced Algorithms and Operators. Institute of Physics Publishing 2000
- Hans-Georg Beyer, The theory of Evolution Strategies, Springer, Berlin, 2001
- W.B. Langdon, Genetic programming + Data Structures = Automatic programming, Kluwer, 1998

Preliminaries

- And Even More…
- D.E. Goldberg, Genetic Algorithms in search, optimisation and machine Learning, Addison-Wesley, 1989.
- D.E.Goldberg, The design of innovation: lessons from and for Competent Genetic Algorithms (Genetic Algorithms and Evolutionary Computation) Kluwer Academic Press, 2002
- J.H. Holland, Adaptation Natural and Artificial Systems, MIT Press, Cambridge, MA, 1992.
- J.R. Koza, Genetic programming, MIT Press, Cambridge, MA 1992.
- J.R. Koza, Genetic programming II, as above 1994.

Introduction

Chapter 1

Let's have some fun!

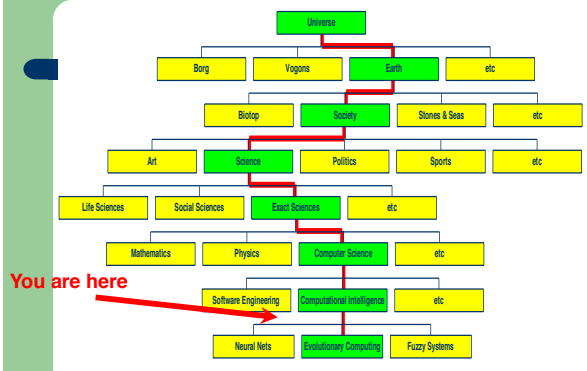
- Car creation in a 2d space?

Let's have some fun!

- The Genetic Car Builder
- http://boxcar2d.com/

Contents

- Positioning of EC and the basic EC metaphor
- Historical perspective
- Biological inspiration:
  - Darwinian evolution theory (simplified!)
  - Genetics (simplified!)
- Motivation for EC
- What can EC do: examples of application areas
- Demo: evolutionary magic square solver



Positioning of EC

- EC is part of computer science
- EC is not part of life sciences/biology
- Biology delivered inspiration and terminology
- EC can be applied in biological research

The Main Evolutionary Computing Metaphor

EVOLUTION		PROBLEM SOLVING
Environment	⟷	Problem
Individual	⟷	Candidate Solution
Fitness	⟷	Quality

Stochastic trial-and-error

Fitness → chances for survival and reproduction
Quality → chance for seeding new solutions

Brief History 1: the ancestors

- 1948, Turing:
  proposes "genetical or evolutionary search"
- 1962, Bremermann
  optimization through evolution and recombination
- 1964, Rechenberg
  introduces evolution strategies
- 1965, L. Fogel, Owens and Walsh
  introduce evolutionary programming
- 1975, Holland
  introduces genetic algorithms
- 1992, Koza
  introduces genetic programming

Brief History 2: The rise of EC

- 1985: first international conference (ICGA)
- 1990: first international conference in Europe (PPSN)
- 1993: first scientific EC journal (MIT Press)
- 1997: launch of European EC Research Network EvoNet

EC in the early 21st Century

- 3 major EC conferences, about 10 small related ones
- 3 scientific core EC journals
- 750-1000 papers published in 2003 (estimate)
- EvoNet has over 150 member institutes
- uncountable (meaning: many) applications
- uncountable (meaning: ?) consultancy and R&D firms

Darwinian Evolution 1: Survival of the fittest

- All environments have finite resources
  (i.e., can only support a limited number of individuals)
- Lifeforms have basic instinct/ lifecycles geared towards reproduction
- Therefore some kind of selection is inevitable
- Those individuals that compete for the resources most effectively have increased chance of reproduction
- Note: fitness in natural evolution is a derived, secondary measure, i.e., we (humans) assign a high fitness to individuals with many offspring

Darwinian Evolution 2: Diversity drives change

- Phenotypic traits:
  - Behaviour / physical differences that affect response to environment
  - Partly determined by inheritance, partly by factors during development
  - Unique to each individual, partly as a result of random changes
- If phenotypic traits:
  - Lead to higher chances of reproduction
  - Can be inherited

  then they will tend to increase in subsequent generations,
- leading to new combinations of traits …

Darwinian Evolution:Summary

- Population consists of diverse set of individuals
- Combinations of traits that are better adapted tend to increase representation in population

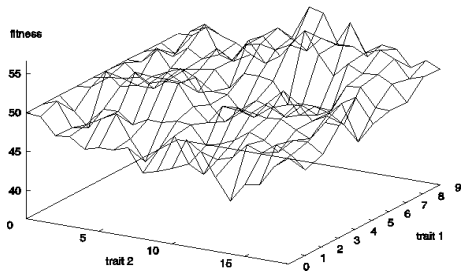
  Individuals are “units of selection”
- Variations occur through random changes yielding constant source of diversity, coupled with selection means that:

  Population is the “unit of evolution”
- Note the absence of “guiding force”

Adaptive landscape metaphor (Wright, 1932)

- Can envisage population with *n* traits as existing in a *n+1*-dimensional space (landscape) with height corresponding to fitness
- Each different individual (phenotype) represents a single point on the landscape
- Population is therefore a “cloud” of points, moving on the landscape over time as it evolves - adaptation

Example with two traits



Adaptive landscape metaphor (cont'd)

•Selection "pushes" population up the landscape

•Genetic drift:
- random variations in feature distribution (+ or -) arising from sampling error
- can cause the population "melt down" hills, thus crossing valleys and leaving local optima

Natural Genetics

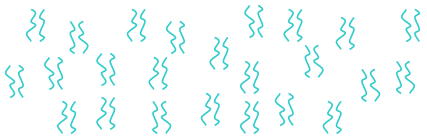
- The information required to build a living organism is coded in the DNA of that organism
- Genotype (DNA inside) determines phenotype
- Genes → phenotypic traits is a complex mapping
  - One gene may affect many traits (pleiotropy)
  - Many genes may affect one trait (polygeny)
- Small changes in the genotype lead to small changes in the organism (e.g., height, hair colour)

Genes and the Genome

- Genes are encoded in strands of DNA called chromosomes
- In most cells, there are two copies of each chromosome (diploidy)
- The complete genetic material in an individual's genotype is called the Genome
- Within a species, most of the genetic material is the same

Example: Homo Sapiens

- Human DNA is organised into chromosomes
- Human body cells contains 23 pairs of chromosomes which together define the physical attributes of the individual:

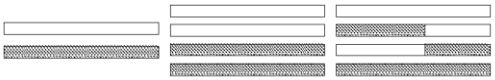


Reproductive Cells

- Gametes (sperm and egg cells) contain 23 individual chromosomes rather than 23 pairs
- Cells with only one copy of each chromosome are called Haploid
- Gametes are formed by a special form of cell splitting called meiosis
- During meiosis the pairs of chromosome undergo an operation called *crossing-over*

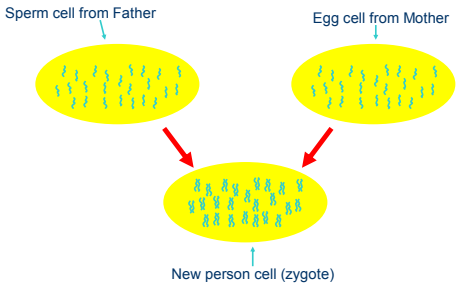
Crossing-over during meiosis

- Chromosome pairs align and duplicate
- Inner pairs link at a *centromere* and swap parts of themselves



- Outcome is one copy of maternal/paternal chromosome plus two entirely new combinations
- After crossing-over one of each pair goes into each gamete

Fertilisation



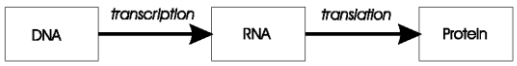
After fertilisation

- New zygote rapidly divides etc creating many cells all with the same genetic contents
- Although all cells contain the same genes, depending on, for example where they are in the organism, they will behave differently
- This process of differential behaviour during development is called ontogenesis
- All of this uses, and is controlled by, the same mechanism for decoding the genes in DNA

Genetic code

- All proteins in life on earth are composed of sequences built from 20 different amino acids
- DNA is built from four nucleotides in a double helix spiral: purines A,G; pyrimidines T,C
- Triplets of these form *codons*, each of which codes for a specific amino acid
- Much redundancy:
  - purines complement pyrimidines
  - the DNA contains much rubbish
  - 4^3=64 codons code for 20 amino acids
  - genetic code = the mapping from codons to amino acids
- **For all natural life on earth, the genetic code is the same !**

Transcription, translation



A central claim in molecular genetics: only one way flow

Genotype → Phenotype

Genotype ↛ Phenotype

Lamarckism (saying that acquired features can be inherited) is thus wrong!

Mutation

- Occasionally some of the genetic material changes very slightly during this process (replication error)
- This means that the child might have genetic material information not inherited from either parent
- This can be
  - catastrophic: offspring in not viable (most likely)
  - neutral: new feature not influences fitness
  - advantageous: strong new feature occurs
- Redundancy in the genetic code forms a good way of error checking

Motivations for EC: 1

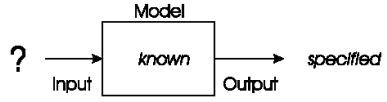
- Nature has always served as a source of inspiration for engineers and scientists
- The best problem solver known in nature is:
  - the (human) brain that created "the wheel, New York, wars and so on" (after Douglas Adams' Hitch-Hikers Guide)
  - the evolution mechanism that created the human brain (after Darwin's Origin of Species)
- Answer 1 → neurocomputing
- Answer 2 → evolutionary computing

Motivations for EC: 2

- Developing, analyzing, applying problem solving methods a.k.a. algorithms is a central theme in mathematics and computer science
- Time for thorough problem analysis decreases
- Complexity of problems to be solved increases
- Consequence:
  Robust problem solving technology needed

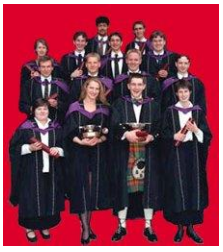
Problem type 1 : Optimisation

- We have a model of our system and seek inputs that give us a specified goal



- e.g.
  - time tables for university, call center, or hospital
  - design specifications, etc etc

Optimisation example 1: University timetabling



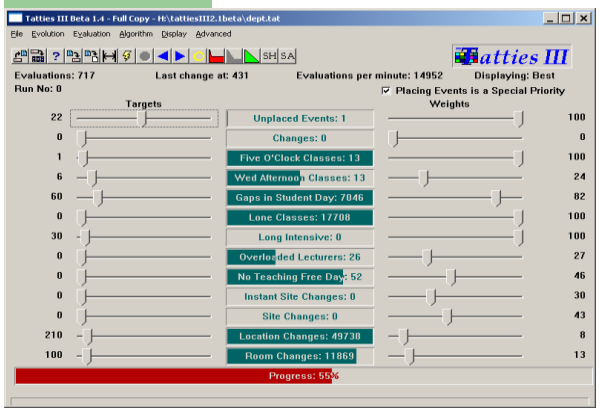
Enormously big search space

Timetables must be *good*

"Good" is defined by a number of competing criteria

Timetables must be feasible

Vast majority of search space is infeasible



Optimisation example 2: Satellite structure

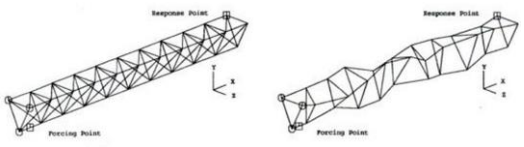
Optimised satellite designs for NASA to maximize vibration isolation

Evolving: design structures

Fitness: vibration resistance

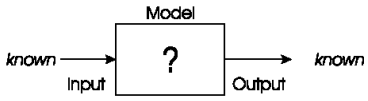
Evolutionary “creativity”

Result!



Problem types 2: Modelling

- We have corresponding sets of inputs & outputs and seek model that delivers correct output for every known input



- Evolutionary machine learning

Modelling example: loan applicant creditibility

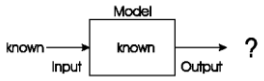
British bank evolved creditability model to predict loan paying behavior of new applicants

Evolving: prediction models

Fitness: model accuracy on historical data

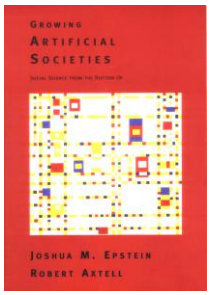
Problem type 3: Simulation

- We have a given model and wish to know the outputs that arise under different input conditions



- Often used to answer "what-if" questions in evolving dynamic environments
- e.g. Evolutionary economics, Artificial Life

Simulation example: evolving artificial societies



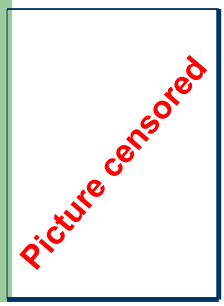
Simulating trade, economic competition, etc. to calibrate models

Use models to optimise strategies and policies

Evolutionary economy

Survival of the fittest is universal (big/small fish)

Simulation example 2: biological interpetations



Incest prevention keeps evolution from rapid degeneration
(we knew this)

Multi-parent reproduction, makes evolution more efficient
(this does not exist on Earth in carbon)

2nd sample of Life

Demonstration: magic square

- Given a 10x10 grid with a small 3x3 square in it
- Problem: arrange the numbers 1-100 on the grid such that
  - all horizontal, vertical, diagonal sums are equal (505)
  - a small 3x3 square forms a solution for 1-9

Demonstration: magic square

Evolutionary approach to solving this puzzle:

- Creating random begin arrangement
- Making N mutants of given arrangement
- Keeping the mutant (child) with the least error
- Stopping when error is zero

Demonstration: magic square

- Software by M. Herdy, TU Berlin
- Interesting parameters:
  - Step1: small mutation, slow & hits the optimum
  - Step10: large mutation, fast & misses (“jumps over” optimum)
  - Mstep: mutation step size modified on-line, fast & hits optimum
- Start: double-click on icon below
- Exit: click on TUBerlin logo (top-right)

Application

What is an Evolutionary Algorithm?

