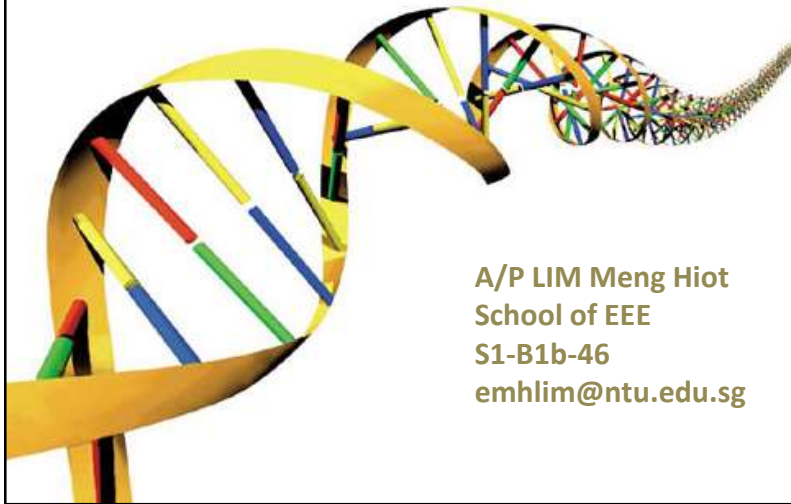


EE6407: Genetic Algorithms and Machine Learning



A/P LIM Meng Hiot
School of EEE
S1-B1b-46
emhlim@ntu.edu.sg

1

Assessment

- Continuous Assessment – 40%
- Final Exam (3-hour) – 60%

Continuous Assessment (Part 1 – 3 Weeks):

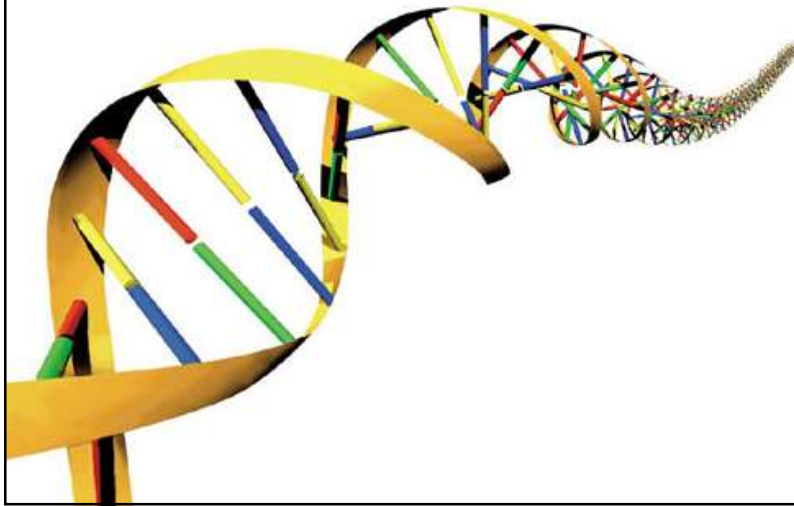
Quiz – 10%

When? – Last hour of Week 3

2

2

Evolutionary Computing



3

Problems to be solved

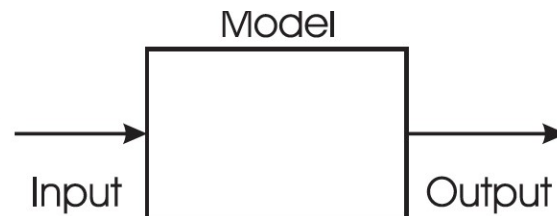
Problems can be classified in different ways:

- Black box model
- Search problems
- Optimization vs constraint satisfaction
- NP problems

4

4

“Black box” model



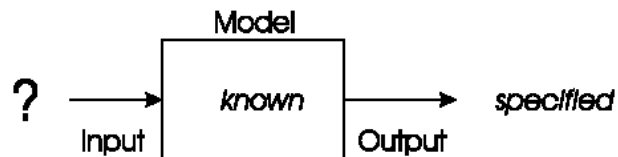
- “Black box” model consists of 3 components
- When one component is unknown: new problem type

5

5

“Black box” model: Optimisation

- Model and desired output is known, task is to find inputs



- Examples:
 - Timetables for university, call center, or hospital
 - Design specifications
 - Traveling salesman problem (TSP)
 - Eight-queens problem, etc.

6

6

“Black box” model: 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

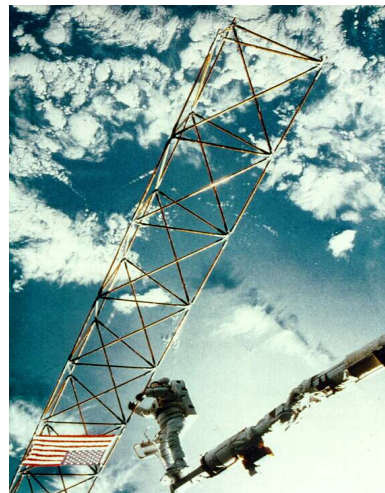


7

7

“Black box” model: Optimisation example 2: satellite structure

- Optimised satellite designs for NASA to maximize vibration isolation
- Evolving: design structures
- Fitness: vibration resistance
- Evolutionary “creativity”

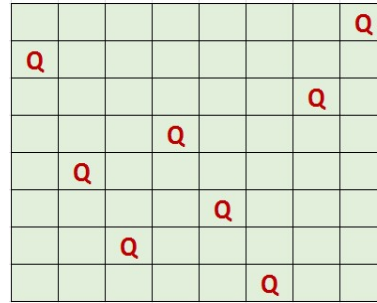


8

8

“Black box” model: Optimisation example 3: 8 queens problem

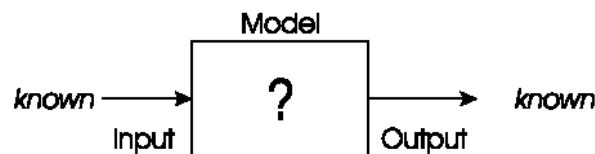
- Given an 8-by-8 chessboard and 8 queens
- Place the 8 queens on the chessboard without any conflict
- Two queens in conflict if they share same row, column or diagonal
- Can be extended to an n queens problem ($n > 8$)



9

“Black box” model: Modelling

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



- Note: modelling problems can be transformed into optimisation problems
 - Evolutionary machine learning
 - Predicting stock exchange
 - Voice control system for smart homes

10

“Black box” model: Modelling example: loan applicant creditability

- Creditability model to predict loan paying behavior of new applicants
- Evolving: prediction models
- Fitness: model accuracy on historical data

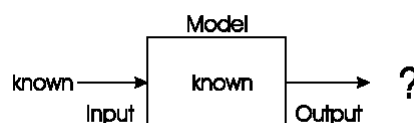


11

11

“Black box” model: Simulation

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

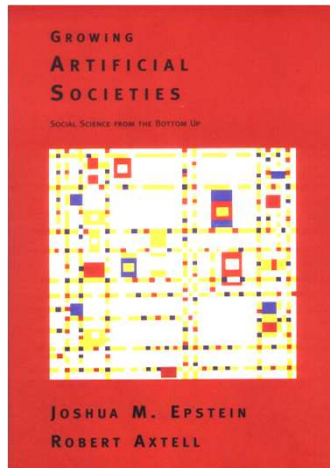


- Often used to answer “what-if” questions in evolving dynamic environments
 - Evolutionary economics, Artificial Life
 - Weather forecast system
 - Impact analysis new tax systems

12

12

“Black box” model: Simulation example: evolving artificial societies



Simulating trade, economic competition, etc. to **calibrate models**

Use models to optimise strategies and policies

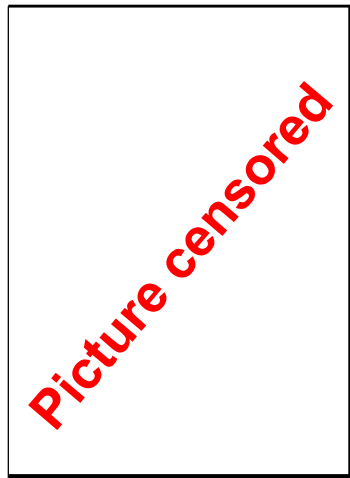
Evolutionary economy

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

11

13

“Black box” model: Simulation example 2: biological interpretations



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

14

14

Search problems

- Simulation is different from optimisation/modelling
- Optimisation/modelling problems search through huge space of possibilities
- Search space: collection of all objects of interest including the desired solution
- **Question:** how large is the search space for different tours through n cities?