Chapter 2

Contents

- Traditional EA
- Recap of Evolutionary Metaphor
- Basic scheme of an EA
- Basic Components:
  - Representation / Evaluation / Population / Parent Selection / Recombination / Mutation / Survivor Selection / Termination
- Examples : eight queens / knapsack
- Typical behaviours of EAs
- EC in context of global optimisation

Traditional EA

- Combined view <= Today
- 3 traditional schools of EA, which now has evolved into one (EC) and/or many!
  - Genetic Algorithms
    - Week 3
  - Evolutionary Strategies
    - **Self-study (Week 4)**
  - Evolutionary Programming
    - Week 3

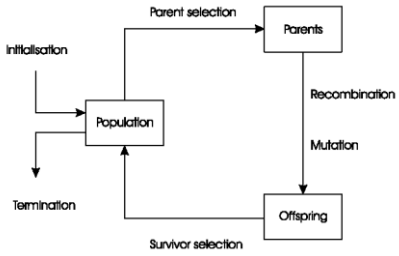
Recap of EC metaphor

- A population of individuals exists in an environment with limited resources
- ***Competition*** for those resources causes selection of those ***fitter*** individuals that are better adapted to the environment
- These individuals act as seeds for the generation of new individuals through recombination and mutation
- The new individuals have their fitness evaluated and compete (possibly also with parents) for survival.
- Over time ***Natural selection*** causes a rise in the fitness of the population

Recap 2:

- EAs fall into the category of “generate and test” algorithms
- They are stochastic, population-based algorithms
- Variation operators (recombination and mutation) create the necessary diversity and thereby facilitate novelty
- Selection reduces diversity and acts as a force pushing quality

General Scheme of EAs



Pseudo-code for typical EA

```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END
```


What are the different types of EAs

- Historically different flavours of EAs have been associated with different representations
  - Binary strings : Genetic Algorithms
  - Real-valued vectors : Evolution Strategies
  - Finite state Machines: Evolutionary Programming
  - LISP trees: Genetic Programming
- These differences are largely irrelevant, best strategy
  - choose representation to suit problem
  - choose variation operators to suit representation
- Selection operators only use fitness and so are independent of representation

Representations

- Candidate solutions (individuals) exist in *phenotype* space
- They are encoded in chromosomes, which exist in *genotype* space
  - Encoding : phenotype=> genotype (not necessarily one to one)
  - Decoding : genotype=> phenotype (must be one to one)
- Chromosomes contain genes, which are in (usually fixed) positions called loci (sing. locus) and have a value (allele)

In order to find the global optimum, every feasible solution must be represented in genotype space

Evaluation (Fitness) Function

- Represents the requirements that the population should adapt to
- a.k.a. *quality* function or *objective* function
- Assigns a single real-valued fitness to each phenotype which forms the basis for selection
  - So the more discrimination (different values) the better
- Typically we talk about fitness being maximised
  - Some problems may be best posed as minimisation problems, but conversion is trivial

Population

- Holds (representations of) possible solutions
- Usually has a fixed size and is a *multiset* of genotypes
- Some sophisticated EAs also assert a spatial structure on the population e.g., a grid.
- Selection operators usually take whole population into account i.e., reproductive probabilities are *relative* to *current* generation
- Diversity of a population refers to the number of different fitnesses / phenotypes / genotypes present (note not the same thing)

Parent Selection Mechanism

- Assigns variable probabilities of individuals acting as parents depending on their fitnesses
- Usually probabilistic
  - high quality solutions more likely to become parents than low quality
  - but not guaranteed
  - even worst in current population usually has non-zero probability of becoming a parent
- This *stochastic* nature can aid escape from local optima

Variation Operators

- Role is to generate new candidate solutions
- Usually divided into two types according to their arity (number of inputs):
  - Arity 1 : mutation operators
  - Arity >1 : Recombination operators
  - Arity = 2 typically called crossover
- There has been much debate about relative importance of recombination and mutation
  - Nowadays most EAs use both
  - Choice of particular variation operators is representation dependant

Mutation

- Acts on one genotype and delivers another
- Element of randomness is essential and differentiates it from other unary heuristic operators
- Importance ascribed depends on representation and dialect:
  - Binary GAs – background operator responsible for preserving and introducing diversity
  - EP for FSM's/ continuous variables – only search operator
  - GP – hardly used
- May guarantee connectedness of search space and hence convergence proofs

Recombination

- Merges information from parents into offspring
- Choice of what information to merge is stochastic
- Most offspring may be worse, or the same as the parents
- Hope is that some are better by combining elements of genotypes that lead to good traits
- Principle has been used for millennia by breeders of plants and livestock

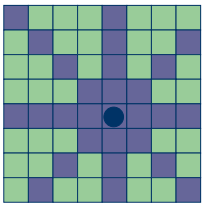
Survivor Selection

- a.k.a. ***replacement***
- Most EAs use fixed population size so need a way of going from (parents + offspring) to next generation
- Often deterministic
  - Fitness based : e.g., rank parents+offspring and take best
  - Age based: make as many offspring as parents and delete all parents
- Sometimes do combination (elitism)

Initialisation / Termination

- Initialisation usually done at random,
  - Need to ensure even spread and mixture of possible allele values
  - Can include existing solutions, or use problem-specific heuristics, to "seed" the population
- Termination condition checked every generation
  - Reaching some (known/hoped for) fitness
  - Reaching some maximum allowed number of generations
  - Reaching some minimum level of diversity
  - Reaching some specified number of generations without fitness improvement

Example: the 8 queens problem



8x8
40320
Combinations
(~112 hours)

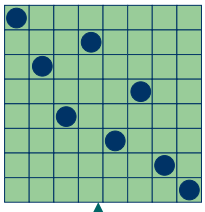
16x16
~21x10^12
Combinations
(~58x10^18 hours)

Place 8 queens on an 8x8 chessboard in such a way that they cannot check each other

The 8 queens problem: representation

Phenotype:
a board configuration

Genotype:
a permutation of
the numbers 1 - 8



Obvious mapping

1	3	5	2	6	4	7	8

8 Queens Problem: Fitness evaluation

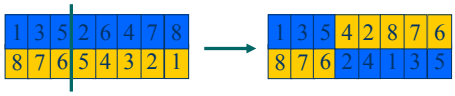
- Penalty of one queen:
  the number of queens she can check.
- Penalty of a configuration:
  the sum of the penalties of all queens.
- Note: penalty is to be minimized
- Fitness of a configuration:
  inverse penalty to be maximized

The 8 queens problem: Mutation

Small variation in one permutation, e.g.:
• swapping values of two randomly chosen positions,

The 8 queens problem: Recombination

Combining two permutations into two new permutations:
• choose random crossover point
• copy first parts into children
• create second part by inserting values from other parent:
  • in the order they appear there
  • beginning after crossover point
  • skipping values already in child



The 8 queens problem: Selection

- Parent selection:
  - Pick 5 parents and take best two to undergo crossover
- Survivor selection (replacement)
  - When inserting a new child into the population, choose an existing member to replace by:
  - sorting the whole population by decreasing fitness
  - enumerating this list from high to low
  - replacing the first with a fitness lower than the given child

8 Queens Problem: summary

Representation	Permutations
Recombination	"Cut-and-crossfill" crossover
Recombination probability	100%
Mutation	Swap
Mutation probability	80%
Parent selection	Best 2 out of random 5
Survival selection	Replace worst
Population size	100
Number of Offspring	2
Initialisation	Random
Termination condition	Solution or 10,000 fitness evaluation

Note that is is ***only one possible***
set of choices of operators and parameters

Typical behaviour of an EA

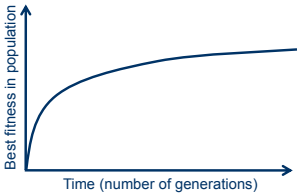
Phases in optimising on a 1-dimensional fitness landscape

Early phase:
quasi-random population distribution

Mid-phase:
population arranged around/on hills

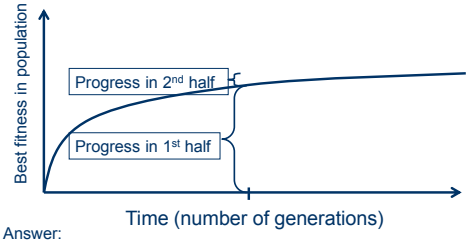
Late phase:
population concentrated on high hills

Typical run: progression of fitness



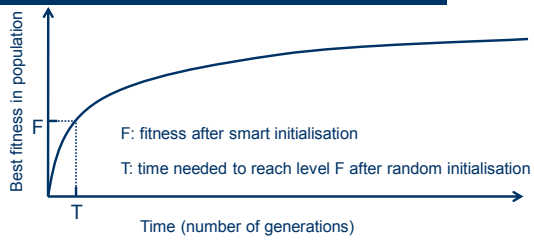
Typical run of an EA shows so-called “anytime behavior”

Are long runs beneficial?



- Answer:
  - it depends how much you want the last bit of progress
  - it may be better to do more shorter runs

Is it worth expending effort on smart initialisation?

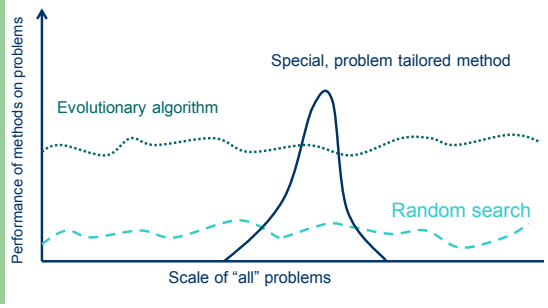


- Answer : it depends:
  - possibly, if good solutions/methods exist.
  - care is needed, see chapter on hybridisation

Evolutionary Algorithms in Context

- There are many views on the use of EAs as robust problem solving tools
- For most problems a problem-specific tool may:
  - perform better than a generic search algorithm on most instances,
  - have limited utility,
  - not do well on all instances
- Goal is to provide robust tools that provide:
  - evenly good performance
  - over a range of problems and instances

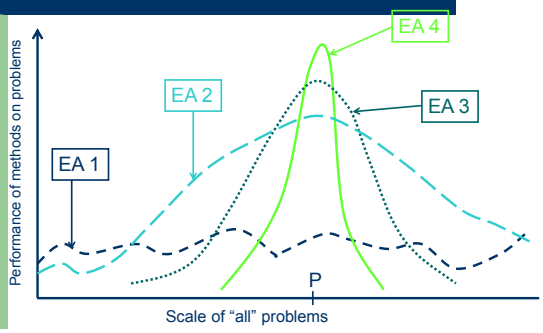
EAs as problem solvers: Goldberg's 1989 view



EAs and domain knowledge

- Trend in the 90's:
  adding problem specific knowledge to EAs
  (special variation operators, repair, etc)
- Result: EA performance curve "deformation":
  - better on problems of the given type
  - worse on problems different from given type
  - amount of added knowledge is variable
- Recent theory suggests the search for an "all-purpose" algorithm may be fruitless

Michalewicz' 1996 view



EC and Global Optimisation

- Global Optimisation: search for finding best solution x^* out of some fixed set S
- Deterministic approaches
  - e.g. box decomposition (branch and bound etc)
  - Guarantee to find x^*, but may run in super-polynomial time
- Heuristic Approaches (generate and test)
  - rules for deciding which $x \in S$ to generate next
  - no guarantees that best solutions found are globally optimal

Evolutionary Programming

Chapter 5

EP quick overview

- Developed: USA in the 1960's
- Early names: D. Fogel
- Typically applied to:
  - traditional EP: machine learning tasks by finite state machines
  - contemporary EP: (numerical) optimization
- Attributed features:
  - very open framework: any representation and mutation op's OK
  - crossbred with ES (contemporary EP)
  - consequently: hard to say what "standard" EP is
- Special:
  - no recombination
  - self-adaptation of parameters standard (contemporary EP)

EP technical summary tableau

Representation	Real-valued vectors
Recombination	None
Mutation	Gaussian perturbation
Parent selection	Deterministic
Survivor selection	Probabilistic ($\mu+\mu$)
Specialty	Self-adaptation of mutation step sizes (in meta-EP)

Historical EP perspective

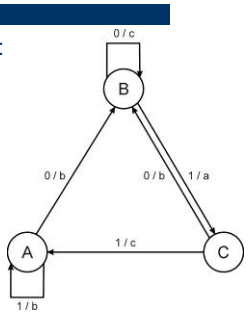
- EP aimed at achieving intelligence
- Intelligence was viewed as adaptive behaviour
- Prediction of the environment was considered a prerequisite to adaptive behaviour
- Thus: capability to predict is key to intelligence

Prediction by finite state machines

- Finite state machine (FSM):
  - States S
  - Inputs I
  - Outputs O
  - Transition function $\delta : S \times I \rightarrow S \times O$
  - Transforms input stream into output stream
- Can be used for predictions, e.g. to predict next input symbol in a sequence

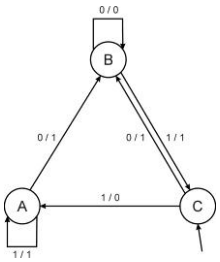
FSM example

- Consider the FSM with:
  - S = {A, B, C}
  - I = {0, 1}
  - O = {a, b, c}
  - δ given by a diagram



FSM as predictor

- Consider the following FSM
- Task: predict next input
- Quality: % of $in_{(i+1)} = out_i$
- Given initial state C
- Input sequence 011101
- Leads to output 110111
- Quality: 3 out of 5



Introductory example: evolving FSMs to predict primes

- P(n) = 1 if n is prime, 0 otherwise
- I = ***N*** = {1,2,3,…, n, …}
- O = {0,1}
- Correct prediction: $out_i = P(in_{(i+1)})$
- Fitness function:
  - 1 point for correct prediction of next input
  - 0 point for incorrect prediction
  - Penalty for “too much” states

Introductory example: evolving FSMs to predict primes

- Parent selection: each FSM is mutated once
- Mutation operators (one selected randomly):
  - Change an output symbol
  - Change a state transition (i.e. redirect edge)
  - Add a state
  - Delete a state
  - Change the initial state
- Survivor selection: $(\mu+\mu)$
- Results: overfitting, after 202 inputs best FSM had one state and both outputs were 0, i.e., it always predicted "not prime"

Modern EP

- No predefined representation in general
- Thus: no predefined mutation (must match representation)
- Often applies self-adaptation of mutation parameters
- In the sequel we present *one EP variant*, not the canonical EP

Representation

- For continuous parameter optimisation
- Chromosomes consist of two parts:
  - Object variables: $x_1,\ldots,x_n$
  - Mutation step sizes: $\sigma_1,\ldots,\sigma_n$
- Full size: $\langle x_1,\ldots,x_n, \sigma_1,\ldots,\sigma_n \rangle$

Mutation

- Chromosomes: $\langle x_1,\ldots,x_n, \sigma_1,\ldots,\sigma_n \rangle$
- $\sigma_i' = \sigma_i \cdot (1 + \alpha \cdot N(0,1))$
- $x_i' = x_i + \sigma_i' \cdot N_i(0,1)$
- $\alpha \approx 0.2$
- boundary rule: $\sigma' < \varepsilon_0 \Rightarrow \sigma' = \varepsilon_0$
- Other variants proposed & tried:
  - Lognormal scheme as in ES
  - Using variance instead of standard deviation
  - Mutate σ-last
  - Other distributions, e.g, Cauchy instead of Gaussian

Recombination

- None
- Rationale: one point in the search space stands for a species, not for an individual and there can be no crossover between species
- Much historical debate “mutation vs. crossover”
- Pragmatic approach seems to prevail today

Parent selection

- Each individual creates one child by mutation
- Thus:
  - Deterministic
  - Not biased by fitness

Survivor selection

- P(t): μ parents, P’(t): μ offspring
- Pairwise competitions in round-robin format:
  - Each solution x from P(t) $\cup$ P’(t) is evaluated against q other randomly chosen solutions
  - For each comparison, a "win" is assigned if x is better than its opponent
  - The μ solutions with the greatest number of wins are retained to be parents of the next generation
- Parameter q allows tuning selection pressure
- Typically q = 10

Example application: the Ackley function (Bäck et al ’93)

- The Ackley function (here used with n =30):

$$f(x) = -20 \cdot \exp\left(-0.2\sqrt{\frac{1}{n}\cdot\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$$

- Representation:
  - $-30 < x_i < 30$ (coincidence of 30’s!)
  - 30 variances as step sizes
- Mutation with changing object variables first !
- Population size μ = 200, selection with q = 10
- Termination : after 200000 fitness evaluations
- Results: average best solution is $1.4 \cdot 10^{-2}$

Example application: the Ackley function (Bäck et al '93)

- The Ackley function (here used with n =30):

$$f(x) = -20 \cdot \exp\left(-0.2\sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi x_i)\right) + 20 + e$$

- Representation:
  - $-30 < x_i < 30$ (coincidence of 30's!)
  - 30 variances as step sizes
- Mutation with changing object variables first !
- Population size μ = 200, selection with q = 10
- Termination : after 200000 fitness evaluations
- Results: average best solution is $1.4 \cdot 10^{-2}$

Example application: evolving checkers players (Fogel'02)

- Neural nets for evaluating future values of moves are evolved
- NNs have fixed structure with 5046 weights, these are evolved + one weight for "kings"
- Representation:
  - vector of 5046 real numbers for object variables (weights)
  - vector of 5046 real numbers for σ's
- Mutation:
  - Gaussian, lognormal scheme with σ-first
  - Plus special mechanism for the kings' weight
- Population size 15

Example application: evolving checkers players (Fogel'02)

- Tournament size q = 5
- Programs (with NN inside) play against other programs, no human trainer or hard-wired intelligence
- After 840 generation (6 months!) best strategy was tested against humans via Internet
- Program earned "expert class" ranking outperforming 99.61% of all rated players

Genetic Algorithms

Chapter 3

GA Quick Overview

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

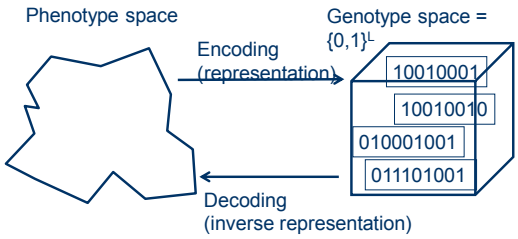
Genetic algorithms

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

SGA technical summary tableau

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

Representation

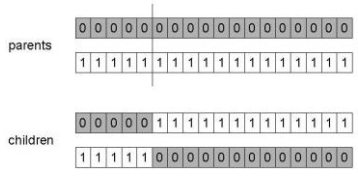


SGA reproduction cycle

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 p_c , otherwise copy parents
4. For each offspring apply mutation (bit-flip with probability p_m independently for each bit)
5. Replace the whole population with the resulting offspring