Benefit of classifying these problems: distinction between

- search problems, which define search spaces, and
- problem-solvers, which tell how to move through search spaces.

15

15

Optimisation vs. constraint satisfaction (1/2)

- Objective function: a way of assigning a value to a possible solution that reflects its quality on scale
 - Number of un-checked queens (maximize)
 - Length of a tour visiting given set of cities (minimize)
- Constraint: binary evaluation telling whether a given requirement holds or not
 - Find a configuration of eight queens on a chessboard such that no two queens check each other
 - Find a tour with minimal length where city X is visited after city Y

16

16

Optimisation vs. constraint satisfaction (2/2)

- When combining the two:

Constraints	Objective function	
	Yes	No
Yes	Constrained optimisation problem	Constraint satisfaction problem
No	Free optimisation problem	No problem

- Where do the examples fit?
- Note: constraint problems can be transformed into optimisation problems
- Question: how can we formulate the 8-queens problem into a FOP/CSP/COP?

17

17

NP problems

- We only looked at classifying the problem, not discussed problem solvers
- This classification scheme needs the properties of the problem solver
- Benefit of this scheme: possible to tell how difficult the problem is
- Explain the basics of this classifier for combinatorial optimization problems (Booleans or integers search space)

18

18

NP problems: Key notions

- **Problem size**: dimensionality of the problem at hand and number of different values for the problem variables
- **Running-time**: number of operations the algorithm takes to terminate
 - Worst-case as a function of problem size
 - Polynomial, super-polynomial, exponential
- **Problem reduction**: transforming current problem into another via mapping

19

19

NP problems: Class

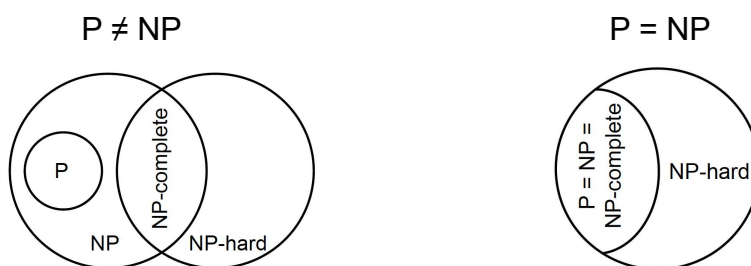
- The difficultness of a problem can now be classified:
 - **Class P**: algorithm can solve the problem in polynomial time (worst-case running-time for problem size n is less than $F(n)$ for some polynomial formula F)
 - **Class NP**: problem can be solved, and any solution can be verified within polynomial time by some other algorithm (P subset of NP)
 - **Class NP-complete**: problem belongs to class NP and any other problem in NP can be reduced to this problem by an algorithm running in polynomial time
 - **Class NP-hard**: problem is at least as hard as any other problem in NP -complete but solution cannot necessarily be verified within polynomial time

20

20

NP problems: Difference between classes

- P is different from NP-hard
- Not known whether P is different from NP



- For now: use of approximation algorithms and metaheuristics

21

21

Computers, Complexity, and Intractability

One day your boss calls you into his office and confides that the company is about to enter the highly competitive “bandersnatch” market. For this reason, a good method is needed for determining whether or not any given set of specifications for a new bandersnatch component can be met and, if so, for constructing a design that meets them. Since you are the company’s chief algorithm designer, your charge is to find an efficient algorithm for doing this.

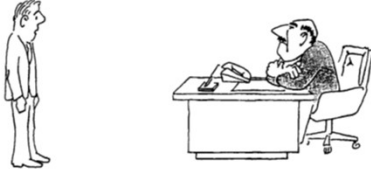
After consulting with the bandersnatch department to determine exactly what the problem is, you eagerly hurry back to your office, pull down your reference books, and plunge into the task with great enthusiasm. Some weeks later, your office filled with mountains of crumpled-up scratch paper, your enthusiasm has lessened considerably. So far you have not been able to come up with any algorithm substantially better than searching through all possible designs. This would not particularly endear you to your boss, since it would involve years of computation time for just one set of

22

22


2 COMPUTERS, COMPLEXITY, AND INTRACTABILITY

specifications, and the bandersnatch department is already 13 components behind schedule. You certainly don't want to return to his office and report:



"I can't find an efficient algorithm, I guess I'm just too dumb."

To avoid serious damage to your position within the company, it would be much better if you could prove that the bandersnatch problem is *inherently* intractable, that no algorithm could possibly solve it quickly. You then could stride confidently into the boss's office and proclaim:




"I can't find an efficient algorithm, because no such algorithm is possible!"

Unfortunately, proving inherent intractability can be just as hard as finding efficient algorithms. Even the best theoreticians have been stymied in their attempts to obtain such proofs for commonly encountered hard problems. However, having read this book, you have discovered something

1.1 INTRODUCTION 3

almost as good. The theory of NP-completeness provides many straightforward techniques for proving that a given problem is "just as hard" as a large number of other problems that are widely recognized as being difficult and that have been confounding the experts for years. Armed with these techniques, you might be able to prove that the bandersnatch problem is NP-complete and, hence, that it is equivalent to all these other hard problems. Then you could march into your boss's office and announce:



"I can't find an efficient algorithm, but neither can all these famous people."

At the very least, this would inform your boss that it would do no good to fire you and hire another expert on algorithms.

Of course, our own bosses would frown upon our writing this book if its sole purpose was to protect the jobs of algorithm designers. Indeed, discovering that a problem is NP-complete is usually just the beginning of work on that problem. The needs of the bandersnatch department won't disappear overnight simply because their problem is known to be NP-complete. However, the knowledge that it is NP-complete does provide valuable information about what lines of approach have the potential of being most productive. Certainly the search for an efficient, exact algorithm should be accorded low priority. It is now more appropriate to concentrate on other, less ambitious, approaches. For example, you might look for efficient algorithms that solve various special cases of the general problem. You might look for algorithms that, though not guaranteed to run quickly, seem likely to do so most of the time. Or you might even relax the problem somewhat, looking for a fast algorithm that merely finds designs that

23

Evolutionary Computing: the Origins

- Historical perspective
- Biological inspiration:
 - Darwinian evolution theory (simplified!)
 - Genetics (simplified!)
- Motivation for EC

24

24

Historical perspective

- 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

25

25

Darwinian Evolution (1/3): Survival of the fittest

Evolutionary Computing: the Origins

- Historical perspective
- Biological inspiration:
 - Darwinian evolution theory (simplified)
 - Genetics (simplified)
- Motivation for EC

- All environments have finite resources
(i.e., can only support a limited number of individuals)
- Life forms 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

26

26

Darwinian Evolution (2/3): 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 ...

27

27

Darwinian Evolution (3/3): 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”

28

28

Genetics: Natural

Evolutionary Computing: the Origins

- Historical perspective
- Biological inspiration:
 - Darwinian evolution theory (simplified!)
 - Genetics (simplified!)
- Motivation for EC

- 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)

29

29

Genetics: 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

30

30

Genetics: 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 is 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

31

31

Recap of EC metaphor (1/2)

Evolutionary Computing: the Origins

- Historical perspective
- Biological inspiration:
 - Darwinian evolution theory (simplified!)
 - Genetics (simplified!)
- Motivation for EC

- 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

32

32

Recap of EC metaphor (2/2)

- 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

33

33

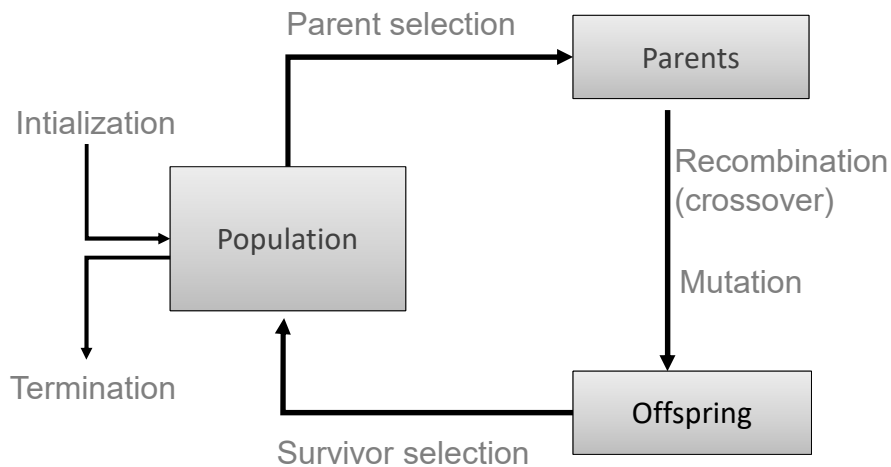
What is an Evolutionary Algorithm?