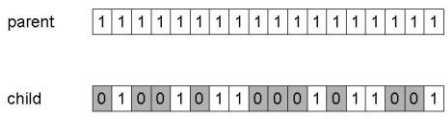
SGA operators: 1-point crossover

- 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)



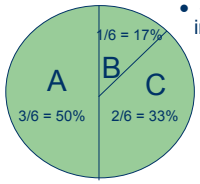
SGA operators: mutation

- 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



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

An example after Goldberg '89 (1)

- Simple problem: max x^2 over {0,1,…,31}
- GA approach:
  - Representation: binary code, e.g. 01101 ↔ 13
  - Population size: 4
  - 1-point xover, bitwise mutation
  - Roulette wheel selection
  - Random initialisation
- 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	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 1 0 0	18	324
Sum				2354
Average				588.5
Max				729

The simple GA

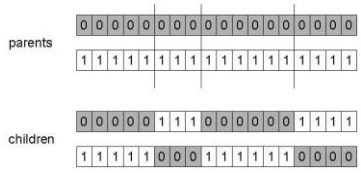
- Has been subject of many (early) studies
  - still often used as benchmark for novel GAs
- Shows many shortcomings, e.g.
  - Representation is too restrictive
  - Mutation & crossovers only applicable for bit-string & integer representations
  - Selection mechanism sensitive for converging populations with close fitness values
  - Generational population model (step 5 in SGA repr. cycle) can be improved with explicit survivor selection

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

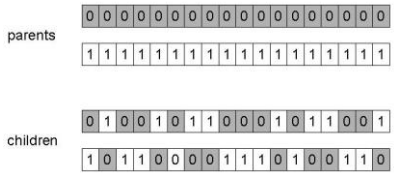
n-point crossover

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



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



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

Floating point mutations 1

General scheme of floating point mutations

$$\bar{x} = \langle x_1, \ldots, x_l \rangle \rightarrow \bar{x}' = \langle x_1', \ldots, x_l' \rangle$$
$$x_i, x_i' \in [LB_i, UB_i]$$

- Uniform mutation:
  x_i' drawn randomly (uniform) from $[LB_i, UB_i]$
- Analogous to bit-flipping (binary) or random resetting (integers)

Floating point mutations 2

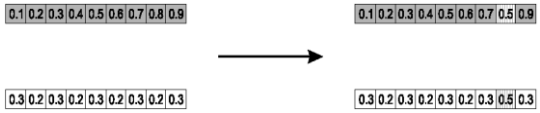
- Non-uniform mutations:
  - Many methods proposed,such as time-varying range of change etc.
  - Most schemes are probabilistic but usually only make a small change to value
  - Most common method is to add random deviate to each variable separately, taken from N(0, σ) Gaussian distribution and then curtail to range
  - Standard deviation σ controls amount of change (2/3 of deviations will lie in range (- σ to + σ)

Crossover operators for real valued GAs

- Discrete:
  - each allele value in offspring *z* comes from one of its parents *(x,y)* with equal probability: $z_i = x_i$ or y_i
  - Could use n-point or uniform
- Intermediate
  - exploits idea of creating children "between" parents (hence a.k.a. *arithmetic* recombination)
  - $z_i = \alpha x_i + (1 - \alpha) y_i$ where $\alpha : 0 \leq \alpha \leq 1$.
  - The parameter α can be:
    - constant: uniform arithmetical crossover
    - variable (e.g. depend on the age of the population)
    - picked at random every time

Single arithmetic crossover

- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$
- Pick a single gene (*k*) at random,
- child$_1$ is: $\langle x_1, \ldots, x_k, \alpha \cdot y_k + (1-\alpha) \cdot x_k, \ldots, x_n \rangle$
- reverse for other child. e.g. with $\alpha = 0.5$

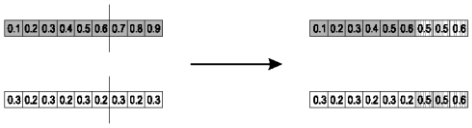


Simple arithmetic crossover

- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$
- Pick random gene *(k)* after this point mix values
- child$_1$ is:

$$\langle x_1, \ldots, x_k, \alpha \cdot y_{k+1} + (1-\alpha) \cdot x_{k+1}, \ldots, \alpha \cdot y_n + (1-\alpha) \cdot x_n \rangle$$

- reverse for other child. e.g. with $\alpha = 0.5$

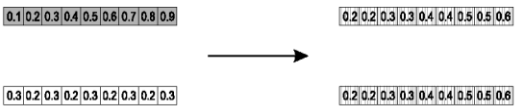


Whole arithmetic crossover

- Most commonly used
- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$
- child$_1$ is:

$$a \cdot \bar{x} + (1-a) \cdot \bar{y}$$

- reverse for other child. e.g. with $\alpha = 0.5$



Permutation Representations

- Ordering/sequencing problems form a special type
- 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

Permutation representation: TSP example

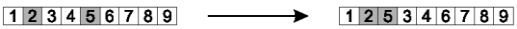
- Problem:
  - Given n cities
  - Find a complete tour with minimal length
- Encoding:
  - Label the cities 1, 2, … , *n*
  - One complete tour is one permutation (e.g. for n =4 [1,2,3,4], [3,4,2,1] are OK)
- Search space is BIG:
  for 30 cities there are 30! ≈ 10^{32} possible tours

Mutation operators 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

Insert Mutation for permutations

- Pick two allele values at random
- Move the second to follow the first, shifting the rest along to accommodate
- Note that this preserves most of the order and the adjacency information

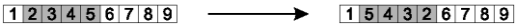


Swap mutation for permutations

- Pick two alleles at random and swap their positions
- Preserves most of adjacency information (4 links broken), disrupts order more

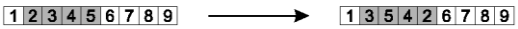
Inversion mutation for permutations

- Pick two alleles at random and then invert the substring between them.
- Preserves most adjacency information (only breaks two links) but disruptive of order information



Scramble mutation for permutations

- Pick a subset of genes at random
- Randomly rearrange the alleles in those positions



(note subset does not have to be contiguous)

Crossover operators for permutations

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

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

Partially Mapped Crossover (PMX)

Informal procedure for parents P1 and P2:

1. Choose random segment and copy it from P1
2. Starting from the first crossover point look for elements in that segment of P2 that have not been copied
3. For each of these *i* look in the offspring to see what element *j* has been copied in its place from P1
4. Place *i* into the position occupied *j* in P2, since we know that we will not be putting *j* there (as is already in offspring)
5. If the place occupied by *j* in P2 has already been filled in the offspring *k*, put *i* in the position occupied by *k* in P2
6. Having dealt with the elements from the crossover segment, the rest of the offspring can be filled from P2.

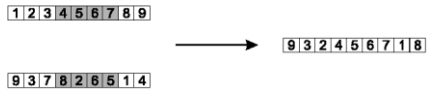
Second child is created analogously

PMX example

- Step 1

- Step 2

- Step 3



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:
  - one offspring is generated per generation,
  - one member of population replaced,
- Generation Gap
  - the proportion of the population replaced
  - 1.0 for GGA, 1/pop_size for SSGA

Fitness Based Competition

- Selection can occur in two places:
  - Selection from current generation to take part in mating (parent selection)
  - Selection from parents + offspring to go into next generation (survivor selection)
- Selection operators work on whole individual
  - i.e. they are representation-independent
- Distinction between selection
  - operators: define selection probabilities
  - algorithms: define how probabilities are implemented

Implementation example: SGA

- Expected number of copies of an individual *i*

$$E(n_i) = \mu \cdot f(i)/\langle f \rangle$$

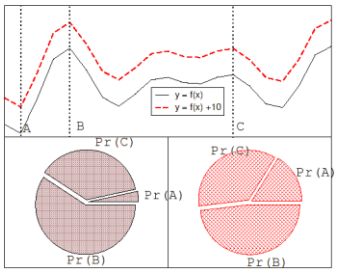
(μ = pop.size, f(i) = fitness of i, $\langle f \rangle$ avg. fitness in pop.)

- Roulette wheel algorithm:
  - Given a probability distribution, spin a 1-armed wheel *n* times to make *n* selections
  - No guarantees on actual value of n_i
- Baker's SUS algorithm:
  - *n* evenly spaced arms on wheel and spin once
  - Guarantees $floor(E(n_i)) \leq n_i \leq ceil(E(n_i))$

Fitness-Proportionate Selection

- Problems include
  - One highly fit member can rapidly take over if rest of population is much less fit: Premature Convergence
  - At end of runs when fitnesses are similar, lose selection pressure
  - Highly susceptible to function transposition
- Scaling can fix last two problems
  - Windowing: $f'(i) = f(i) - \beta^t$
    - where β is worst fitness in this (last n) generations
  - Sigma Scaling: $f'(i) = max(f(i) - (\langle f \rangle - c \cdot \sigma_f), 0.0)$
    - c is a constant, usually 2.0, <f> = mean(f), σ_f stand dev.

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(expected # of offspring)
- This imposes a sorting overhead on the algorithm, but this is usually negligible compared to the fitness evaluation time

Linear Ranking

$$P_{lin-rank}(i) = \frac{(2-s)}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}$$

- Parameterised by factor *s:* $1.0 < s \leq 2.0$
  - measures advantage of best individual
  - in GGA this is the number of children allotted to it
- Simple 3 member example

	Fitness	Rank	P_{selFP}	P_{selLR} $(s=2)$	P_{selLR} $(s=1.5)$
A	1	1	0.1	0	0.167
B	5	2	0.5	0.67	0.5
C	4	2	0.4	0.33	0.33
Sum	10		1.0	1.0	1.0

Tournament Selection

- All methods above rely on global population statistics
  - Could be a bottleneck esp. on parallel machines
  - Relies on presence of external 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 fittest contestant always wins (deterministic) or this happens with probability *p*
  - Whether contestants are picked with replacement
    - Picking without replacement increases selection pressure
    - Especially in deterministic tournaments
- For $k = 2$, time for fittest individual to take over population is the same as linear ranking with $s = 2 \cdot p$

Survivor Selection

- Most of methods above used for parent selection
- Survivor selection can be divided into two approaches:
  - Age-Based Selection
    - e.g. SGA
    - In SSGA can implement as "delete-random" (not recommended) or as first-in-first-out (a.k.a. delete-oldest)
  - Fitness-Based Selection
    - Using one of the methods above

Two Special Cases

- Elitism
  - Widely used in both population models (GGA, SSGA)
  - Always keep at least one copy of the fittest solution so far
- GENITOR: a.k.a. “delete-worst”
  - From Whitley’s original Steady-State algorithm (he also used linear ranking for parent selection)
  - Rapid takeover : use with large populations or “no duplicates” policy

Example application of order based GAs: JSSP

Precedence constrained job shop scheduling problem

- J is a set of jobs.
- O is a set of operations
- M is a set of machines
- *Able* $\subseteq$ O $\times$ M defines which machines can perform which operations
- *Pre* $\subseteq$ O $\times$ O defines which operation should precede which
- *Dur* : $\subseteq$ O $\times$ M $\rightarrow$ IR defines the duration of o $\in$ O on m $\in$ M

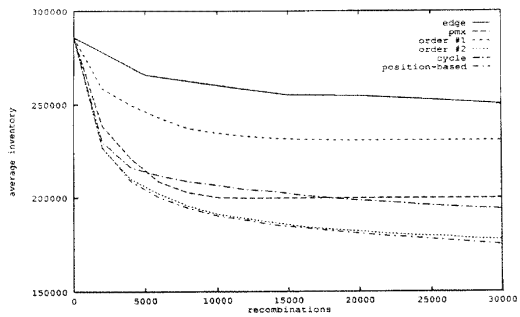
The goal is now to find a schedule that is:

- Complete: all jobs are scheduled
- Correct: all conditions defined by *Able* and *Pre* are satisfied
- Optimal: the total duration of the schedule is minimal

Precedence constrained job shop scheduling GA

- Representation: individuals are permutations of operations
- Permutations are decoded to schedules by a decoding procedure
  - take the first (next) operation from the individual
  - look up its machine (here we assume there is only one)
  - assign the earliest possible starting time on this machine, subject to
    - machine occupation
    - precedence relations holding for this operation in the schedule created so far
- fitness of a permutation is the duration of the corresponding schedule (to be minimized)
- use any suitable mutation and crossover
- use roulette wheel parent selection on inverse fitness
- Generational GA model for survivor selection
- use random initialisation

JSSP example: operator comparison



Evolution strategies

Chapter 4

ES quick overview

- Developed: Germany in the 1970's
- Early names: I. Rechenberg, H.-P. Schwefel
- Typically applied to:
  - numerical optimisation
- Attributed features:
  - fast
  - good optimizer for real-valued optimisation
  - relatively much theory
- Special:
  - self-adaptation of (mutation) parameters standard

ES technical summary tableau

Representation	Real-valued vectors
Recombination	Discrete or intermediary
Mutation	Gaussian perturbation
Parent selection	Uniform random
Survivor selection	(μ,λ) or $(\mu+\lambda)$
Specialty	Self-adaptation of mutation step sizes

Introductory example

- Task: minimimise $f : R^n \rightarrow R$
- Algorithm: "two-membered ES" using
  - Vectors from R^n directly as chromosomes
  - Population size 1
  - Only mutation creating one child
  - Greedy selection

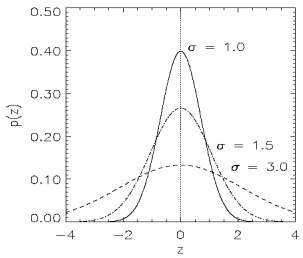
Introductory example: pseudocde

- Set t = 0
- Create initial point $x^t = \langle x_1^t,\ldots,x_n^t \rangle$
- REPEAT UNTIL (*TERMIN.COND* satisfied) DO
- Draw z_i from a normal distr. for all i = 1,…,n
- $y_i^t = x_i^t + z_i$
- IF $f(x^t) < f(y^t)$ THEN $x^{t+1} = x^t$
  - ELSE $x^{t+1} = y^t$
  - FI
  - Set t = t+1
- OD

Introductory example: mutation mechanism

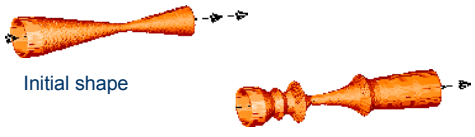
- z values drawn from normal distribution $N(\xi,\sigma)$
  - mean ξ is set to 0
  - variation σ is called mutation step size
- σ is varied on the fly by the "1/5 success rule":
- This rule resets σ after every k iterations by
  - $\sigma = \sigma / c$ if $p_s > 1/5$
  - $\sigma = \sigma \cdot c$ if $p_s < 1/5$
  - $\sigma = \sigma$ if $p_s = 1/5$
- where p_s is the % of successful mutations, $0.8 \leq c < 1$

Illustration of normal distribution



Another historical example: the jet nozzle experiment

Task: to optimize the shape of a jet nozzle
Approach: random mutations to shape + selection

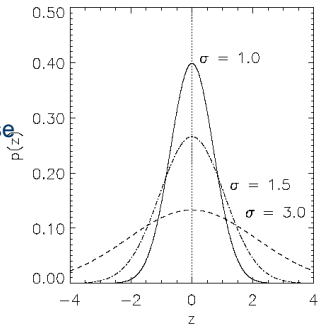


Initial shape

Final shape

Genetic operators: mutations (2)

The one dimensional case



Representation

- Chromosomes consist of three parts:
  - Object variables: $x_1,\ldots,x_n$
  - Strategy parameters:
    - Mutation step sizes: $\sigma_1,\ldots,\sigma_{n_\sigma}$
    - Rotation angles: $\alpha_1,\ldots,\alpha_{n_\alpha}$
- Not every component is always present
- Full size: $\langle x_1,\ldots,x_n, \sigma_1,\ldots,\sigma_n, \alpha_1,\ldots,\alpha_k \rangle$
- where k = n(n-1)/2 (no. of i,j pairs)

Mutation

- Main mechanism: changing value by adding random noise drawn from normal distribution
- $x'_i = x_i + N(0,\sigma)$
- Key idea:
  - σ is part of the chromosome $\langle x_1,\ldots,x_n, \sigma \rangle$
  - σ is also mutated into σ' (see later how)
- Thus: mutation step size σ is coevolving with the solution x

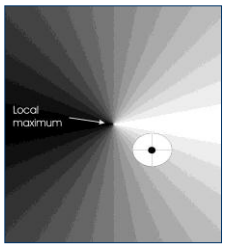
Mutate σ first

- Net mutation effect: $\langle x, \sigma \rangle \rightarrow \langle x', \sigma' \rangle$
- Order is important:
  - first $\sigma \rightarrow \sigma'$ (see later how)
  - then $x \rightarrow x' = x + N(0,\sigma')$
- Rationale: new $\langle x', \sigma' \rangle$ is evaluated twice
  - Primary: x' is good if f(x') is good
  - Secondary: σ' is good if the x' it created is good
- Reversing mutation order this would not work

Mutation case 1: Uncorrelated mutation with one σ

- Chromosomes: $\langle x_1,\ldots,x_n, \sigma \rangle$
- $\sigma' = \sigma \cdot \exp(\tau \cdot N(0,1))$
- $x'_i = x_i + \sigma' \cdot N(0,1)$
- Typically the "learning rate" $\tau \propto 1/\sqrt{n}$
- And we have a boundary rule $\sigma' < \varepsilon_0 \Rightarrow \sigma' = \varepsilon_0$

Mutants with equal likelihood



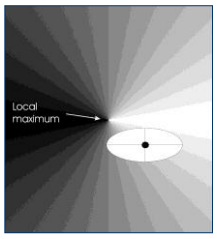
Big difference in effect to fitness on the two axis

Circle: mutants having the same chance to be created

Mutation case 2: Uncorrelated mutation with n σ's

- Chromosomes: $\langle x_1,\ldots,x_n, \sigma_1,\ldots, \sigma_n \rangle$
- $\sigma'_i = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1))$
- $x'_i = x_i + \sigma'_i \cdot N_i(0,1)$
- Two learning rate parameters:
  - τ' overall learning rate
  - τ coordinate wise learning rate
- $\tau \propto 1/\sqrt{2n}$ and $\tau \propto 1/\sqrt{2\sqrt{n}}$
- And $\sigma_i' < \varepsilon_0 \Rightarrow \sigma_i' = \varepsilon_0$

Mutants with equal likelihood



Smaller difference in effect to fitness on the two axis

Ellipse: mutants having the same chance to be created

Mutation case 3: Correlated mutations

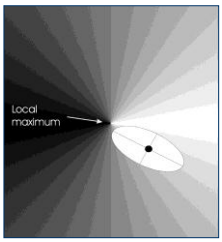
- Chromosomes: $\langle x_1,\ldots,x_n, \sigma_1,\ldots, \sigma_n, \alpha_1,\ldots, \alpha_k \rangle$
- where $k = n \cdot (n-1)/2$
- and the covariance matrix C is defined as:
  - $c_{ii} = \sigma_i^2$
  - $c_{ij} = 0$ if i and j are not correlated
  - $c_{ij} = \frac{1}{2} \cdot (\sigma_i^2 - \sigma_j^2) \cdot \tan(2 \alpha_{ij})$ if i and j are correlated
- Note the numbering / indices of the α's

Correlated mutations cont'd

The mutation mechanism is then:

- $\sigma'_i = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i (0,1))$
- $\alpha'_j = \alpha_j + \beta \cdot N (0,1)$
- $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{N(0,C')}$
  - $\boldsymbol{x}$ stands for the vector $\langle x_1,\ldots,x_n \rangle$
  - $\boldsymbol{C'}$ is the covariance matrix $\boldsymbol{C}$ after mutation of the α values
- $\tau \propto 1/(2 n)^{\frac{1}{2}}$ and $\tau \propto 1/(2 n^{\frac{1}{2}})^{\frac{1}{2}}$ and $\beta \approx 5°$
- $\sigma'_i < \varepsilon_0 \Rightarrow \sigma'_i = \varepsilon_0$ and
- $| \alpha'_j | > \pi \Rightarrow \alpha'_j = \alpha'_j - 2 \pi \text{ sign}(\alpha'_j)$

Mutants with equal likelihood



Even smaller difference in effect to fitness on the two axis

Ellipse: mutants having the same chance to be created

Recombination

- Creates one child
- Acts per variable / position by either
  - Averaging parental values, or
  - Selecting one of the parental values
- From two or more parents by either:
  - Using two selected parents to make a child
  - Selecting two parents for each position anew

Names of recombinations

	Two fixed parents	Two parents selected for each i
$z_i = (x_i + y_i)/2$	Local intermediary	Global intermediary
z_i is x_i or y_i chosen randomly	Local discrete	Global discrete

Parent selection

- Parents are selected by uniform random distribution whenever an operator needs one/some
- Thus: ES parent selection is unbiased - every individual has the same probability to be selected
- Note that in ES "parent" means a population member (in GA's: a population member selected to undergo variation)

Survivor selection

- Applied after creating λ children from the μ parents by mutation and recombination
- Deterministically chops off the "bad stuff"
- Basis of selection is either:
  - The set of children only: (μ,λ)-selection
  - The set of parents and children: $(\mu+\lambda)$-selection

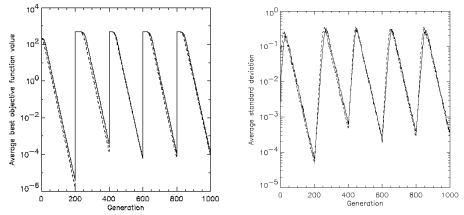
Survivor selection cont'd

- $(\mu+\lambda)$-selection is an elitist strategy
- (μ,λ)-selection can "forget"
- Often (μ,λ)-selection is preferred for:
  - Better in leaving local optima
  - Better in following moving optima
  - Using the + strategy bad σ values can survive in $\langle x,\sigma \rangle$ too long if their host x is very fit
- Selective pressure in ES is very high ($\lambda \approx 7 \cdot \mu$ is the common setting)

Self-adaptation illustrated

- Given a dynamically changing fitness landscape (optimum location shifted every 200 generations)
- Self-adaptive ES is able to
  - follow the optimum and
  - adjust the mutation step size after every shift !

Self-adaptation illustrated cont'd



Changes in the fitness values (left) and the mutation step sizes (right)

Prerequisites for self-adaptation

- $\mu > 1$ to carry different strategies
- $\lambda > \mu$ to generate offspring surplus
- Not "too" strong selection, e.g., $\lambda \approx 7 \cdot \mu$
- (μ,λ)-selection to get rid of misadapted σ's
- Mixing strategy parameters by (intermediary) recombination on them

Example application: the cherry brandy experiment

- Task to create a colour mix yielding a target colour (that of a well known cherry brandy)
- Ingredients: water + red, yellow, blue dye
- Representation: $\langle$ w, r, y ,b $\rangle$ no self-adaptation!
- Values scaled to give a predefined total volume (30 ml)
- Mutation: lo / med / hi σ values used with equal chance
- Selection: (1,8) strategy

Example application: cherry brandy experiment cont'd

- Fitness: students effectively making the mix and comparing it with target colour
- Termination criterion: student satisfied with mixed colour
- Solution is found mostly within 20 generations
- Accuracy is very good

Example application: the Ackley function (Bäck et al '93)

- The Ackley function (here used with n =30):

$$f(x) = -20 \cdot \exp\left(-0.2\sqrt{\frac{1}{n}\cdot\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$$

- Evolution strategy:
  - Representation:
    - $-30 < x_i < 30$ (coincidence of 30's!)
    - 30 step sizes
  - (30,200) selection
  - Termination : after 200000 fitness evaluations
  - Results: average best solution is $7.48 \cdot 10^{-8}$ (very good)

Genetic Programming

Chapter 6

GP quick overview

- Developed: USA in the 1990's
- Early names: J. Koza
- Typically applied to:
  - machine learning tasks (prediction, classification…)
- Attributed features:
  - competes with neural nets and alike
  - needs huge populations (thousands)
  - slow
- Special:
  - non-linear chromosomes: trees, graphs
  - mutation possible but not necessary (disputed!)

GP technical summary tableau

Representation	Tree structures
Recombination	Exchange of subtrees
Mutation	Random change in trees
Parent selection	Fitness proportional
Survivor selection	Generational replacement

Introductory example: credit scoring

- Bank wants to distinguish good from bad loan applicants
- Model needed that matches historical data

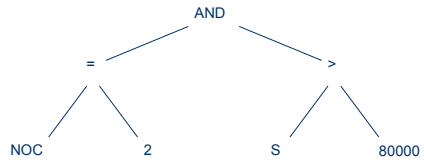
ID	No of children	Salary	Marital status	OK?
ID-1	2	45000	Married	0
ID-2	0	30000	Single	1
ID-3	1	40000	Divorced	1
…				

Introductory example: credit scoring

- A possible model:
  IF (NOC = 2) AND (S > 80000) THEN good ELSE bad
- In general:
  IF formula THEN good ELSE bad
- Only unknown is the right formula, hence
- Our search space (phenotypes) is the set of formulas
- Natural fitness of a formula: percentage of well classified cases of the model it stands for
- Natural representation of formulas (genotypes) is: parse trees

Introductory example: credit scoring

IF (NOC = 2) AND (S > 80000) THEN good ELSE bad
can be represented by the following tree

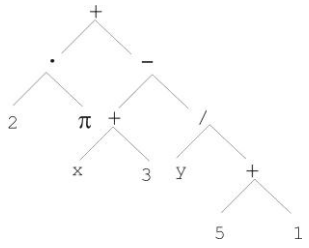


Tree based representation

- Trees are a universal form, e.g. consider
- Arithmetic formula $2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$
- Logical formula $(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$
- Program

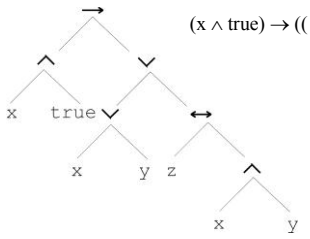
```
i = 1;
while (i < 20)
{
    i = i + 1
}
```


Tree based representation



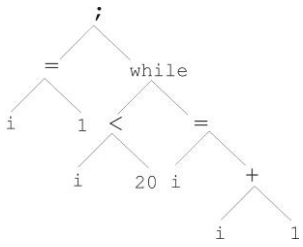
$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$

Tree based representation



$(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$

Tree based representation




```
i =1;
while (i < 20)
{
        i = i +1
}
```


Tree based representation

- In GA, ES, EP chromosomes are linear structures (bit strings, integer string, real-valued vectors, permutations)
- Tree shaped chromosomes are non-linear structures
- In GA, ES, EP the size of the chromosomes is fixed
- Trees in GP may vary in depth and width

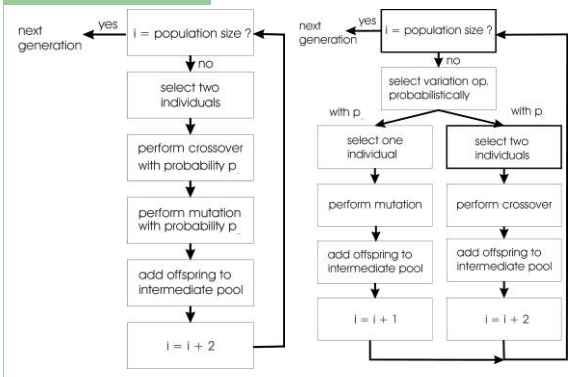
Tree based representation

- Symbolic expressions can be defined by
  - Terminal set T
  - Function set F (with the arities of function symbols)
- Adopting the following general recursive definition:
  1. Every $t \in T$ is a correct expression
  2. $f(e_1, \ldots, e_n)$ is a correct expression if $f \in F$, arity(f)=n and $e_1, \ldots, e_n$ are correct expressions
  3. There are no other forms of correct expressions
- In general, expressions in GP are not typed (closure property: any $f \in F$ can take any $g \in F$ as argument)

Offspring creation scheme

Compare

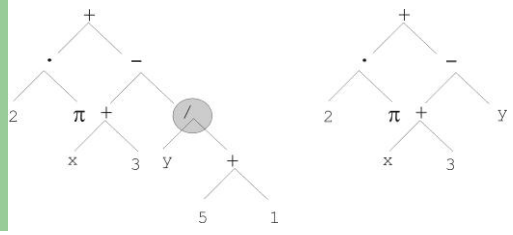
- GA scheme using crossover AND mutation sequentially (be it probabilistically)
- GP scheme using crossover OR mutation (chosen probabilistically)



GA flowchart GP flowchart

Mutation

- Most common mutation: replace randomly chosen subtree by randomly generated tree

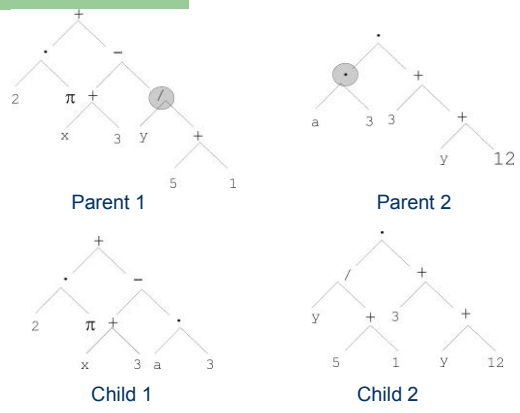


Mutation cont'd

- Mutation has two parameters:
  - Probability p_m to choose mutation vs. recombination
  - Probability to chose an internal point as the root of the subtree to be replaced
- Remarkably p_m is advised to be 0 (Koza'92) or very small, like 0.05 (Banzhaf et al. '98)
- The size of the child can exceed the size of the parent

Recombination

- Most common recombination: exchange two randomly chosen subtrees among the parents
- Recombination has two parameters:
  - Probability p_c to choose recombination vs. mutation
  - Probability to chose an internal point within each parent as crossover point
- The size of offspring can exceed that of the parents



Selection

- Parent selection typically fitness proportionate
- Over-selection in very large populations
  - rank population by fitness and divide it into two groups:
  - group 1: best x% of population, group 2 other (100-x)%
  - 80% of selection operations chooses from group 1, 20% from group 2
  - for pop. size = 1000, 2000, 4000, 8000 x = 32%, 16%, 8%, 4%
  - motivation: to increase efficiency, %'s come from rule of thumb
- Survivor selection:
  - Typical: generational scheme (thus none)
  - Recently steady-state (keep part of parent pool) is becoming popular for its elitism

Initialisation

- Maximum initial depth of trees D_{max} is set
- Full method (each branch has depth = D_{max}):
  - nodes at depth $d < D_{max}$ randomly chosen from function set F
  - nodes at depth $d = D_{max}$ randomly chosen from terminal set T
- Grow method (each branch has depth $\leq D_{max}$):
  - nodes at depth $d < D_{max}$ randomly chosen from $F \cup T$
  - nodes at depth $d = D_{max}$ randomly chosen from T
- Common GP initialisation: ramped half-and-half, where grow & full method each deliver half of initial population

Bloat

- Bloat = “survival of the fattest”, i.e., the tree sizes in the population are increasing over time
- Ongoing research and debate about the reasons
- Needs countermeasures, e.g.
  - Prohibiting variation operators that would deliver “too big” children
  - Parsimony pressure: penalty for being oversized

Problems involving “physical” environments

- Trees for data fitting vs. trees (programs) that are “really” executable
- Execution can change the environment → the calculation of fitness
- Example: robot controller
- Fitness calculations mostly by simulation, ranging from expensive to extremely expensive (in time)
- But evolved controllers are often to very good

Example application: symbolic regression

- Given some points in $\mathbf{R}^2$, $(x_1, y_1), \ldots, (x_n, y_n)$
- Find function f(x) s.t. $\forall i = 1, \ldots, n : f(x_i) = y_i$
- Possible GP solution:
  - Representation by F = {+, -, /, sin, cos}, T = $\mathbf{R} \cup \{x\}$
  - Fitness is the error $err(f) = \sum_{i=1}^{n} (f(x_i) - y_i)^2$
  - All operators standard
  - pop.size = 1000, ramped half-half initialisation
  - Termination: n “hits” or 50000 fitness evaluations reached (where “hit” is if $| f(x_i) - y_i | < 0.0001$)

Discussion

Is GP:

The art of evolving computer programs ?
Means to automated programming of computers?
GA with another representation?