- Scheme of an EA
- Main EA components:
 - Representation / evaluation / population
 - Parent selection / survivor selection
 - Recombination / mutation
- Examples: eight-queens problem
- Typical EA behaviour
- EAs and global optimisation
- EC and neighbourhood search

34

34

Scheme of an EA: General scheme of EAs



35

35

Scheme of an EA: EA scheme in pseudo-code

```

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
  
```

36

36

Scheme of an EA: Common model of evolutionary processes

- **Population** of individuals
- Individuals have a **fitness**
- **Variation** operators: crossover, mutation
- **Selection** towards higher fitness
 - “survival of the fittest” and
 - “mating of the fittest”

Neo Darwinism:

Evolutionary progress towards higher life forms

=

Optimization according to some fitness-criterion
(optimization on a fitness landscape)

37

37

Scheme of an EA: Two pillars of evolution

There are two competing forces

Increasing population **diversity**
by genetic operators

- mutation
- recombination

Push towards **novelty**

Decreasing population **diversity** by
selection

- of parents
- of survivors

Push towards **quality**

38

38

Main EA components: Representation (1/2)

What is an Evolutionary Algorithm?

- Scheme of an EA
- Main EA components:
 - Representation / evaluation / population
 - Parent selection / survivor selection
 - Recombination / mutation
- Examples: eight-queens problem
- Typical EA behaviour
- EAs and global optimisation
- EC and neighbourhood search

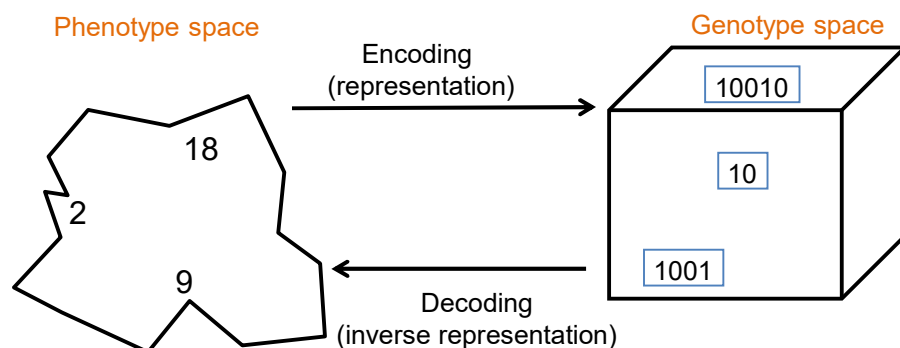
- Role: provides code for candidate solutions that can be manipulated by variation operators
- Leads to two levels of existence
 - **phenotype**: **object** in original problem context, the outside
 - **genotype**: **code** to denote that object, the inside (chromosome, “digital DNA”)
- Implies two mappings:
 - 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**)

39

39

Main EA components: Representation (2/2)

Example: represent integer values by their binary code



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

40

40

Main EA components: Evaluation (fitness) function

What is an Evolutionary Algorithm?

- Scheme of an EA
- Main EA components:
 - Representation / evaluation / population
 - Parent selection / survivor selection
 - Recombination / mutation
- Examples: eight-queens problem
- Typical EA behaviour
- EAs and global optimisation
- EC and neighbourhood search

- Role:
 - Represents the task to solve, the requirements to adapt to (can be seen as “the environment”)
 - Enables selection (provides basis for comparison)
 - e.g., some phenotypic traits are advantageous, desirable, e.g. big ears cool better, these traits are rewarded by more offspring that will expectedly carry the same trait
- 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

41

41

Main EA components: Population (1/2)

What is an Evolutionary Algorithm?

- Scheme of an EA
- Main EA components:
 - Representation / evaluation / population
 - Parent selection / survivor selection
 - Recombination / mutation
- Examples: eight-queens problem
- Typical EA behaviour
- EAs and global optimisation
- EC and neighbourhood search

- Role: holds the candidate solutions of the problem as individuals (genotypes)
- Formally, a population is a multiset of individuals, i.e. repetitions are possible
- Population is the basic unit of evolution, i.e., the population is evolving, not the individuals
- Selection operators act on population level
- Variation operators act on individual level

42

42

Main EA components: Population (2/2)

- 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)

43

43

Main EA components: Selection mechanism (1/3)

Role:

- Identifies individuals
 - to become parents
 - to survive
- Pushes population towards higher fitness
- Usually probabilistic
 - high quality solutions more likely to be selected than low quality
 - but not guaranteed
 - even worst in current population usually has non-zero probability of being selected
- This *stochastic* nature can aid escape from local optima

What is an Evolutionary Algorithm?

- Scheme of an EA
- Main EA components:
 - Representation / evaluation / population
 - Parent selection / survivor selection
 - Recombination / mutation
- Examples: eight-queens problem
- Typical EA behaviour
- EAs and global optimisation
- EC and neighbourhood search

44

44

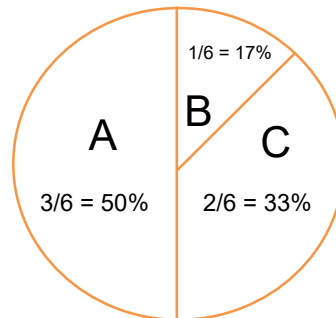
Main EA components: Selection mechanism (2/3)

Example: roulette wheel selection

fitness(A) = 3

fitness(B) = 1

fitness(C) = 2



In principle, any selection mechanism can be used for parent selection as well as for survivor selection

45

45

Main EA components: Selection mechanism (3/3)

- 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 (while parent selection is usually stochastic)
 - Fitness based : e.g., rank parents + offspring and take best
 - Age based: make as many offspring as parents and delete all parents
- Sometimes a combination of stochastic and deterministic (elitism)

46

46

Main EA components: Variation operators

What is an Evolutionary Algorithm?

- Scheme of an EA
- Main EA components:
 - Representation / evaluation / population
 - Parent selection / survivor selection
 - Recombination / mutation
- Examples: eight-queens problem
- Typical EA behaviour
- EAs and global optimisation
- EC and neighbourhood search

- Role: 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**
 - Arity > 2 is formally possible, seldom used in EC
- There has been much debate about relative importance of recombination and mutation
 - Nowadays most EAs use both
 - Variation operators must match the given representation

47

47

Main EA components: Mutation (1/2)

- Role: causes small, random variance
- 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 historical 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

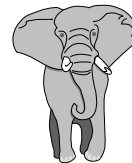
48

48

Main EA components: Mutation (2/2)

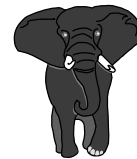
before

1 1 1 1 1 1 1



after

1 1 1 0 1 1 1



49

49

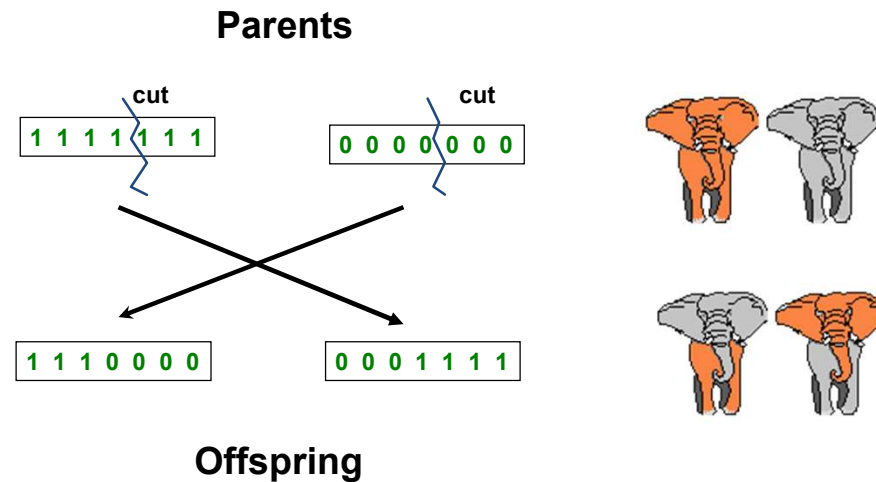
Main EA components: Recombination (1/2)

- Role: 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

50

50

Main EA components: Recombination (2/2)



51

51

Main EA components: Initialisation / Termination

What is an Evolutionary Algorithm?