Learning Classier Systems

Karsten Øster Lundqvist

LCS Background

- Derived from the evolutionary studies (e.g. Genetic Algorithms) of Holland in the mid-seventies. First described in
  - J. H. Holland. Progress in Theoretical Biology IV, chapter Adaptation, pages 263–293. Academic Press, 1976.
- This is only a short overview, so read more in your own time (Especially if you are on LCS...)
  - Introduction to Evolutionary Computing, chap 7
  - Clever Algorithms Nature-Inspired Programming Recipes, Lulu, section 3.9. (Includes a working ZCS in Ruby)
  - Learning Classifier Systems: A Brief Introduction, Larry Bull (on BB)
  - http://www.cs.bris.ac.uk/~kovacs/lcs/search.html

LCS Background

- Creating an adaptive system based on evolution
  - Suited for problems that have: Perpetually novel events with significant noise, continual real-time requirements for action, implicitly or inexactly defined goals, and sparse payoff or reinforcement obtainable only through long sequences of tasks. (Clever Algorithms)
- Cooperative population of sub-solutions
- 'if < conditions > then < action >' production rules within population
- Reinforcement learning

LCS System

- The state of the environment is transmitted to the system via a set of **detectors** whose output is put on a **message list**.
- This message list may also contain other signals posted there by rules that have fired on previous cycles, or the detector signals from previous cycles. This means it may act as a form of memory.
- The **condition part** of each rule in the rule base is then examined to see if it matches the current message list. Those rules that match are tagged as belonging to the **match set** for this cycle. Note that different rules in the match set may advocate different actions.
- The rules in the match set are grouped according to the action they advocate, and a predicted payoff for each action is calculated from the individual rules' predictions.
- Based on these predictions, an action is chosen, and all of the rules that advocated that action are tagged as belonging to the **action set** for that time step.
- The action is posted to the message list. The action consists of instructions to be read by the effectors (which interact with the environment), and (optionally) signals to the left on the "internal" message list.
- Periodically a reward signal is received from the environment. A **credit allocation mechanism** is used to distribute that reward amongst the rules, usually amongst the chain of action sets that led to the reward.
- Periodically the GA is run on the population of rules to generate new rules and delete poorly performing ones.

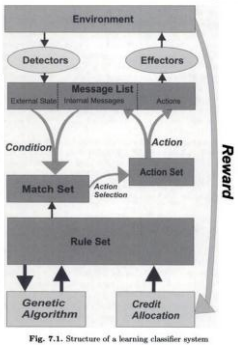


Fig. 7.1. Structure of a learning classifier system

LCS Rules

- Binary with schemata
- E.g. 0#1
  - Satisfied by: (bit1 == 1) AND (bit3 == 0)
  - i.e. bit2 is unspecified, and satisfy both 0 and 1.

Pseudo Code

Did I mention you should read in your own time?

Algorithm 3.9.1: Pseudocode for the LCS.

```
Input: EnvironmentDetails
Output: Population
env ← InitializeEnvironment(EnvironmentDetails);
Population ← InitializePopulation();
ActionSet_{t-1} ← ∅;
Input_{t-1} ← ∅;
Reward_{t-1} ← ∅;
while ¬StopCondition() do
    Input_t ← env;
    Matchset ← GenerateMatchSet(Population, Input_t);
    Prediction ← GeneratePrediction(Matchset);
    Action ← SelectionAction(Prediction);
    ActionSet_t ← GenerateActionSet(Action, Matchset);
    Reward_t ← ExecuteAction(Action, env);
    if ActionSet_{t-1} ≠ ∅ then
        Payoff_t ← CalculatePayoff(Reward_{t-1}, Prediction);
        PerformLearning(ActionSet_{t-1}, Payoff_t, Population);
        RunGeneticAlgorithm(ActionSet_{t-1}, Input_{t-1}, Population);
    end
    if LastStepOfTask(env, Action) then
        Payoff_t ← Reward_t;
        PerformLearning(ActionSet_t, Payoff_t, Population);
        RunGeneticAlgorithm(ActionSet_t, Input_t, Population);
        ActionSet_{t-1} ← ∅;
    else
        ActionSet_{t-1} ← ActionSet_t;
        Input_{t-1} ← Input_t;
        Reward_{t-1} ← Reward_t;
    end
end
```


ZCS

- LCS can be quite slow at learning / impractical
- Wilson's "Zeroth-Level" Classier System
  - Stripped down LCS
    - No message list

ZCS

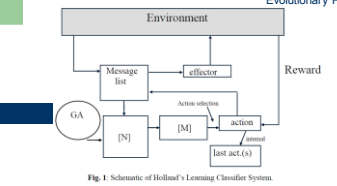


Fig. 1: Schematic of Holland's Learning Classifier System.

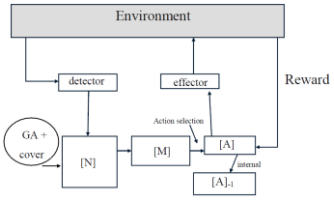


Fig. 2: Schematic of ZCS.

Real Code

- zcs.rb on BB from Clever Algorithms
- Multiplexer - Static problem

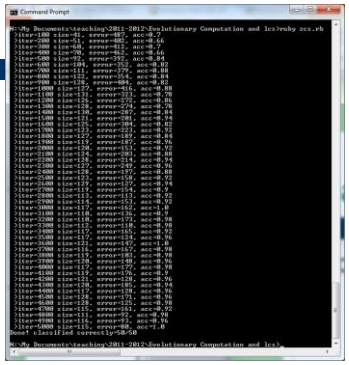
```
def neg(bit)
  return (bit==1) ? 0 : 1
end

def target_function(s)
  ints = Array.new(6){|i| s[i].chr.to_i}
  x0,x1,x2,x3,x4,x5 = ints
  return neg(x0)*neg(x1)*x2 + neg(x0)*x1*x3 + x0*neg(x1)*x4 + x0*x1*x5
end

def new_classifier(condition, action, gen, p1=10.0, e1=0.0, f1=10.0)
  other = {}
  other[:condition],other[:action],other[:lasttime] = condition, action, gen
  other[:pred], other[:error], other[:fitness] = p1, e1, f1
  other[:exp], other[:setsize], other[:num] = 0.0, 1.0, 1.0
  return other
end

def copy_classifier(parent)
  copy = {}
  parent.keys.each do |k|
    copy[k] = (parent[k].kind_of? String) ? ""+parent[k] : parent[k]
  end
```


Real Code



Real Code in C

- https://ccrma.stanford.edu/CCRMA/Courses/220b/Lectures/6/Examples/cbn/code/src/zcs.c

XCS

- eXtended Learning Classier System
  - http://www.youtube.com/watch?v=jYp_hgwewPc&feature=related

XCS

- Rule fitness not based on payoff received from environment, but on the accuracy of predictions in payoff.
  - Connect LCS with reinforced learning

i) Each rule's error is updated: $\varepsilon_j = \varepsilon_j + \beta(|P - p_j| - \varepsilon_j)$
ii) Rule's predictions are then updated: $p_j = p_j + \beta(P - p_j)$
iii) Each rule's accuracy κ_j is determined: $\kappa = \alpha(\varepsilon_j/\varepsilon_0)^{-v}$ or $\kappa=1$ where $\varepsilon < \varepsilon_0$
iv) A relative accuracy κ_j' is determined for each rule by dividing its accuracy by the total of the accuracies in the set.
v) The relative accuracy is then used to adjust the classifier's fitness F_j using the moyenne adaptive modifee (MAM) procedure: If the fitness has been adjusted $1/\beta$ times, $F_j = F_j + \beta(\kappa_j' - F_j)$. Otherwise F_j is set to the average of the current and previous values of κ_j'.

XCS

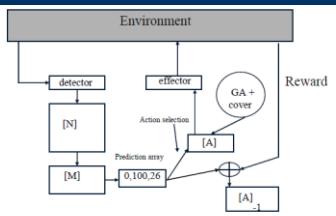


Fig. 3: Schematic of XCS.

Working with Evolutionary Algorithms

Chapter 14

Issues considered

- Experiment design
- Algorithm design
- Test problems
- Measurements and statistics
- Some tips and summary

Experimentation

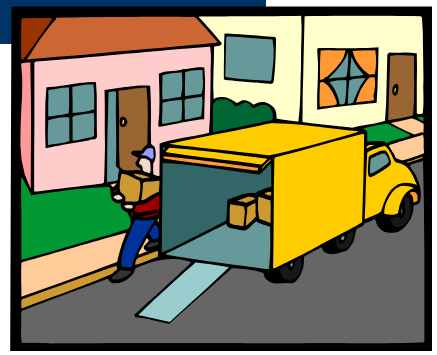
- Has a goal or goals
- Involves algorithm design and implementation
- Needs problem(s) to run the algorithm(s) on
- Amounts to running the algorithm(s) on the problem(s)
- Delivers measurement data, the results
- Is concluded with evaluating the results in the light of the given goal(s)
- Is often documented (see tutorial on paper writing)

EA experimentation

- EA objectives determined by problem context:

- Design (engineering) problems – single ‘good’ solution required.
- Control (optimization) problems – requiring many ‘good’ yet ‘timely’ solutions.

Example: Production Perspective



- Optimizing Internet shopping delivery routes
  - Different destinations each day
  - Limited time to run algorithm each day
  - Must *always* be *reasonably* good route in limited time

Example: Design Perspective

- Optimizing spending on improvements to national road network
  - –Total cost: billions of Euro
  - –Computing costs negligible
  - –Six months to run algorithm on hundreds computers
  - –Many runs possible
  - –Must produce *very* good result just *once*

Perspectives of an EA's goals

- Design perspective:

  find a very good solution at least once
- Production perspective:

  find a good solution at almost every run
- Academic perspective:

  must meet scientific standards

These perspectives have very different implications when evaluating EA results.

Algorithm design

- Design a representation
- Design a way of mapping a genotype to a phenotype
- Design a way of evaluating an individual
- Design suitable mutation operator(s)
- Design suitable recombination operator(s)
- Decide how to select individuals to be parents
- Decide how to select individuals for the next generation (how to manage the population)
- Decide how to start: initialization method
- Decide how to stop: termination criterion

Test problems for experimental comparisons

- Use problem instances from an academic repository
- Use randomly generated problem instances
- Use real life problem instances

Test problems for experimental comparisons

- 5 DeJong functions
- 25 “hard” objective functions
- Frequently encountered or otherwise important variants of given practical problem
- Selection from recognized benchmark problem repository (“challenging” by being NP--- ?!)
- Problem instances made by random generator

Choice has severe implications on

- generalizability and
- scope of the results

Bad example

- I invented “tricky mutation”
- Showed that it is a good idea by:
  - Running standard (?) GA and tricky GA
  - On 10 objective functions from the literature
  - Finding tricky GA better on 7, equal on 1, worse on 2 cases
- I wrote it down in a paper
- And it got published!
- Q: what did I learned from this experience?
- Q: is this good work?

Bad example

- What did I (my readers) did not learn:
  - How relevant are these results (test functions)?
  - What is the scope of claims about the superiority of the tricky GA?
  - Is there a property distinguishing the 7 good and the 2 bad functions?
  - Can the results be generalized ? (Is the tricky GA applicable for other problems? Which ones?)

Getting Problem Instances 1

- Testing on real data
- Advantages:
  - Results are application oriented
- Disadvantages
  - Can be few available sets of real data
  - May be commercial sensitive – difficult to publish and to allow others to compare
  - Results are hard to generalize

Getting Problem Instances 2

- Standard data sets in problem repositories, e.g.:
  - OR-Library
    http://www.ms.ic.ac.uk/info.html
  - UCI Machine Learning Repository
    www.ics.uci.edu/~mlearn/MLRepository.html
- Advantage:
  - Tried and tested problems and instances (hopefully)
  - Much other work on these → results comparable
- Disadvantage:
  - Not real – might miss crucial aspect
  - Algorithms get tuned for popular test suites

Getting Problem Instances 3

- Problem instance generators produce simulated data for given parameters, e.g.:
  - GA/EA Repository of Test Problem Generators

    http://www.cs.uwyo.edu/~wspears/generators.html
- Advantage:
  - Allow systematic investigation of an objective function parameter range
  - Can be shared allowing comparisons with other researchers
- Disadvantage:
  - Not real – might miss crucial aspect
  - Given generator might have hidden bias

Basic rules of experimentation

- EAs are stochastic →
  never draw any conclusion from a single run
  - perform sufficient number of independent runs
  - use statistical measures (averages, standard deviations)
  - use statistical tests to assess reliability of conclusions
- EA experimentation is about comparison →
  always do a fair competition
  - use the same amount of resources for the competitors
  - try different competition limits
  - use the same performance measures

Things to Measure

Many different ways. Examples:

- Average result in given time
- Average time for given result
- Proportion of runs within % of target
- Best result over *n* runs
- Amount of computing required to reach target in given time with % confidence
- …

What time units do we use?

- Elapsed time?
  - Depends on computer, network, etc…
- CPU Time?
  - Depends on skill of programmer, implementation, etc…
- Generations?
  - Difficult to compare when parameters like population size change
- Evaluations?
  - Evaluation time could depend on algorithm, e.g. direct vs. indirect representation

Measures

- Performance measures (off-line)
  - Efficiency (alg. speed)
    - CPU time
    - No. of steps, i.e., generated points in the search space
  - Effectivity (alg. quality)
    - Success rate
    - Solution quality at termination
- "Working" measures (on-line)
  - Population distribution (genotypic)
  - Fitness distribution (phenotypic)
  - Improvements per time unit or per genetic operator
  - …

Performance measures

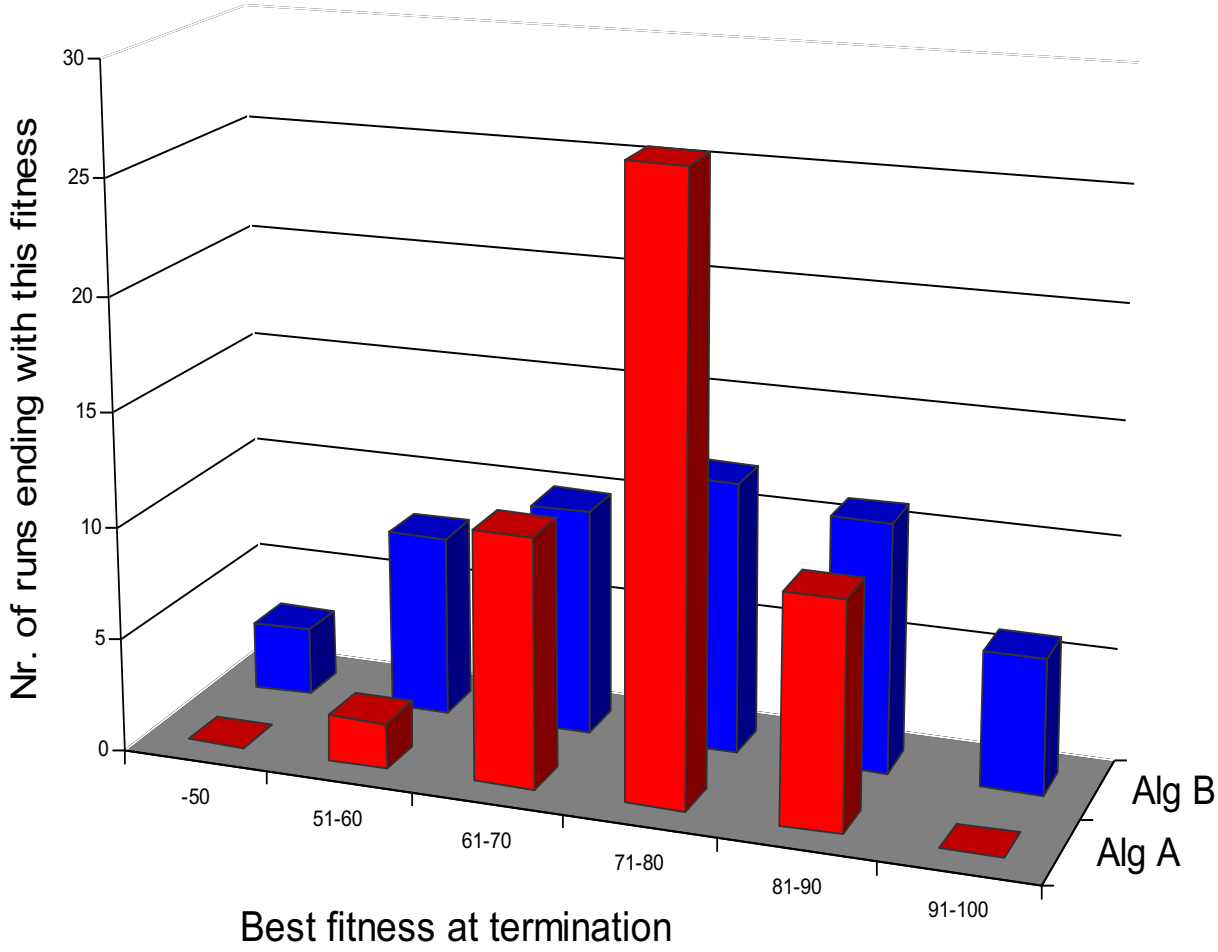
- No. of generated points in the search space
  = no. of fitness evaluations
  (don't use no. of generations!)
- AES: average no. of evaluations to solution
- SR: success rate = % of runs finding a solution (individual with acceptabe quality / fitness)
- MBF: mean best fitness at termination, i.e., best per run, mean over a set of runs
- SR ≠ MBF
  - Low SR, high MBF: good approximizer (more time helps?)
  - High SR, low MBF: "Murphy" algorithm