- Scheme of an EA
- Main EA components:
 - Representation / evaluation / population
 - Parent selection / survivor selection
 - Recombination / mutation
- Examples: eight-queens problem
- Typical EA behaviour
- EAs and global optimisation
- EC and neighbourhood search

- 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

52

52

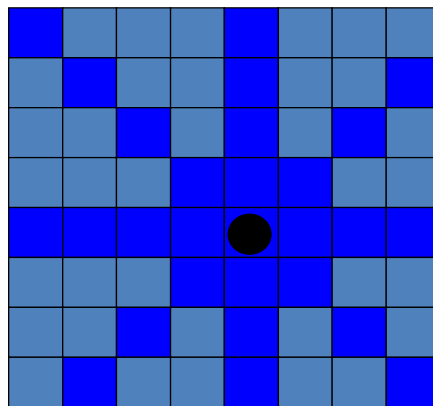
Main EA components: What are the different types of EAs

- Historically different flavours of EAs have been associated with different data types to represent solutions
 - 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

53

53

Example: The 8-queens problem



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

54

54

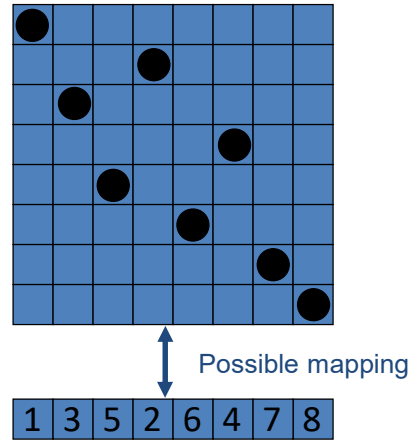
What is an Evolutionary Algorithm?

- Scheme of an EA
- Main EA components:
 - Representation / evaluation / population
 - Parent selection / survivor selection
 - Recombination / mutation
- Examples: eight-queens problem
- Typical EA behaviour
- EAs and global optimisation
- EC and neighbourhood search

The 8-queens problem: Representation

Phenotype:
a board configuration

Genotype:
a permutation of
the numbers 1–8



55

55

The 8-queens problem: Fitness evaluation

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

56

56

The 8-queens problem: Mutation

Small variation in one permutation, e.g.:

- swapping values of two randomly chosen positions,



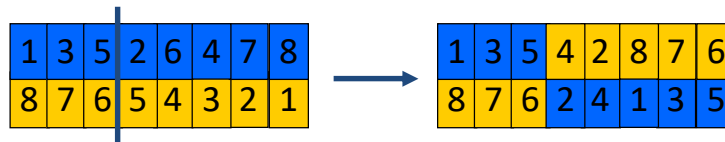
57

57

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



58

58

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

59

59

The 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 **only one possible**
set of choices of operators and parameters

60

60

SGA Example: $f(x)=x^2$

What is an Evolutionary Algorithm?

- Scheme of an EA
- Main EA components:
 - Representation / evaluation / population
 - Parent selection / survivor selection
 - Recombination / mutation
- Examples: eight-queens problem
- Typical EA behaviour
- EAs and global optimisation
- EC and neighbourhood search

- Consider a problem of maximizing $f(x)=x^2$.
- Assume x is an integer between 0 and 31.
- Go through with more details and highlight the 6 major steps.

61

61

SGA – Steps

Step 1 (Encoding)

- In this simple problem, we have just one integer variable restricted between 0 and 31.
- One obvious encoding is to use 5 bits.

Step 2 (Initial population)

- The initial population is generated randomly.

Step 3 (Decoding)

- The strings are converted to integers and $f(x)$ are evaluated.
- The (initial) population, decoded decision variable x , fitness $f(x)$, fitness proportions are all shown in Table 1.2.

Step 4 (Computing fitness proportions)

- We can use the roulette wheel to obtain the mating pool (shown in column 7 in Table 1.2)

62

62

SGA – Steps...

Step 5 (Crossover)

- The 1st step is to select $n/2$ pairs of parent strings for crossover. The randomly chosen pairs are shown in column 8 under “Mate”.
- Then the one-point crossover sites are randomly chosen between 1 and 4. It is shown in column 9 as “C’over site”.
- The result of crossover operation is given in column 10, as “New Popn”.

Step 6

- Here we assume that the mutation probability is 0.001. There are 20 bits in total in the population. Hence, 0.02 bits should mutate.
- Hence, no bit mutate, and the final result (next generation population) is shown in column 10.

63

63

Evolution & Termination of the SGA

- In general, steps 1 & 2 performed just once in the beginning of the SGA.
- Steps 3, 4, 5 and 6 are repeated until a termination condition is satisfied.
- Possible termination conditions may be
 - A given number of generations (or a number of fitness evaluations) has been completed
 - The fitness (i.e. the objective value of the optimization problem) of the best solution or the pool has reached an acceptable level.
 - The sequential increase in the fitness is negligible over several generations.
 - A combination of the above conditions.

64

64

1	2	3	4	5	6	7	8	9	10	11	12
String No.	Initial Popn.	x	$f(x)$	% of Total	No. Sel.	Mating Pool	Mate	C'over Site	New Popn.	x	$f(x)$
1	01101	13	169	14.4	1	01101	2	4	01100	12	144
2	11000	24	576	49.2	2	11000	1	4	11001	25	625
3	01000	8	64	5.5	0	11000	4	2	11011	27	729
4	10011	19	361	30.9	1	10011	3	2	10000	16	256
Sum			1170	100.0	4						1754
Average			293								439

Table 1.2 Simulation of a simple GA

65

65

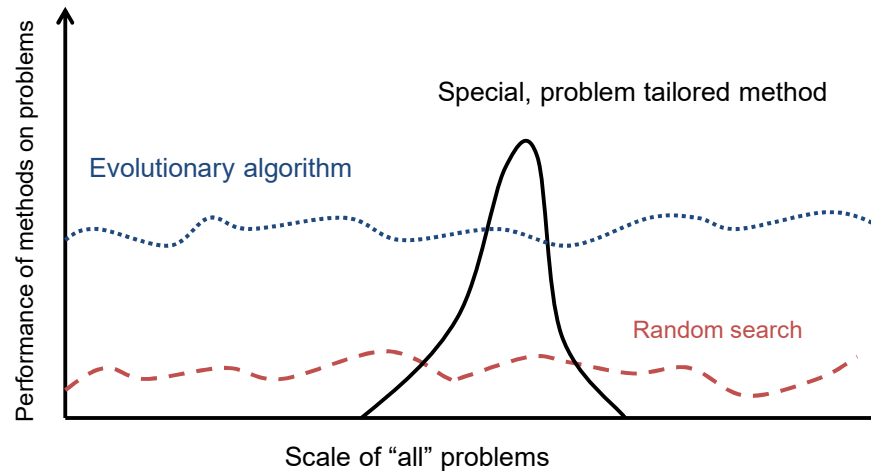
Typical EA behaviour: Evolutionary Algorithms in context

- 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

66

66

Typical EA behaviour: EAs as problem solvers: Goldberg view (1989)



67

67

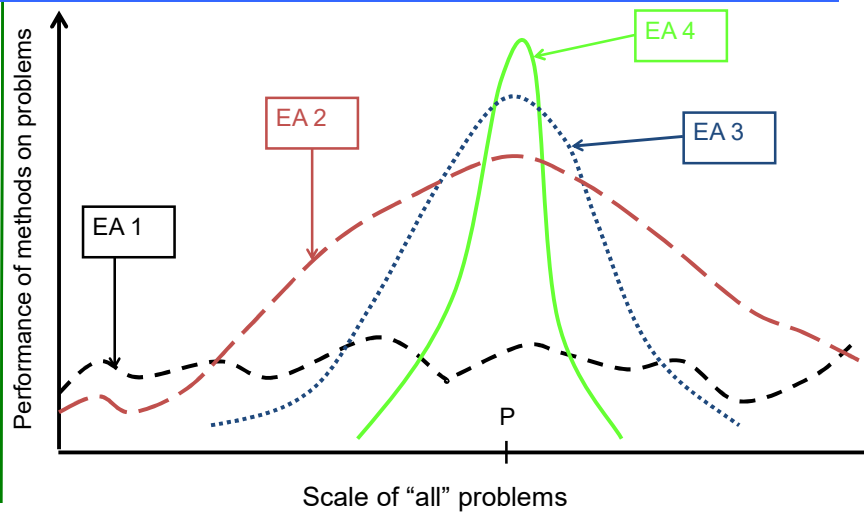
Typical EA behaviour: EAs and domain knowledge

- 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

68

68

Typical EA behaviour: EAs as problem solvers: Michalewicz view (1996)



69

69

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^* ,
 - May have bounds on runtime, usually super-polynomial
- Heuristic Approaches (generate and test)
 - rules for deciding which $x \in S$ to generate next
 - no guarantees that best solutions found are globally optimal
 - no bounds on runtime
- “I don’t care if it works as long as it converges”
vs.
- “I don’t care if it converges as long as it works”

70

70

EC and neighbourhood search

What is an Evolutionary Algorithm?

- Scheme of an EA
- Main EA components:
 - Representation / evaluation / population
 - Parent selection / survivor selection
 - Recombination / mutation
- Examples: eight-queens problem
- Typical EA behaviour
- EAs and global optimisation
- EC and neighbourhood search

- Many heuristics impose a neighbourhood structure on S
- Such heuristics may guarantee that best point found is *locally optimal* e.g. Hill-Climbers:
 - **But** problems often exhibit many local optima
 - Often very quick to identify good solutions
- EAs are distinguished by:
 - Use of population,
 - Use of multiple, stochastic search operators
 - Especially variation operators with arity >1
 - Stochastic selection
- Question: what is the neighbourhood in an EA?

71

71

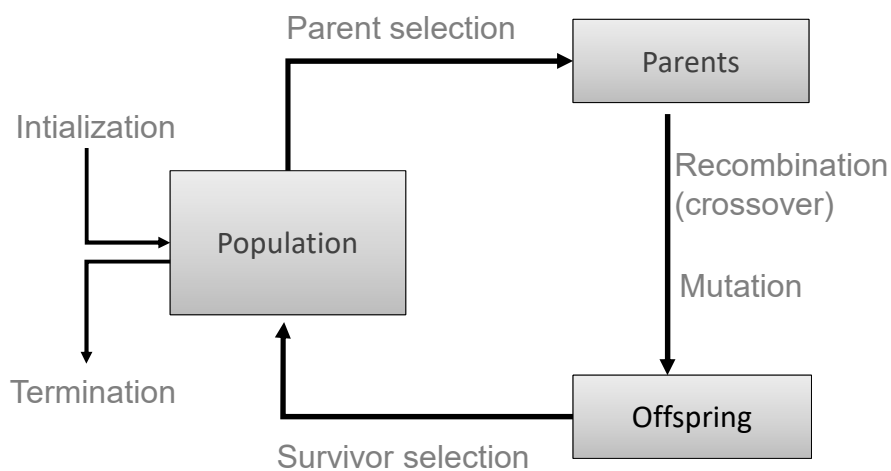
Representation, Mutation, and Recombination

- Role of representation and variation operators
- Most common representation of genomes:
 - Binary
 - Integer
 - Real-Valued or Floating-Point
 - Permutation
 - Tree

72

72

Scheme of an EA: General scheme of EAs



73

73

Role of representation and variation operators

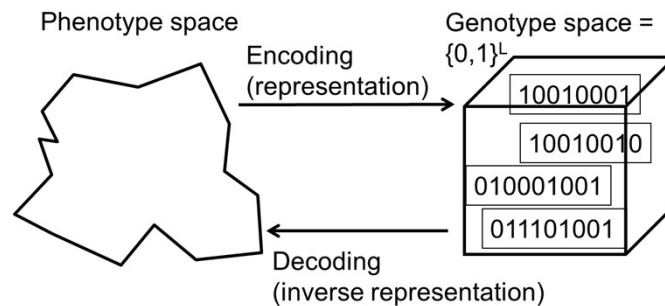
- First stage of building an EA and most difficult one: choose *right* representation for the problem
- Variation operators: mutation and crossover
- Type of variation operators needed depends on chosen representation
- TSP problem
 - What are possible representations?

74

74

Binary Representation

- One of the earliest representations
- Genotype consists of a string of binary digits



75

75

Binary Representation: Mutation

- Alter each gene independently with a probability p_m
- p_m is called the mutation rate
 - Typically between $1/\text{pop_size}$ and $1/\text{chromosome_length}$



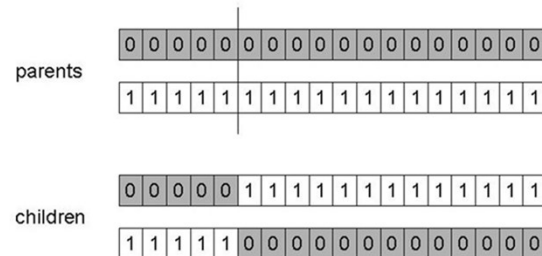
- Mutation can cause variable effect (use gray coding)

76

76

Binary Representation: 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)



77

77

Binary Representation: Alternative Crossover Operators

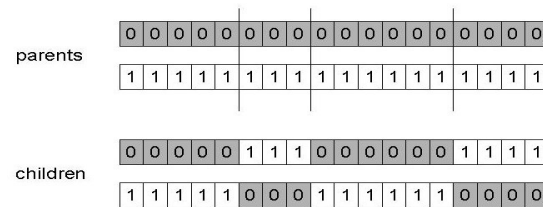
- Why do we need other crossover(s)?
- Performance with 1-point crossover depends on the order that variables occur in the representation
 - More likely to keep together genes that are near each other
 - Can never keep together genes from opposite ends of string
 - This is known as Positional Bias
 - Can be exploited if we know about the structure of our problem, but this is not usually the case

78

78

Binary Representation: 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)

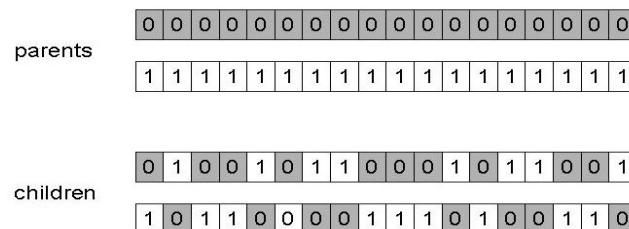


79

79

Binary Representation: 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



80

80

Binary Representation: Crossover OR mutation? (1/3)

- Decade long debate: which one is better / necessary / main-background
- Answer (at least, rather wide agreement):
 - it depends on the problem, but
 - in general, it is good to have both
 - both have another role
 - mutation-only-EA is possible, crossover-only-EA would not work

81

81

Binary Representation: Crossover OR mutation? (2/3)

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

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

There is co-operation AND competition between them

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

82

82

Binary Representation: Crossover OR mutation? (3/3)

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

83

83

Integer Representation

- Nowadays it is generally accepted that it is better to encode numerical variables directly (integers, floating point variables)
- 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
 - Adding a small (positive or negative) value to each gene with probability p
 - Random resetting (esp. categorical variables)
 - With probability p_m a new value is chosen at random
- Same recombination as for binary representation

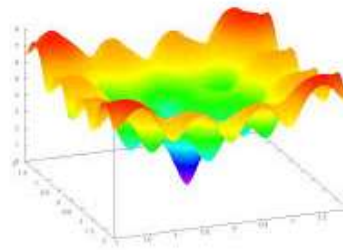
84

84

Real-Valued or Floating-Point Representation

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

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



85

85

Real-Valued or Floating-Point Representation: Mapping real values on bit strings

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

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

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

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

86

86

Real-Valued or Floating-Point Representation: Uniform Mutation

- General scheme of floating point mutations

$$\bar{x} = \langle x_1, \dots, x_l \rangle \rightarrow \bar{x}' = \langle x'_1, \dots, x'_l \rangle$$

$$x_i, x'_i \in [LB_i, UB_i]$$

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

87

87

Real-Valued or Floating-Point Representation: Nonuniform Mutation

- 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 $N(0, \sigma)$ Gaussian distribution and then curtail to range

$$x'_i = x_i + N(0, \sigma)$$
 - Standard deviation σ , *mutation step size*, controls amount of change (2/3 of drawings will lie in range $(-\sigma \text{ to } +\sigma)$)

88

88

Real-Valued or Floating-Point Representation: Self-Adaptive Mutation (1/2)

- Step-sizes are included in the genome and undergo variation and selection themselves: $\langle x_1, \dots, x_n, \sigma \rangle$
- Mutation step size is not set by user but coevolves with solution
- Different mutation strategies may be appropriate in different stages of the evolutionary search process.