Fair experiments

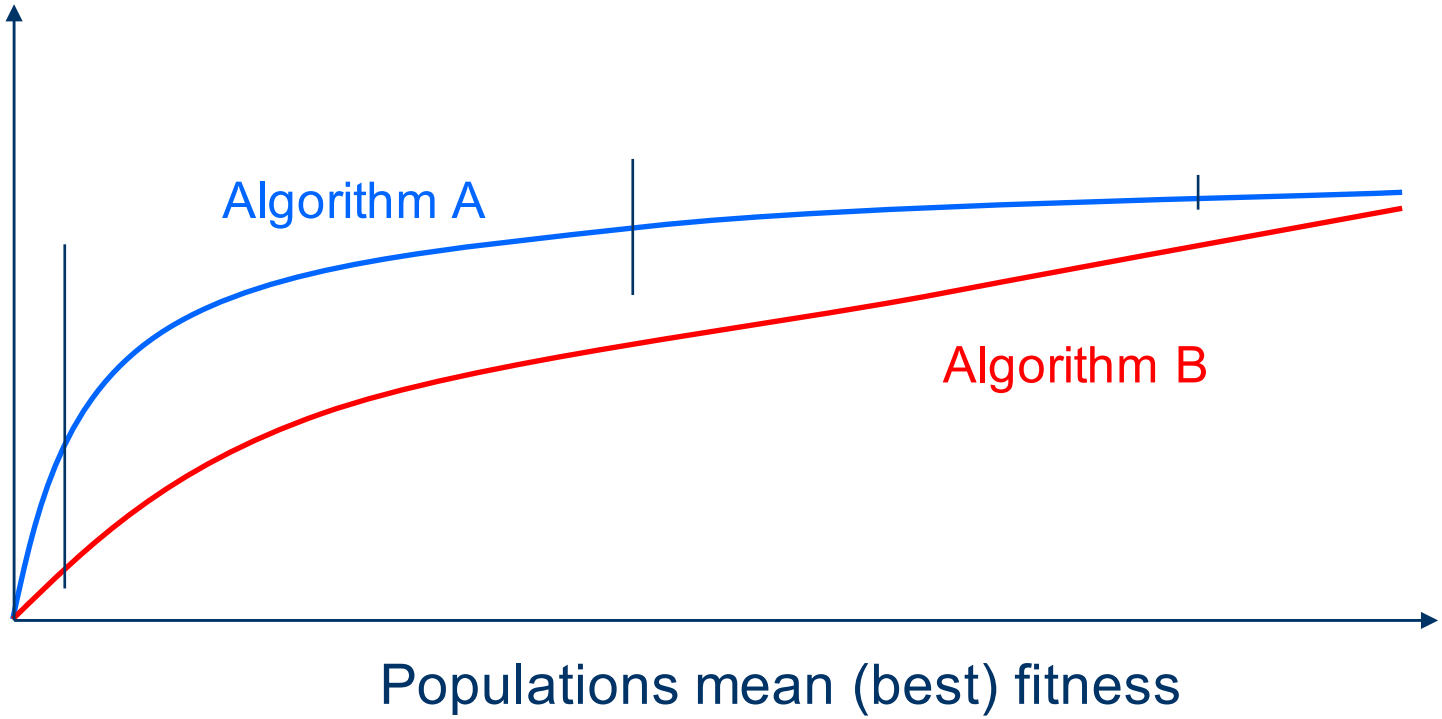
- Basic rule: use the same computational limit for each competitor
- Allow each EA the same no. of evaluations, but
  - Beware of hidden labour, e.g. in heuristic mutation operators
  - Beware of possibly fewer evaluations by smart operators
- EA vs. heuristic: allow the same no. of steps:
  - Defining “step” is crucial, might imply bias!
  - Scale-up comparisons eliminate this bias

Example: off-line performance measure evaluation

Which algorith is better?
Why?
When?

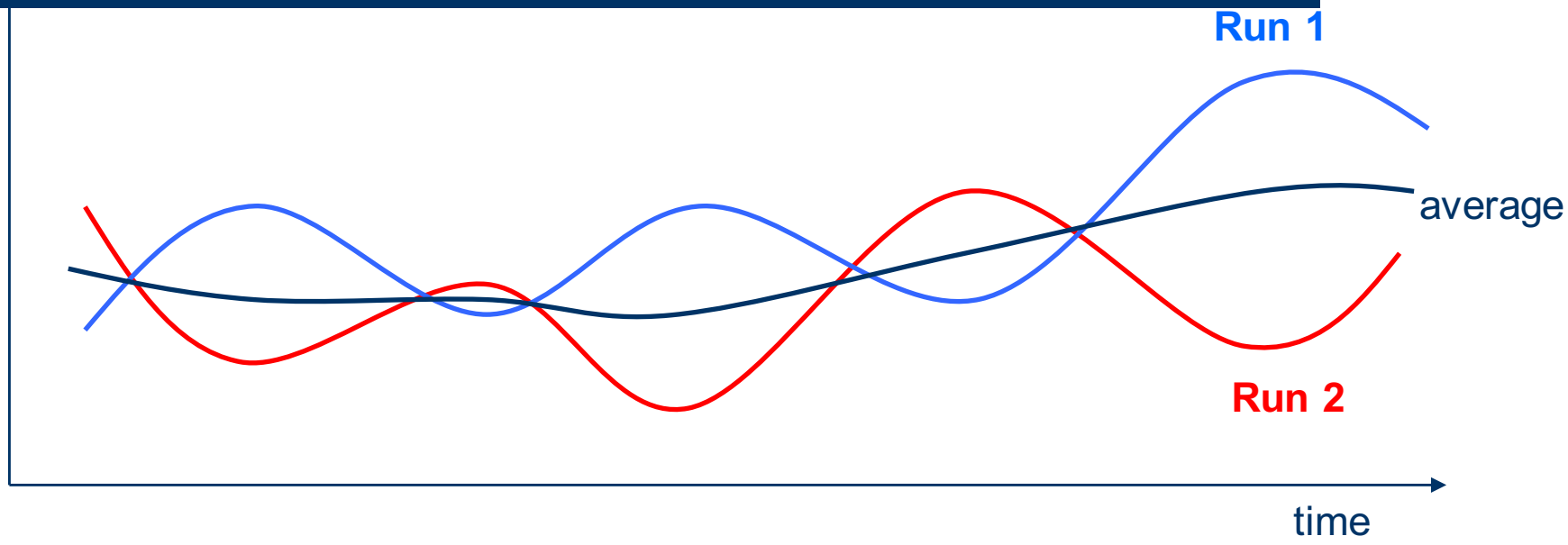


Example: on-line performance measure evaluation



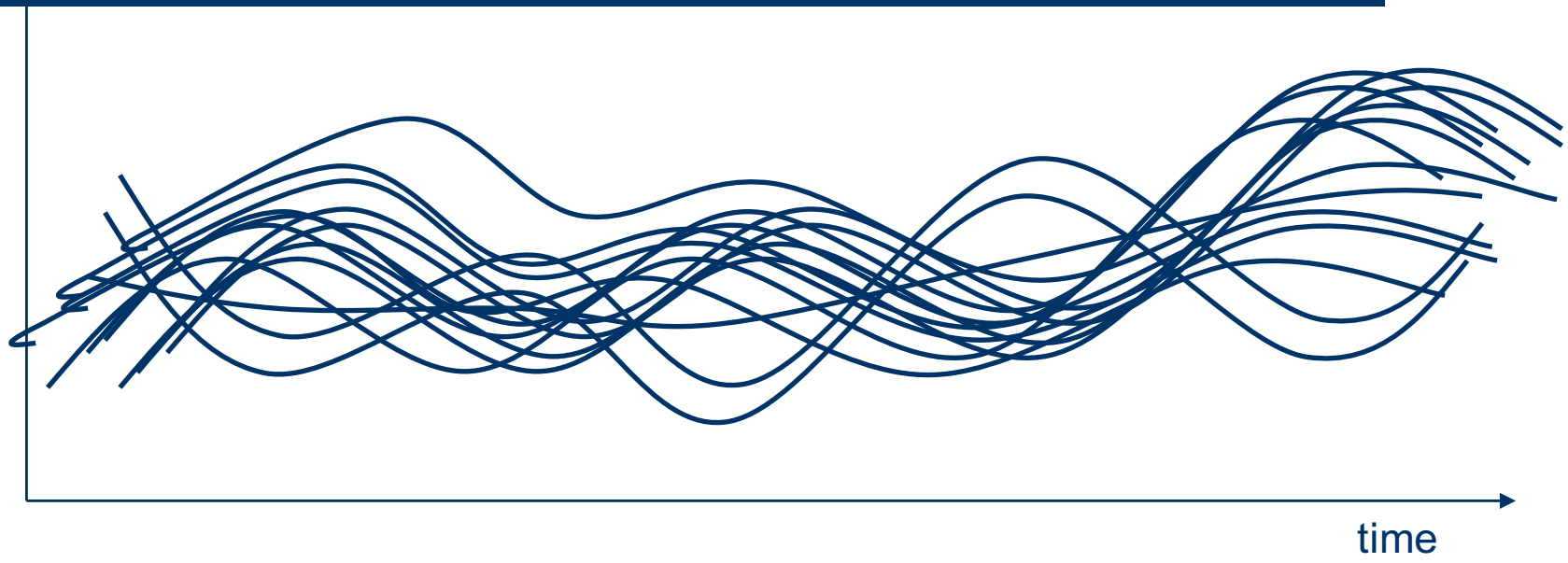
Which algorith is better? Why? When?

Example: averaging on-line measures



Averaging can “choke” interesting onformation

Example: overlaying on-line measures



Overlay of curves can lead to very “cloudy” figures

Statistical Comparisons and Significance

- Algorithms are stochastic
- Results have element of “luck”
- Sometimes can get away with less rigour – e.g. parameter tuning
- For scientific papers where a claim is made: “Newbie recombination is better ran uniform crossover”, need to show statistical significance of comparisons

Example

Trial	Old Method	New Method
1	500	657
2	600	543
3	556	654
4	573	565
5	420	654
6	590	712
7	700	456
8	472	564
9	534	675
10	512	643
Average	545.7	612.3

Is the new method better?

Example (cont'd)

Trial	Old Method	New Method
1	500	657
2	600	543
3	556	654
4	573	565
5	420	654
6	590	712
7	700	456
8	472	564
9	534	675
10	512	643
Average	545.7	612.3
SD	73.5962635	73.5473317
T-test	**0.07080798**	

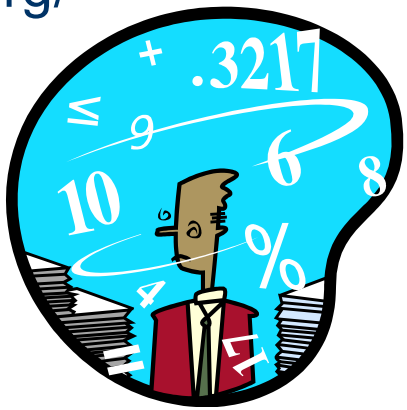
- Standard deviations supply additional info
- T-test (and alike) indicate the chance that the values came from the same underlying distribution (difference is due to random effetcs) E.g. with 7% chance in this example.

Statistical tests

- T-test assummes:
  - Data taken from continuous interval or close approximation
  - Normal distribution
  - Similar variances for too few data points
  - Similar sized groups of data points
- Other tests:
  - Wilcoxon – preferred to t-test where numbers are small or distribution is not known.
  - F-test – tests if two samples have different variances.

Statistical Resources

- http://fonsg3.let.uva.nl/Service/Statistics.html
- http://department.obg.cuhk.edu.hk/ResearchSupport/
- http://faculty.vassar.edu/lowry/webtext.html
- Microsoft Excel
- http://www.octave.org/



Better example: problem setting

- I invented myEA for problem X
- Looked and found 3 other EAs and a traditional benchmark heuristic for problem X in the literature
- Asked myself when and why is myEA better

Better example: experiments

- Found/made problem instance generator for problem X with 2 parameters:
  - *n* (problem size)
  - *k* (some problem specific indicator)
- Selected 5 values for *k* and 5 values for *n*
- Generated 100 problem instances for all combinations
- Executed all alg's on each instance 100 times (benchmark was also stochastic)
- Recorded AES, SR, MBF values w/ same comp. limit (AES for benchmark?)
- Put my program code and the instances on the Web

Better example: evaluation

- Arranged results “in 3D” (*n,k*) + performance
  (with special attention to the effect of *n*, as for scale-up)
- Assessed statistical significance of results
- Found the niche for my_EA:
  - Weak in … cases, strong in - - - cases, comparable otherwise
  - Thereby I answered the “when question”
- Analyzed the specific features and the niches of each algorithm thus answering the “why question”
- Learned a lot about problem X and its solvers
- Achieved generalizable results, or at least claims with well-identified scope based on solid data
- Facilitated reproducing my results → further research

Some tips

- Be organized
- Decide what you want & define appropriate measures
- Choose test problems carefully
- Make an experiment plan (estimate time when possible)
- Perform sufficient number of runs
- Keep all experimental data (never throw away anything)
- Use good statistics (“standard” tools from Web, MS)
- Present results well (figures, graphs, tables, …)
- Watch the scope of your claims
- Aim at generalisable results
- Publish code for reproducibility of results (if applicable)

Theory

Chapter 11

Overview

- Motivations and problems
- Holland's Schema Theorem
  - Derivation, Implications, Refinements
- Dynamical Systems & Markov Chain Models
- Statistical Mechanics
- Reductionist Techniques
- Techniques for Continuous Spaces
- No Free Lunch ?

Why Bother with Theory?

- Might provide performance guarantees
  - Convergence to the global optimum can be guaranteed providing certain conditions hold
- Might aid better algorithm design
  - Increased understanding can be gained about operator interplay etc.
- Mathematical Models of EAs also inform theoretical biologists
- Because you never know ….

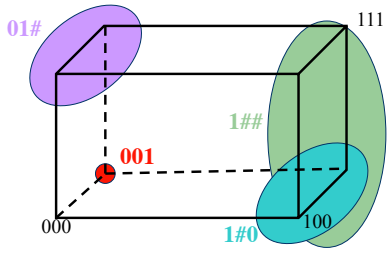
Problems with Theory ?

- EAs are vast, complex dynamical systems with many degrees of freedom
- The type of problems for which they do well, are precisely those it is hard to model
- The degree of randomness involved means
  - stochastic analysis techniques must be used
  - Results tend to describe average behaviour
- After 100 years of work in theoretical biology, they are still using fairly crude models of very simple systems ….