89

89

Real-Valued or Floating-Point Representation: Self-Adaptive Mutation (2/2)

- 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

90

90

Real-Valued or Floating-Point Representation: Crossover operators

- 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

91

91

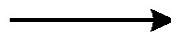
Real-Valued or Floating-Point Representation: Single arithmetic crossover

- Parents: $\langle x_1, \dots, x_n \rangle$ and $\langle y_1, \dots, y_n \rangle$
- Pick a single gene (k) at random,
- child₁ is:

$$\langle x_1, \dots, x_k, \alpha \cdot y_k + (1 - \alpha) \cdot x_k, \dots, x_n \rangle$$
- Reverse for other child. e.g. with $\alpha = 0.5$

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
-----	-----	-----	-----	-----	-----	-----	-----	-----

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.5	0.9
-----	-----	-----	-----	-----	-----	-----	-----	-----



0.3	0.2	0.3	0.2	0.3	0.2	0.3	0.2	0.3
-----	-----	-----	-----	-----	-----	-----	-----	-----

0.3	0.2	0.3	0.2	0.3	0.2	0.3	0.5	0.3
-----	-----	-----	-----	-----	-----	-----	-----	-----

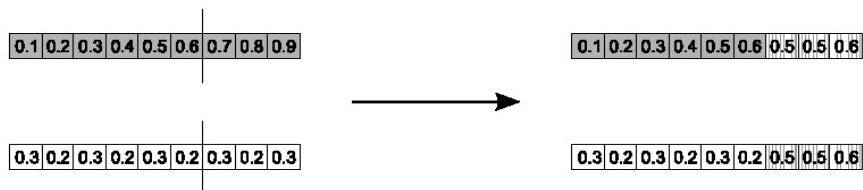
92

92

Real-Valued or Floating-Point Representation: Simple arithmetic crossover

- Parents: $\langle x_1, \dots, x_n \rangle$ and $\langle y_1, \dots, y_n \rangle$
- Pick a random gene (k) after this point mix values
- child₁ is:

$$\langle x_1, \dots, x_k, \alpha \cdot y_{k+1} + (1 - \alpha) \cdot x_{k+1}, \dots, \alpha \cdot y_n + (1 - \alpha) \cdot x_n \rangle$$
- reverse for other child. e.g. with $\alpha = 0.5$



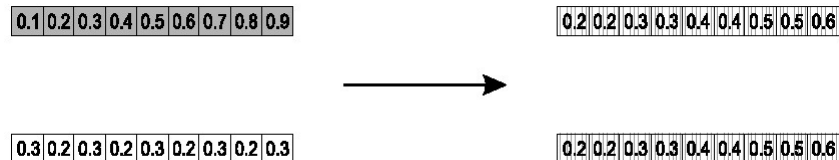
93

93

Real-Valued or Floating-Point Representation: Whole arithmetic crossover

- Most commonly used
- Parents: $\langle x_1, \dots, x_n \rangle$ and $\langle y_1, \dots, y_n \rangle$
- Child₁ is:

$$\alpha \cdot \bar{x} + (1 - \alpha) \cdot \bar{y}$$
- reverse for other child. e.g. with $\alpha = 0.5$



94

94

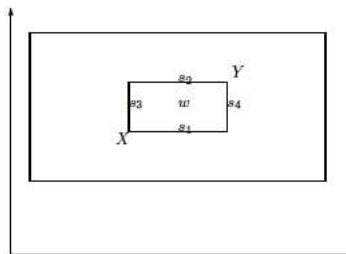
Real-Valued or Floating-Point Representation: Blend Crossover

- Parents: $\langle x_1, \dots, x_n \rangle$ and $\langle y_1, \dots, y_n \rangle$
- Assume $x_i < y_i$
- $d_i = y_i - x_i$
- Random sample $z_i = [x_i - \alpha d_i, x_i + \alpha d_i]$
- Original authors had best results with $\alpha = 0.5$

95

95

Real-Valued or Floating-Point Representation: Overview different possible offspring



- Single arithmetic: $\{s_1, s_2, s_3, s_4\}$
- Simple arithmetic / whole arithmetic: inner box ($w = \alpha 0.5$)
- Blend crossover: outer box

96

96

Real-Valued or Floating-Point Representation: Multi-parent recombination

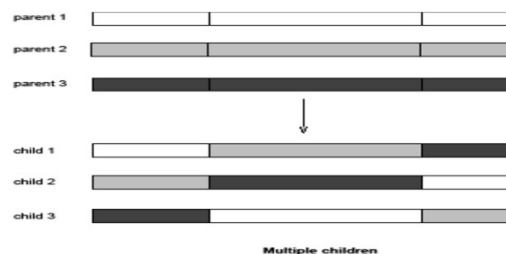
- Recall that we are not constricted by the practicalities of nature
- Noting that mutation uses $n = 1$ parent, and “traditional” crossover $n = 2$, the extension to $n > 2$ is natural to examine
- Been around since 1960s, still rare but studies indicate useful

32

97

Real-Valued or Floating-Point Representation: Multi-parent recombination, type 1

- Idea: segment and recombine parents
- Example: diagonal crossover for n parents:
 - Choose $n-1$ crossover points (same in each parent)
 - Compose n children from the segments of the parents in along a “diagonal”, wrapping around



- This operator generalises 1-point crossover

98

98

Real-Valued or Floating-Point Representation: Multi-parent recombination, type 2

- Idea: arithmetical combination of (real valued) alleles
- Example: arithmetic crossover for n parents:
 - i -th allele in child is the average of the parents' i -th alleles
- Creates center of mass as child
- Odd in genetic algorithms, long known and used in evolution strategies

99

99

Permutation Representations

- Ordering/sequencing problems form a special type
- Task is (or can be solved by) arranging some objects in a certain order
 - Example: production scheduling: important thing is which elements are scheduled 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

100

100

Permutation Representation: TSP example

- Problem:
 - Given n cities
 - Find a complete tour with minimal length
- Encoding:
 - Label the cities $1, 2, \dots, n$
 - 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 $30! \approx 10^{32}$ possible tours



101

101

Permutation Representations: Mutation

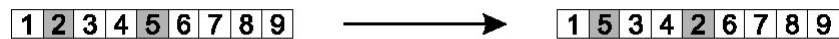
- Normal mutation operators lead to inadmissible solutions
 - e.g. bit-wise mutation: let gene i have value j
 - changing to some other value k would mean that k occurred twice and j no longer occurred
- Therefore must change at least two values
- Mutation parameter now reflects the probability that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

102

102

Permutation Representations: Swap mutation

- Pick two alleles at random and swap their positions

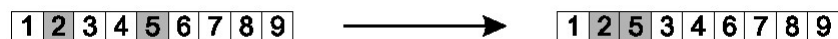


103

103

Permutation Representations: Insert Mutation

- 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

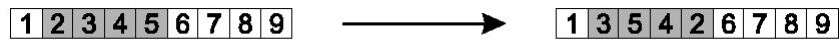


104

104

Permutation Representations: Scramble mutation

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

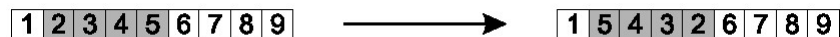


105

105

Permutation Representations: Inversion mutation

- 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

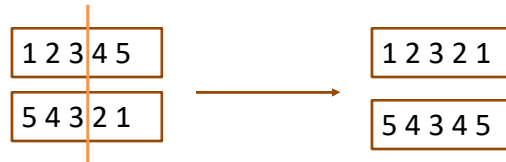


106

106

Permutation Representations: Crossover operators

- “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

107

107

Permutation Representations: Order 1 crossover (1/2)

- Idea is to preserve relative order that elements occur
- Informal procedure:
 - 1. Choose an arbitrary part from the first parent
 - 2. Copy this part to the first child
 - 3. Copy the numbers that are not in the first part, to the first child:
 - starting right from cut point of the copied part,
 - using the **order** of the second parent
 - and wrapping around at the end
 - 4. Analogous for the second child, with parent roles reversed

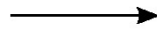
108

108

Permutation Representations: Order 1 crossover (2/2)

- Copy randomly selected set from first parent

1 2 3 4 5 6 7 8 9



 4 5 6 7

9 3 7 8 2 6 5 1 4

- Copy rest from second parent in order 1,9,3,8,2

1 2 3 4 5 6 7 8 9



3 8 2 4 5 6 7 1 9

9 3 7 8 2 6 5 1 4

109

109

Permutation Representations: Partially Mapped Crossover (PMX) (1/2)

Informal procedure for parents P1 and P2:

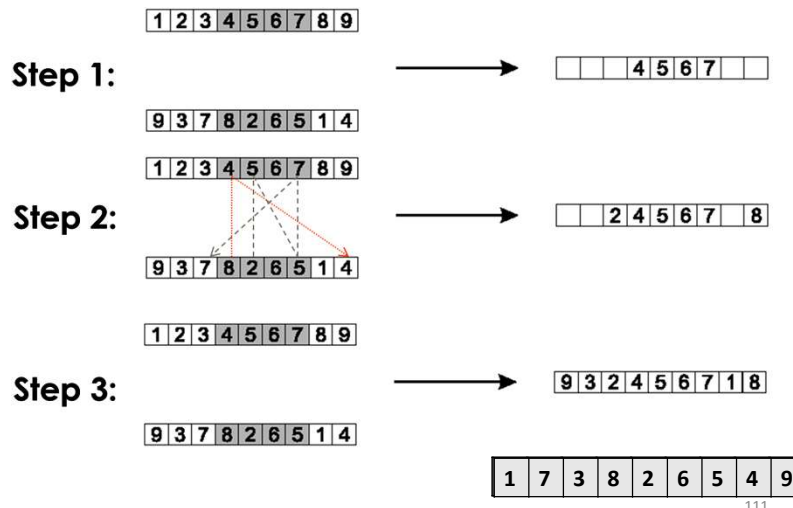
1. Choose random segment and copy it from P1
2. Starting from the first crossover point look for elements in that segment of P2 that have not been copied
3. For each of these i look in the offspring to see what element j has been copied in its place from P1
4. Place i into the position occupied j in 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

110

110

Permutation Representations: Partially Mapped Crossover (PMX) (2/2)



111

Permutation Representations: Cycle crossover (1/2)

Basic idea:

Each allele comes from one parent *together with its position*.

Informal procedure:

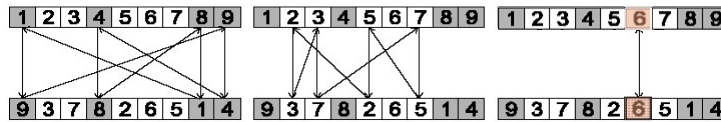
1. Make a cycle of alleles from P1 in the following way.
 - (a) Start with the first allele of P1.
 - (b) Look at the allele at the *same position* in P2.
 - (c) Go to the position with the *same allele* in P1.
 - (d) Add this allele to the cycle.
 - (e) Repeat step b through d until you arrive at the first allele of P1.
2. Put the alleles of the cycle in the first child on the positions they have in the first parent.
3. Take next cycle from second parent

112

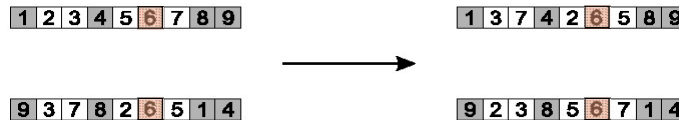
112

Permutation Representations: Cycle crossover (2/2)

- Step 1: identify cycles



- Step 2: copy alternate cycles into offspring



113

113

Permutation Representations: Edge Recombination (1/3)

- Works by constructing a table listing which edges are present in the two parents, if an edge is common to both, mark with a +
- e.g. [1 2 3 4 5 6 7 8 9] and [9 3 7 8 2 6 5 1 4]

Element	Edges	Element	Edges
1	2,5,4,9	6	2,5+,7
2	1,3,6,8	7	3,6,8+
3	2,4,7,9	8	2,7+, 9
4	1,3,5,9	9	1,3,4,8
5	1,4,6+		

114

114

Permutation Representations: Edge Recombination (2/3)

Informal procedure: once edge table is constructed

1. Pick an initial element, *entry*, at random and put it in the offspring
2. Set the variable *current element* = *entry*
3. Remove all references to *current element* from the table
4. Examine list for current element:
 - If there is a common edge, pick that to be next element
 - Otherwise pick the entry in the list which itself has the shortest list
 - Ties are split at random
5. In the case of reaching an empty list:
 - a new element is chosen at random

115

115

Permutation Representations: Edge Recombination (3/3)

Element	Edges	Element	Edges
1	2,5,4,9	6	2,5+,7
2	1,3,6,8	7	3,6,8+
3	2,4,7,9	8	2,7+, 9
4	1,3,5,9	9	1,3,4,8
5	1,4,6+		

Choices	Element selected	Reason	Partial result
All	1	Random	[1]
2,5,4,9	5	Shortest list	[1 5]
4,6	6	Common edge	[1 5 6]
2,7	2	Random choice (both have two items in list)	[1 5 6 2]
3,8	8	Shortest list	[1 5 6 2 8]
7,9	7	Common edge	[1 5 6 2 8 7]
3	3	Only item in list	[1 5 6 2 8 7 3]
4,9	9	Random choice	[1 5 6 2 8 7 3 9]
4	4	Last element	[1 5 6 2 8 7 3 9 4]

116

116

Tree Representation (1/6)

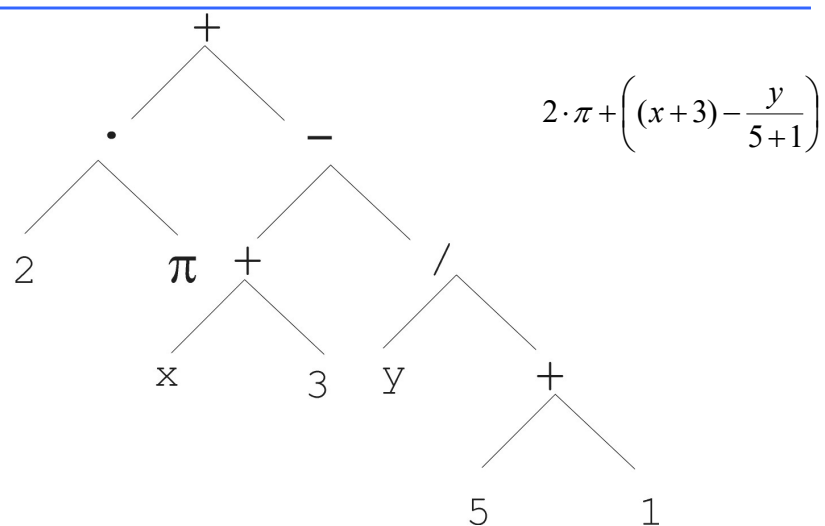
- Trees are a universal form, e.g. consider
- Arithmetic formula: $2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$
- Logical formula: $(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$
- Program:


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

117

117

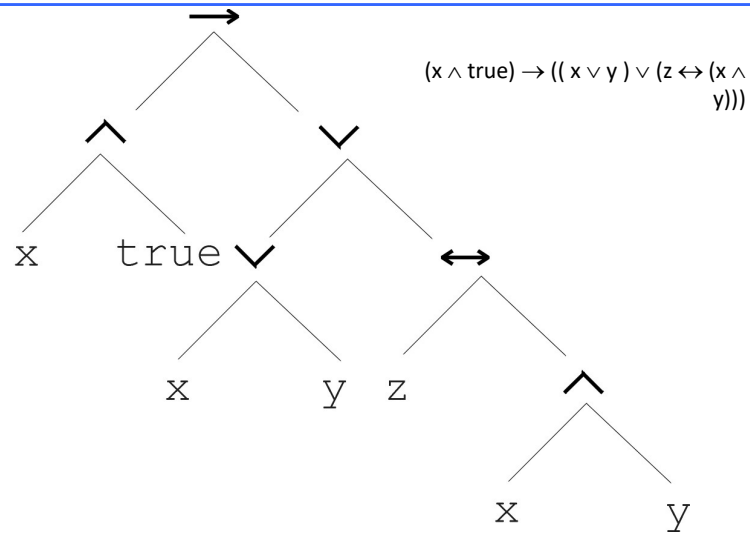
Tree Representation (2/6)



118

118

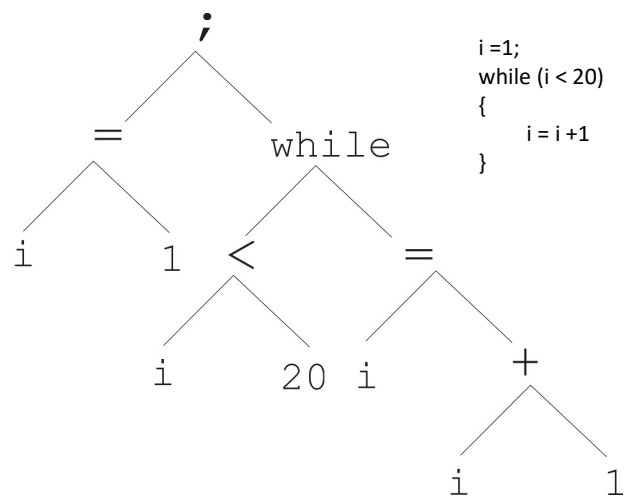
Tree Representation (3/6)



119

119

Tree Representation (4/6)



120

120

Tree Representation (5/6)