Holland's Schema Theorem

- A schema (*pl.* schemata) is a string in a ternary alphabet (0,1 # = "don't care") representing a hyperplane within the solution space.
  - E.g. 0001# #1# #0#, ##1##0## etc
- Two values can be used to describe schemata,
  - the ***Order*** (number of defined positions) = 6,2
  - the ***Defining Length*** - length of sub-string between outmost defined positions = 9, 3

Example Schemata



H	o(H)	d(H)
001	3	2
01#	2	1
1#0	2	2
1##	1	0

Schemata

- Aggregation
  - Simpler to analyse the effect of evolutionary operators of schemata then on individuals
  - Need to show how
    - operator increase fitness of the schemata
    - Or disrupts the fitness of schemata in population
- Implicit parallelism
  - a schema represents many possible individuals
  - Manipulating one schema affects many individuals
  - Computationally efficient
  - Often cited as on of the performance advantages of EA mechanism

Schema Fitnesses

- The true "fitness" of a schema *H* is taken by averaging over all possible values in the "don't care" positions, but this is effectively sampled by the population, giving an estimated fitness *f(H)*
- With Fitness Proportionate Selection
  P_s(instance of *H*) = *n(H,t) * f(H,t) / (<f> * μ)*
  therefore proportion in next parent pool is:
  m'(H,t+1) = m(H,t) * f(H,t) / <f>

<f>: mean population fitness

Schema Disruption I

- One Point Crossover selects a crossover point at random from the *l-1* possible points
- For a schema with defining length *d* the random point will fall inside the schema with probability = *d(H) / (l-1).*
- If recombination is applied with probability P_c the survival probability is 1.0 - P_c*d(H)/(l-1)

Schema Disruption II

- The probability that bit-wise mutation with probability P_m will NOT disrupt the schemata is simply the probability that mutation does NOT occur in any of the defining positions,

  $P_{survive}$ (mutation) = (1- $Pm)o^{(H)}$

  $= 1 – o(H) * P_m + terms\ in\ P_m^2 + …$
- For low mutation rates, this survival probability under mutation approximates to 1 - $o(h)* P_m$

o(h): Order of Schema h

The Schema Theorem

- Put together, the proportion of a schema *H* in successive generations varies as:

$$m(H,t+1) \geq m(H,t) \cdot \frac{f(H)}{<f>} \cdot \left[1 - \left(p_c \cdot \frac{d(H)}{l-1}\right)\right] \cdot [1 - p_m \cdot o(H)].$$

- Condition for schema to increase its representation is:

$$\frac{f(H)}{<f>} > \left[1 - \left(p_c \cdot \frac{d(H)}{l-1}\right)\right] \cdot [1 - p_m \cdot o(H)]$$

- Inequality is due to convergence affecting crossover disruption, exact versions have been developed

Implications 1: Operator Bias

- One Point Crossover
  - less likely to disrupt schemata which have **short** defining lengths relative to their order, as it will tend to keep together adjacent genes
  - this is an example of ***Positional Bias***
- Uniform Crossover
  - No positional bias since choices independent
  - BUT is far more likely to pick 50% of the bits from each parent, less likely to pick (say) 90% from one
  - this is called ***Distributional Bias***
- Mutation
  - also shows Distributional Bias, but not Positional

Operator Biases ctd

- Operator Bias has been extensively studied by Eschelman and Schaffer (empirically) and theoretically by Spears & DeJong.
- Results emphasise the importance of utilising all available problem specific knowledge when choosing a representation and operators for a new problem

Implications 2:The Building Block Hypothesis

- Closely related to the Schema Theorem is the "Building Block Hypothesis" (Goldberg 1989)
- This suggests that Genetic Algorithms work by discovering and exploiting "building blocks" - groups of closely interacting genes - and then successively combining these (via crossover) to produce successively larger building blocks until the problem is solved.
- Has motivated study of ***Deceptive*** problems
  - Based on the notion that the lower order schemata within a partition lead the search in the opposite direction to the global optimum
  - If interested look up Walsh functions

Criticisms of the Schema Theorem

- It presents an inequality that does not take into account the constructive effects of crossover and mutation
  - Exact versions have been derived
  - Have links to Price's theorem in biology
- Because the mean population fitness, and the estimated fitness of a schema will vary from generation to generation, it says ***nothing*** about gen. t+2 etc.
- "Royal Road" problems constructed to be GA-easy based on schema theorem turned out to be better solved by random mutation hill-climbers
- BUT it remains a useful conceptual tool and has historical importance

Theories

- Many others exist
  - Landscape Metrics
  - Markow Chains
  - Vose's Dynamical Systems Model
  - Statistical Mechanics Analysis
  - Reductionist Approaches
- Read the book…

Other Landscape Metrics

- As well as epistasis and deception, several other features of search landscapes have been proposed as providing explanations as to what sort of problems will prove hard for GAs
  - fitness-distance correlation
  - number of peaks present in the landscape
  - the existence of plateaus
  - all these imply a neighbourhood structure to the search space.
- It must be emphasised that these only hold for one operator

Markov Chain Analysis

- A system is called a Markov Chain if
  - It can exist only in one of a finite number of states
  - So can be described by a variable X^t
  - The probability of being in any state at time $t+1$ depends only on the state at time t.
- Frequently these probabilities can be defined in a transition matrix, and the theory of stochastic processes allows us to reason using them.
- Has been used to provide convergence proofs
- Can be used with F and M to create exact probabilistic models for binary coded Gas, but these are huge

No Free Lunch Theorems

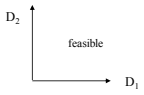
- **IN LAYMAN'S TERMS,**
  - **Averaged over all problems**
  - **For any performance metric related to number of distinct points seen**
  - **All non-revisiting black-box algorithms will display the same performance**
- **Implications**
  - **New black box algorithm is good for one problem => probably poor for another**
  - **Makes sense not to use "black-box algorithms"**
- **Lots of ongoing work showing counter-examples**

No Free Lunch

- Assumptions
  - Black box – uses no problem instance or specific knowledge
  - Nonrevisiting – means in a generate and test scenario the same point is not used twice
  - EAs revisit but avoid this by keeping a history and excluding children already seen
- Implications
  - If you make an algorithm that appears to be the best for solving some class of problems then it pays for it by not being very good for another class.
  - For a given problem can avoid the NFL problem by building in problem-specific knowledge

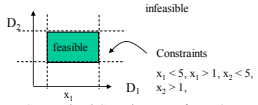
EAs and Constraints

- Constrained problem definition
  - Suppose a problem has n variables $x_1, x_2, \ldots x_n$
  - A feasible solution is any choice of the x_i which satisfies a set of conditions (or constraints)
  - x_i drawn from a set D_i
    - Discrete values (finite set)
    - Continuous data
  - Search space (S) of solution
    - Is $S=D_1xD_2x\ldots xD_n$



Free Search space for n=2

- Many practical problems are constrained
- Many constrained problems are NP-hard or NP-complete
- Handling constraints not straightforward with an EA
  - EA operators are 'blind'
  - Generate infeasible as well as feasible solutions



Constrained Search space for n=2

Types of problem

- Free search space
  - Each point can be tested for membership independently
  - Feasible search space is conjunction of points satisfying the condition
- Objective function and constraints
  - Can be used to define a search problem
  - FOPs are naturally solved by EAs since the objective function is also a fitness function
- Solve COPs and CSP by
  - Mapping to a FOP
  - Can use direct or indirect constraint handling

	Objective function	
Constraints	Yes	No
Yes	Constrained Optimisation problem (COP)	Constraint Satisfaction problem (CSP)
No	Free Optimisation Problem (FOP)	?

Examples:

FOP: find the max of some function y=f(x)

COP: find the max of f(x1, x2) in the interval $1\leq x1\leq n$, $1\leq x2\leq n$

CSP: find a colouring of a graph G(V,E) such that adjacent nodes have different colours and only three colours are used in total

Handling Constraints (1)

- Direct handling
  - Generate a phenotype from the genotype
  - Check all constraints satisfied (i.e. solution is feasible)
  - Use optimization function to choose best feasible solution
- Indirect handling
  - Convert each constraint to an objective
  - Combine objectives using penalty functions
  - Problem is now a FOP and can ignore explicit constraint handling
  - Conversion is done before solving the problem with an EA

Example:

COP: find the max of f(x1, x2) in the interval $1\leq x1\leq n$, $1\leq x2\leq n$ (assume non-negative for simplicity)

Convert by defining:

f1(x1) = if $(1\leq x1\leq n)$ then 1 else 0

f2(x2) = if $(1\leq x2\leq n)$ then 1 else 0

FOP : fnew(x1, x2)= f(x1,x2)-w1*f1(x1)-w2*f2(x2)

Where w1 and w2 are weights chosen to give infeasible point in solution space poor fitness

Exercise: formulate the Sudoku grid problem as a CSP and convert it into a FOP

Handling Constraints (2)

- Ideally only want to generate feasible solutions
  - Avoid time spent in unprofitable areas of search space
  - Focus on 'good' solutions
  - Can use domain specific knowledge to help eliminate infeasible possibilities

Usually the feasible set (F) is much smaller than the search space (S)

An interior point is on inside F and exterior point outside F

- Four basic mechanisms
  - Use of penalty functions
    - Fitness reduced in proportion to number of constraints violated
  - Repair an infeasible soln
    - Convert it into a feasible soln
  - use an alphabet and operators to reduce chances of infeasible soln
    - Create an unambiguous mapping from genotype to phenotype
  - Always generate feasible soln
    - Decode each genotype so that phenotype is always feasible
    - Many to one mapping of genotype to phenotype

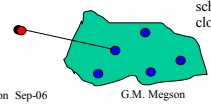
Handling Constraints (2)

- More on penalty functions
  - Popular and easy to use
  - Can work on disjoint Feasible regions
  - Useful when global optimal is close to boundary of feasible region
- Success depends on
  - Balance of exploring infeasible region and not wasting time
  - Severity of penalties for violating constraints
- Can use a distance metric
  - Indicates how far infeasible point is from boundary of F
- Severe penalties
  - Infeasible point near boundary of F discarded
  - Can slow down the search
- Weak penalties
  - Infeasible points can dominate feasible solution
  - Algorithm can stagnate
- Penalty function types
  - Static
    - Choose weights by experimentation
    - Distance metric (Boolean or Euclidean distance)
  - Dynamic
    - Allow weight penalties to vary with time
  - Adaptive
    - Learn the weights as algorithm runs

Handling Constraints (3)

- Repair functions
  - Takes an infeasible point and produces a feasible point
  - For example using a local search to reduce constraint violations
- Repair type
  - Baldwinian
    - Fitness of repaired point is allocated to infeasible point
  - Larmarkian
    - Infeasible point is overwritten by the feasible point
- Michalewicz's repair method
  - Maintain two populations P_s and P_r
  - P_s are search points
  - P_r are reference points (i.e. inside F)
  - when an infeasible point is created the nearest reference point is located
  - A 'line' is drawn between the infeasible point and the reference point
  - A search (or averaging) scheme is used to find a point close to the boundary of F

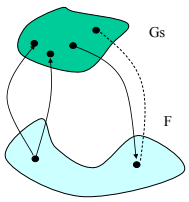
Blue in P_r
Black in P_s
Red repair



Feasible region F

Handling Constraints (3)

- Decoding functions
  - Map the genotype space (G_s) to feasible set
  - Thus every phenotype is guaranteed to produce a feasible solution
  - Not always possible depends on problem being solved
- Decoder function properties
  - Every g in G_s must map to a point s in F
  - Every solution s in F must have at least one representation g in G_s
  - Every s in F must have the same number of representations in G_s (not necessarily one)



Example : knapsack problem: Let genotype be binary string of objects to be taken. Scan left to right including an item if gene allele is one. Ignore rest of string as soon as a constraint is violated

Example : three colouring a graph

- Representation
  - Let the graph be G(N,E) where
    - N finite set of n>0 nodes
    - E set of edges
  - Genotype
    - Let G be a vector of length n
    - each element takes the values 1, 2, or 3
  - Adjacency matrix
    - Let C be an n by n matrix
    - C[i,j]=1 if an edge between nodes i and j in graph
    - C[i,j]=0 if no edge between nodes i and j
- FOP formulation
  - Define the penalty function as blow
  - Maximise 1/eval
  - Optimal answer is 1

```
Function eval(G:genotype): fitness;
{ penalty := 1; // to avoid divide by zero
  for i := 1 to n do
    for j := 1 to n do
      if C[i,j] <> 0 then
        if G[i] = G[j] then
          penalty := penalty +1 ;
  return eval = penalty;
}
```

Example : Sudoku grids

- Representation
  - Genotype
    - G is an n by n array
    - G[i,j] has takes a value 1..9
  - Suppose the existence of functions
    - Row(G, i) sums the values in row i of G
    - Col(G,j) sums the value in col j of G
    - Block(G,i,j) recovers and sums the block (G, i,j) on the grid
  - Assume a standard grid
    - n=9 and there are nine 3 by 3 blocks
    - Problem specific knowledge – a valid row, col, or block sums to 45

- Conversion to FOP
  - Penalty function (see below)
  - Maximise 1/eval
  - Optimal answer is 1;

```
Function eval(G:genotype): fitness;
{ penalty := 1; // to avoid divide by zero
  for i := 1 to n do
     if Row(G, i) <> 45 then penalty := penalty +1;
  for j := 1 to ndo
     if Col(G, j) <> 45 then penalty := penalty+1;
  for i:= 1 to 3 do
     for j := 1 to 3 do
        if  Block(G, i, j) then penalty := penalty +1 ;
  return eval = penalty;
}
```

Question: does this eval function always work? Can you suggest and alternative?

Parameter control

Chapter 8

Motivation 1

An EA has many strategy parameters, e.g.

- mutation operator and mutation rate
- crossover operator and crossover rate
- selection mechanism and selective pressure (e.g. tournament size)
- population size

Good parameter values facilitate good performance

Q1 How to find good parameter values ?

Motivation 2

EA parameters are rigid (constant during a run)

BUT

an EA is a dynamic, adaptive process

THUS

optimal parameter values may vary during a run

Q2: How to vary parameter values?

Parameter tuning

Parameter tuning: the traditional way of testing and comparing different values before the “real” run

Problems:

- users mistakes in settings can be sources of errors or sub-optimal performance
- costs much time
- parameters interact: exhaustive search is not practicable
- good values may become bad during the run

Parameter control

Parameter control: setting values on-line, during the actual run, e.g.

- predetermined time-varying schedule p = p(t)
- using feedback from the search process
- encoding parameters in chromosomes and rely on natural selection

Problems:

- finding optimal p is hard, finding optimal p(t) is harder
- still user-defined feedback mechanism, how to ``optimize"?
- when would natural selection work for strategy parameters?