- 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

121

121

Tree Representation (6/6)

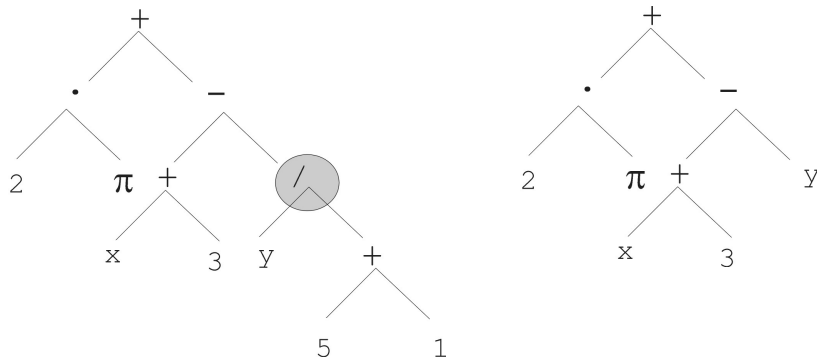
- Symbolic expressions can be defined by
 - Terminal set T
 - Function set F (with the arities of function symbols)
- Adopting the following general recursive definition:
 - Every $t \in T$ is a correct expression
 - $f(e_1, \dots, e_n)$ is a correct expression if $f \in F$, $\text{arity}(f)=n$ and e_1, \dots, e_n are correct expressions
 - 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)

122

122

Tree Representation: Mutation (1/2)

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



123

123

Tree Representation: Mutation (2/2)

- Mutation has two parameters:
 - Probability p_m to choose mutation
 - Probability to choose 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

124

124

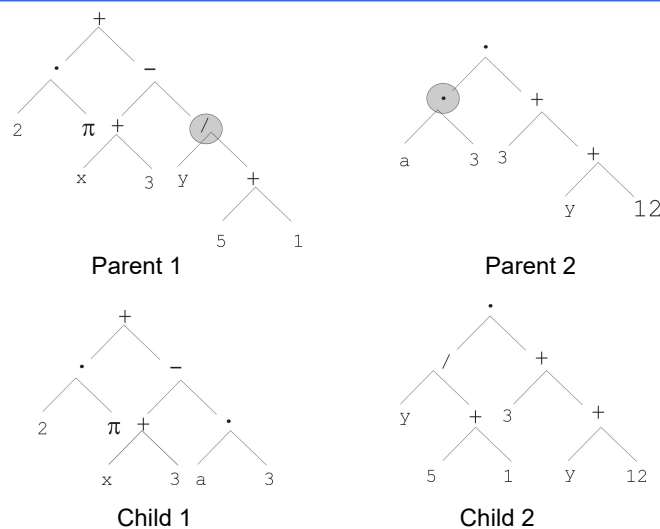
Tree Representation: Recombination (1/2)

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

125

125

Tree Representation: Recombination (2/2)



126

126

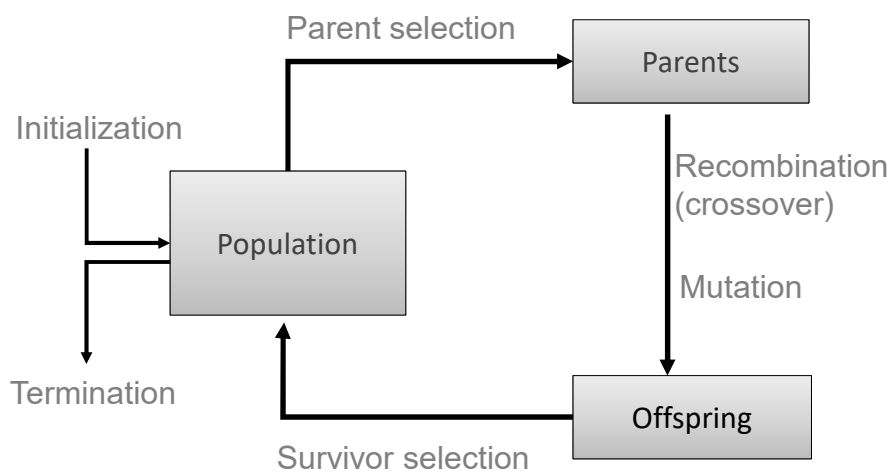
Fitness, Selection and Population Management

- Selection is second fundamental force for evolutionary systems
- Components exist of:
 - Population management models
 - Selection operators
 - Preserving diversity

127

127

Scheme of an EA: General scheme of EAs



128

128

Population Management Models: Introduction

- Two different population management models exist:
 - Generational model
 - each individual survives for exactly one generation
 - the entire set of parents is replaced by the offspring
 - Steady-state model
 - one offspring is generated per generation
 - one member of population replaced
- Generation Gap
 - The proportion of the population replaced
 - Parameter = 1.0 for GGA, = $1/\text{pop_size}$ for SSGA

129

129

Population Management Models: 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

130

130

Parent Selection: Fitness-Proportionate Selection

- Probability for individual i to be selected for mating in a population size μ with FPS is

$$P_{FPS}(i) = f_i / \sum_{j=1}^{\mu} f_j$$

- 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, loss of selection pressure
 - Highly susceptible to function transposition (example next slide)
- 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: $f'(i) = \max(f(i) - (\bar{f} - c \cdot \sigma_f), 0)$

where c is a constant, usually 2.0

131

131

Parent Selection: 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 (fittest has rank $\mu-1$ and worst rank 0)
- This imposes a sorting overhead on the algorithm, but this is usually negligible compared to the fitness evaluation time

132

132

Rank-based Selection: Linear Ranking

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

- Parameterised by factor s : $1 < s \leq 2$
 - measures advantage of best individual
- Simple 3 member example

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

133

133

Rank-based selection: Exponential Ranking

$$P_{exp-rank}(i) = \frac{1 - e^{-i}}{c}$$

- Linear Ranking is limited in selection pressure
- Exponential Ranking can allocate more than 2 copies to fittest individual
- Normalise constant factor c according to population size

Sample mating pool from the selection probability distribution (roulette wheel, stochastic universal sampling)

134

134

Parent Selection: Tournament Selection (1/2)

- All methods above rely on global population statistics
 - Could be a bottleneck esp. on parallel machines, very large population
 - Relies on presence of external fitness function which might not exist: e.g. evolving game players
- Idea for a procedure using only local fitness information:
 - Pick k members at random then select the best of these
 - Repeat to select more individuals

135

135

Parent Selection: Tournament Selection (2/2)

- Probability of selecting i will depend on:
 - Rank of i
 - Size of sample k
 - higher k increases selection pressure
 - Whether contestants are picked with replacement
 - Picking without replacement increases selection pressure
 - Whether fittest contestant always wins (deterministic) or this happens with probability p

136

136

Parent Selection: Uniform

$$P_{uniform}(i) = \frac{1}{\mu}$$

- Parents are selected by uniform random distribution whenever an operator needs one/some
- Uniform parent selection is unbiased - every individual has the same probability to be selected
- When working with extremely large populations, over-selection can be used.

137

137

Survivor Selection

- Managing the process of reducing the working memory of the EA from a set of μ parents and λ offspring to a set of μ individuals forming the next generation
- The parent selection mechanisms can also be used for selecting survivors
- Survivor selection can be divided into two approaches:
 - Age-Based Selection
 - Fitness is not taken into account
 - In SSGA can implement as “delete-random” (not recommended) or as first-in-first-out (a.k.a. delete-oldest)
 - Fitness-Based Replacement

138

138

Fitness-based replacement (1/2)

- Elitism
 - Always keep at least one copy of the fittest solution so far
 - Widely used in both population models (GGA, SSGA)
- 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
- Round-robin tournament
 - $P(t)$: 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 $q = 10$

139

139

Fitness-based replacement (2/2)

- (μ, λ) -selection
 - based on the set of children only ($\lambda > \mu$)
 - choose best μ
- $(\mu + \lambda)$ -selection
 - based on the set of parents and children
 - choose best μ
- 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
- $\lambda \approx 7 \cdot \mu$ is a traditionally good setting (decreasing over the last couple of years, $\lambda \approx 3 \cdot \mu$ seems more popular lately)

140

140

Selection Pressure

- Takeover time τ^* is a measure to quantify the selection pressure
- The number of generations it takes until the application of selection completely fills the population with copies of the best individual
- Goldberg and Deb showed:

$$\tau^* = \frac{\ln \lambda}{\ln(\lambda / \mu)}$$

- For proportional selection in a genetic algorithm the takeover time is $\lambda \ln(\lambda)$

141