Example

Task to solve:

- min $f(x_1,\ldots,x_n)$
- $L_i \le x_i \le U_i$ for i = 1,…,n bounds
- $g_i(x) \le 0$ for i = 1,…,q inequality constraints
- $h_i(x) = 0$ for i = q+1,…,m equality constraints

Algorithm:

- EA with real-valued representation $(x_1,\ldots,x_n)$
- arithmetic averaging crossover
- Gaussian mutation: $x'_i = x_i + N(0, \sigma)$
  standard deviation σ is called mutation step size

Varying mutation step size: option1

Replace the constant σ by a function $\sigma(t)$

$$\sigma(t) = 1 - 0.9 \times \frac{t}{T}$$

$0 \le t \le T$ is the current generation number

Features:

changes in σ are independent from the search progress
strong user control of σ by the above formula
σ is fully predictable
a given σ acts on all individuals of the population

Varying mutation step size: option2

Replace the constant σ by a function $\sigma(t)$ updated after every n steps by the 1/5 success rule (cf. ES chapter):

$$\sigma(t) = \begin{cases} \sigma(t-n)/c & \text{if } p_s > 1/5 \\ \sigma(t-n)\cdot c & \text{if } p_s < 1/5 \\ \sigma(t-n) & \text{otherwise} \end{cases}$$

Features:

changes in σ are based on feedback from the search progress
some user control of σ by the above formula
σ is not predictable
a given σ acts on all individuals of the population

Varying mutation step size: option3

Assign a personal σ to each individual
Incorporate this σ into the chromosome: (x_1, …, x_n, σ)
Apply variation operators to x_i's and σ

$$\sigma' = \sigma \times e^{N(0,\tau)}$$
$$x'_i = x_i + N(0,\sigma')$$

Features:
changes in σ are results of natural selection
(almost) no user control of σ
σ is not predictable
a given σ acts on one individual

Varying mutation step size: option4

Assign a personal σ to each variable in each individual
Incorporate σ's into the chromosomes: (x_1, …, x_n, σ_1, …, σ_n)
Apply variation operators to x_i's and σ_i's

$$\sigma'_i = \sigma_i \times e^{N(0,\tau)}$$
$$x'_i = x_i + N(0,\sigma'_i)$$

Features:
changes in σ_i are results of natural selection
(almost) no user control of σ_i
σ_i is not predictable
a given σ_i acts on 1 gene of one individual

Example cont'd

Constraints
- $g_i(x) \leq 0$ for i = 1,…,q inequality constraints
- $h_i(x) = 0$ for i = q+1,…,m equality constraints

are handled by penalties:

eval(x) = f(x) + W × penalty(x)

where $penalty(x) = \sum_{j=1}^{m} \begin{cases} 1 & \text{for violated constraint} \\ 0 & \text{for satisfied constraint} \end{cases}$

Varying penalty: option 1

Replace the constant W by a function W(t)

$$W(t) = (C \times t)^{\alpha}$$

$0 \leq t \leq T$ is the current generation number

Features:
changes in W are independent from the search progress
strong user control of W by the above formula
W is fully predictable
a given W acts on all individuals of the population

Varying penalty: option 2

Replace the constant W by W(t) updated in each generation

$$W(t+1) = \begin{cases} \beta \times W(t) & \text{if last } k \text{ champions all feasible} \\ \gamma \times W(t) & \text{if last } k \text{ champions all infeasible} \\ W(t) & \text{otherwise} \end{cases}$$

$\beta < 1$, $\gamma > 1$, $\beta \times \gamma \neq 1$ champion: best of its generation

Features:

changes in W are based on feedback from the search progress
some user control of W by the above formula
W is not predictable
a given W acts on all individuals of the population

Varying penalty: option 3

Assign a personal W to each individual
Incorporate this W into the chromosome: $(x_1, \ldots, x_n, W)$
Apply variation operators to x_i's and W

Alert:

$$eval((x, W)) = f(x) + W \times penalty(x)$$

while for mutation step sizes we had

$$eval((x, \sigma)) = f(x)$$

this option is thus sensitive "cheating" $\Rightarrow$ makes no sense

Lessons learned from examples

Various forms of parameter control can be distinguished by:

- primary features:
  - what component of the EA is changed
  - how the change is made
- secondary features:
  - evidence/data backing up changes
  - level/scope of change

What

Practically any EA component can be parameterized and thus controlled on-the-fly:

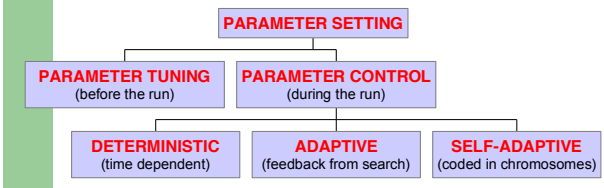
- representation
- evaluation function
- variation operators
- selection operator (parent or mating selection)
- replacement operator (survival or environmental selection)
- population (size, topology)

How

Three major types of parameter control:

- deterministic: some rule modifies strategy parameter without feedback from the search (based on some counter)
- adaptive: feedback rule based on some measure monitoring search progress
- self-adaptative: parameter values evolve along with solutions; encoded onto chromosomes they undergo variation and selection

Global taxonomy



Evidence informing the change

The parameter changes may be based on:

- time or nr. of evaluations (deterministic control)
- population statistics (adaptive control)
  - progress made
  - population diversity
  - gene distribution, etc.
- relative fitness of individuals creeated with given values (adaptive or self-adaptive control)

Evidence informing the change

- Absolute evidence: predefined event triggers change, e.g. increase p_m by 10% if population diversity falls under threshold x
- Direction and magnitude of change is fixed
- Relative evidence: compare values through solutions created with them, e.g. increase p_m if top quality offspring came by high mut. Rates
- Direction and magnitude of change is not fixed

Scope/level

The parameter may take effect on different levels:

- environment (fitness function)
- population
- individual
- sub-individual

Note: given component (parameter) determines possibilities
Thus: scope/level is a derived or secondary feature in the classification scheme

Refined taxonomy

- Combinations of types and evidences
  - Possible: +
  - Impossible: -

	Deterministic	Adaptive	Self-adaptive
Absolute	+	+	-
Relative	-	+	+

Evaluation / Summary

- Parameter control offers the possibility to use appropriate values in various stages of the search
- Adaptive and self-adaptive parameter control
  - offer users "liberation" from parameter tuning
  - delegate parameter setting task to the evolutionary process
  - the latter implies a double task for an EA: problem solving + self-calibrating (overhead)
- Robustness, insensivity of EA for variations assumed
  - If no. of parameters is increased by using (self)adaptation
  - For the "meta-parameters" introduced in methods

CoEvolution

- Definition
  - Interactive evolution
  - External influences
  - Can be more than one population
  - One population influences the other
  - Can deal with subjective fitness functions
- Motivation
  - In nature fitness of an organism depends on environmental niche
  - A species and the area seen on the solution space depends on interaction with other species
  - Types of interaction
    - Mutualism/symbiosis (positive species enhancement)
    - Predation/parasitism (negative effects of species)

Examples:

Ability of a rabbit to run at 30kph depends on the top speed of a fox that preys on it

Ability of a plant to pollinate depends bees to pick up and spread the pollen

Exercise: find out about the Iterated prisoner's dilemma used to evolve game-playing strategies

Cooperative CoEvolution

- Model
  - Set of different species
  - Each represent part of the problem
  - Must cooperate to solve a larger problem
- Computational Examples
  - Job shop scheduling
  - High-dimensional function optimisation
- Advantage
  - Allows function decomposition
  - But requires user to provide a partitioning of problem
- Real world example
  - Endosymbiosis where two species physically linked
  - e.g. gut bacteria that has to live inside hosts body.
- Computing
  - A parts of a problem cannot be separated.
  - Use linkage flags to show which solutions for different populations should stay together
- Genetic programming
  - Use the idea of automatically defined functions
  - GP can call functions which are themselves being evolved.

Competitive CoEvolution

- Model
  - Populations compete against each other
  - Grab fitness or parts of fitness landscape at the expense of each other
- Iterated prisoner's dilemma
  - Simplest scheme 2 players
  - Each decides whether to cooperate or defect
  - Reward is decided by a payoff matrix
  - Evolve a strategy where player can only see the last three strategies of opponent
  - Best strategy Tit-for-Tat
- Hillis Sorting networks
  - Used two species (population)
  - Task to evolve the best sorting networks
  - Fitness determined by how many examples were sorted correctly
  - Found better networks than previously known
- Pairing
  - As population's evolve can pair up to cooperate
  - If competition occurs in single pop use fitness and ranking to choose members
  - Between populations requires competitive fitness evaluation

CoEvolution in constraint problems

- Here have a number of constraints
  - Each constraint must be satisfied to get high fitness
- Two populations
  - Can measure fitness of one by ability to solve problem
  - Can measure second by its ability to frustrate the first population
- Choice of populations
  - Pop 1: individuals try to satisfy the problem
  - Pop 2: are constraints
- Frustration
  - A candidate s in pop1 frustrates a candidate c in pop2 if it satisfies the constraint represented by c
  - A candidate c if pop 2 frustrates a candidate s in pop 1 if constraint c makes s fail
- Mutual frustration
  - Taking repeated cycles evolve pop1 which has a high fitness in frustrating pop2 (i.e satifies constraints)
  - Pop2 remains static – cannot evolve constraints (usually know at design-time)

Interactive Evolution

- Method
  - User becomes part of evolutionary system
  - Usually in determining the fitness of individuals
- Example: agricultural breeding
  - Selective breeding by man
  - Creates better cattle, faster horses, different types of dogs
- User's influence
  - Subjective (user chooses best)
  - Aesthetic (used in context of evolutionary art)
- Application
  - Situations where no clear fitness function exists
  - To improve search ability, when method gets stuck in area of search space
  - To increase exploration and diversity of population
- Disadvantages
  - Slow compared with automated execution
  - Inconsistency: humans change their mind as to what is 'best'
  - Limited coverage: humans only concentrate for a limited time or for small populations

Hybrid Evolutionary Algorithms

Chapter 10

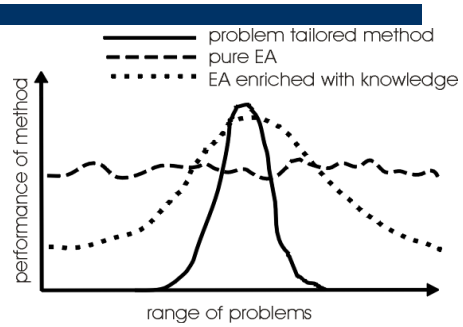
Overview

- Why to Hybridise
- Where to hybridise
- Incorporating good solutions
- Local Search and graphs
- Lamarkian vs. Baldwinian adaptation
- Diversity
- Operator choice

Why Hybridise

- Might want to put in EA as part of larger system
- Might be looking to improve on existing techniques but not re-invent wheel
- Might be looking to improve EA search for good solutions

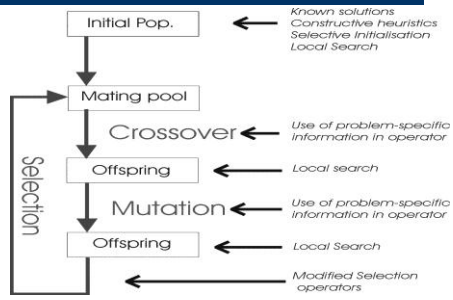
Michalewicz's view on EAs in context



Memetic Algorithms

- The combination of Evolutionary Algorithms with Local Search Operators that work within the EA loop has been termed "Memetic Algorithms"
- Term also applies to EAs that use instance specific knowledge in operators
- Memetic Algorithms have been shown to be orders of magnitude faster and more accurate than EAs on some problems, and are the "state of the art" on many problems

Where to Hybridise



Heuristics for Initialising Population

- Bramlette ran experiments with limited time scale and suggested holding a *n*-way tournament amongst randomly created solutions to pick initial population
  (n.b. NOT the same as taking the best *popsize* of *n.popsize* random points)
- Multi-Start Local Search is another option: pick *popsize* points at random to climb from
- Constructive Heuristics often exist

Initialisation Issues

- Another common approach would be to initialise population with solutions already known, or found by another technique (beware, performance may appear to drop at first if local optima on different landscapes do not coincide)
- Surry & Radcliffe (1994) studied ways of "inoculating" population with solutions gained from previous runs or other algorithms/heuristics
  - found ***mean*** performance increased as population was biased towards know solutions,
  - but ***best*** performance came from more random solutions

"Intelligent" Operators

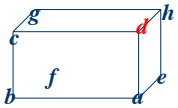
- It is sometimes possible to incorporate problem or instance specific knowledge within crossover or mutation operators
  - E.g. Merz's DPX operator for TSP inherits common sub tours from parents then connects them using a nearesr neighbour heuristic
  - Smith (97) evolving microprocessor instruction sequences: group instructions (alleles) into classes so mutation is more likely to switch gene to value having a similar effect
  - Many other examples in literature

Local Search acting on offspring

- Can be viewed as a sort of "lifetime learning"
- Lots of early research done using EAs to evolve the structure of Artificial Neural Networks and then Back-propagation to learn connection weights
- Often used to speed-up the "endgame" of an EA by making the search in the vicinity of good solutions more systematic than mutation alone

Local Search

- Defined by combination of ***neighbourhood*** and ***pivot rule***
- Related to landscape metaphor
- *N(x)* is defined as the set of points that can be reached from *x* with one application of a move operator
  - e.g. bit flipping search on binary problems



$N(d) = \{a,c,h\}$

Landscapes & Graphs

- The combination of representation and operator defines a graph *G(v,E)* on the search space. (useful for analysis)
- *v*, the set of vertices, is the set of all points that can be represented (the potential solutions)
- *E*, the set of edges, is the possible transitions that can arise from a single application of the operator
  - note that the edges in *E* can have weights attached to them, and that they need not be symmetrical

Example Graphs for binary

- example : binary problem as above
  - $v = \{a,b,c,d,e,f,g,h,\}$
  - Search by flipping each bit in turn
    - $E_1 = \{ ab, ad, ae, bc, bf, cd, cg, dh, fg, fe, gh, eh\}$
    - symmetrical and all values equally likely
  - Bit flipping mutation with prob p per bit
  - $E = p.E_1 \cup \ p^2\{ac,bd,af,be,dg, ch, fh, ge, ah, de, bg, cf\} \cup p^3 \{ag, bh, ce, df\}$

Graphs

- The ***Degree*** of a graph is the maximum number of edges coming into/out of a single point, - the size of the biggest neighbourhood
  - single bit changing search: degree is l
  - bit-wise mutation on binary: degree is 2^l -1
  - 2-opt: degree is $O(N^2)$
- Local Search algorithms look at points in the neighbourhood of a solution, so complexity is related to degree of graph

Pivot Rules

- Is the neighbourhood searched randomly, systematically or exhaustively ?
- does the search stop as soon as a fitter neighbour is found (***Greedy Ascent***)
- or is the whole set of neighbours examined and the best chosen (***Steepest Ascent***)
- of course there is no one best answer, but some are quicker than others to run .....

Variations of Local Search

- Does the search happen in representation space or Solution Space ?
- How many iterations of the local search are done ?
- Is local search applied to the whole population?
  - or just the best ?
  - or just the worst ?
  - see work (PhD theses) by Hart (www.cs.sandia.gov/~wehart), and Land

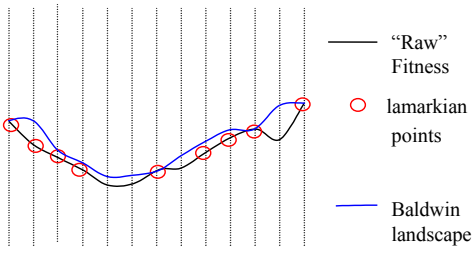
Two Models of Lifetime Adaptation

- Lamarkian
  - traits acquired by an individual during its lifetime can be transmitted to its offspring
  - e.g. replace individual with fitter neighbour
- Baldwinian
  - traits acquired by individual cannot be transmitted to its offspring
  - e.g. individual receives fitness (but not genotype) of fitter neighbour

The Baldwin effect

- LOTS of work has been done on this
  - the central dogma of genetics is that traits acquired during an organisms lifetime ***cannot*** be written back into its gametes
  - e.g. Hinton & Nowlan '87, ECJ special issue etc
- In MAs we are not constrained by biological realities so can do lamarkianism

Induced landscapes



Information Use in Local Search

- Most Memetic Algorithms use an operator acting on a single point, and only use that information
- However this is an arbitrary restriction
  - Jones (1995), Merz & Friesleben (1996) suggest the use of a crossover hillclimber which uses information from two points in the search space
  - Krasnogor & Smith (2000) - see later - use information from whole of current population to govern acceptance of inferior moves
  - Could use Tabu search with a common list

Diversity

- Maintenance of diversity within the population can be a problem, and some successful algorithms explicitly use mechanisms to preserve diversity:
  - Merz's DPX crossover explicitly generates individuals at same distance to each parent as they are apart
  - Krasnogor's Adaptive Boltzmann Operator uses a Simulated-Annealing like acceptance criteria where "temperature" is inversely proportional to population diversity

Boltzman MAs: acceptance criteria

- Assuming a maximisation problem,
  Let Δf = fitness of neighbour – current fitness

$$P(\textit{accepting neighbour}) = \begin{cases} 1 & \Delta f > 0 \\ e^{\frac{k\Delta f}{f_{\max} - f_{avg}}} & \Delta f < 0 \end{cases}$$

Boltzmann MAs:2

- Induced dynamic is such that:
  - Population is diverse => spread of fitness is large, therefore *temperature* is low, so only accept improving moves => *Exploitation*
  - Population is converged => temperature is high, more likely to accept worse moves => *Exploration*
- Krasnogor showed this improved final fitness and preserved diversity longer on a range of TSP and Protein Structure Prediction problems

Choice of Operators

- Krasnogor (2002) will show that there are theoretical advantages to using a local search with a move operator that is DIFFERENT to the move operators used by mutation and crossover
- Can be helpful since local optima on one landscape might be point on a slope on another
- Easy implementation is to use a range of local search operators, with mechanism for choosing which to use. (Similar to Variable Neighbourhood Search)
- This could be learned & adapted on-line (e.g. Krasnogor & Smith 2001)

Hybrid Algorithms Summary

- It is common practice to hybridise EA's when using them in a real world context.
- this may involve the use of operators from other algorithms which have already been used on the problem (e.g. 2-opt for TSP), or the incorporation of domain-specific knowledge (e.g PSP operators)
- Memetic algorithms have been shown to be orders of magnitude faster and more accurate than GAs on some problems, and are the "state of the art" on many problems