ECM3412/ECMM406
Nature Inspired Computation
Lecture 4

Evolutionary Computation

Selection Schemes, Operators and Representations/Encodings

Today

- Algorithm Type Recap
- Selection Operators & Issues
- Mutation Operators
- Crossover Operators
- Replacement Operators
- Representations

Some Terms Used in Evolutionary Computation

- Algorithm Type
  - Generational
  - Steady State
  - Elitist
- Recombination (Crossover)
  - Single-point
  - Multi-point
  - Uniform
- Selection
  - Rank-Biased
  - Roulette Wheel
  - Tournament
- Replacer
  - Weakest
  - First weaker
- Mutation
- Parameters
  - Population size
  - Generations/iterations
  - Mutation rate
  - Crossover rate
  - Tournament size….
  - Etc

2 Types of Genetic Algorithm

- Generational – genetic operators applied repeatedly to generate new population.
- New solutions in yellow.

Initial Pop

Name	F(S)
S1	0.1
S2	0.5
S3	0.3
S4	0.2
S5	0.9
S6	0.7
S7	0.3
S8	0.4
S9	0.4
S10	0.1

Apply selection & genetic operators 10x to give new population →

Generation 1

Name	F(S)
S11	0.5
S12	0.3
S13	0.3
S14	0.7
S15	0.7
S16	0.9
S17	0.4
S18	0.9
S19	0.9
S20	0.3

Apply selection & genetic operators 10x to give new population →

Generation 2

Name	F(S)
S21	0.7
S22	0.8
S23	0.9
S24	0.9
S25	0.7
S26	0.8
S27	0.5
S28	0.7
S29	0.6
S30	0.7

2 Types of Genetic Algorithm II

- Steady State – genetic operators applied N times and bad solutions replaced
- New solutions in yellow.

Initial Pop

Name	F(S)
S1	0.1
S2	0.5
S3	0.3
S4	0.2
S5	0.9
S6	0.7
S7	0.3
S8	0.4
S9	0.4
S10	0.1

Apply selection & genetic operators N times to give new individuals

Name	F(S)
S11	0.1
S12	0.5

Replace weak solutions in population

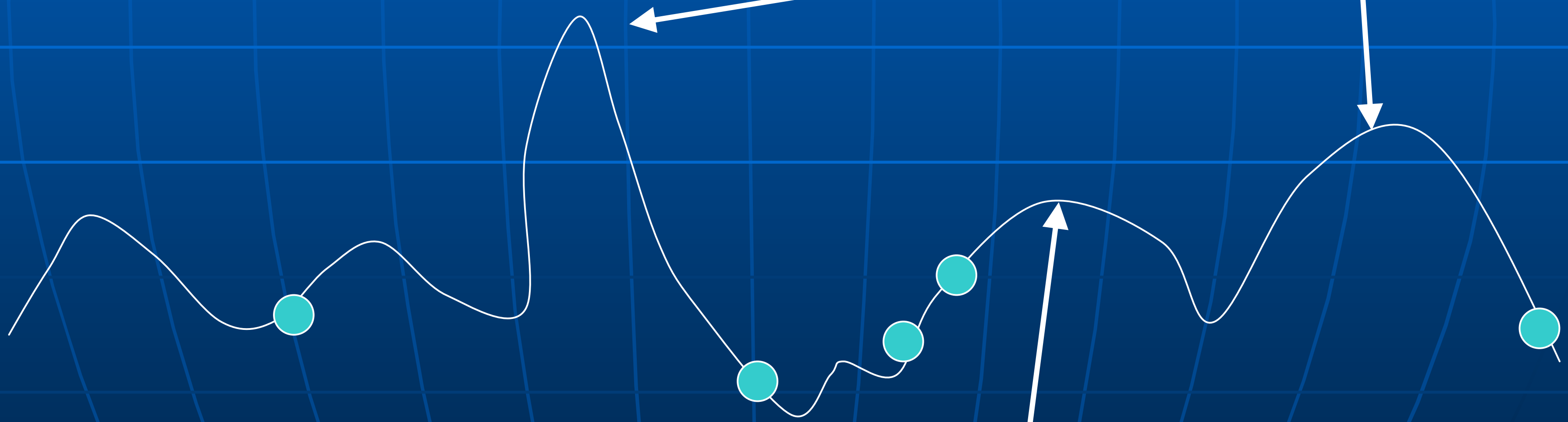
Generation 1

Name	F(S)
S12	0.5
S2	0.5
S3	0.3
S4	0.2
S5	0.9
S6	0.7
S7	0.3
S8	0.4
S9	0.4
S11	0.1

Selection Issues

Very low pressure selection (e.g. random)
No evolutionary `progress' at all.

A modest level of pressure.
You may well find yourself here or here:



Very high pressure (e.g. always select best)
You will end up here:

Some Selection Methods

Grand old method:

Fitness Proportionate Selection also called

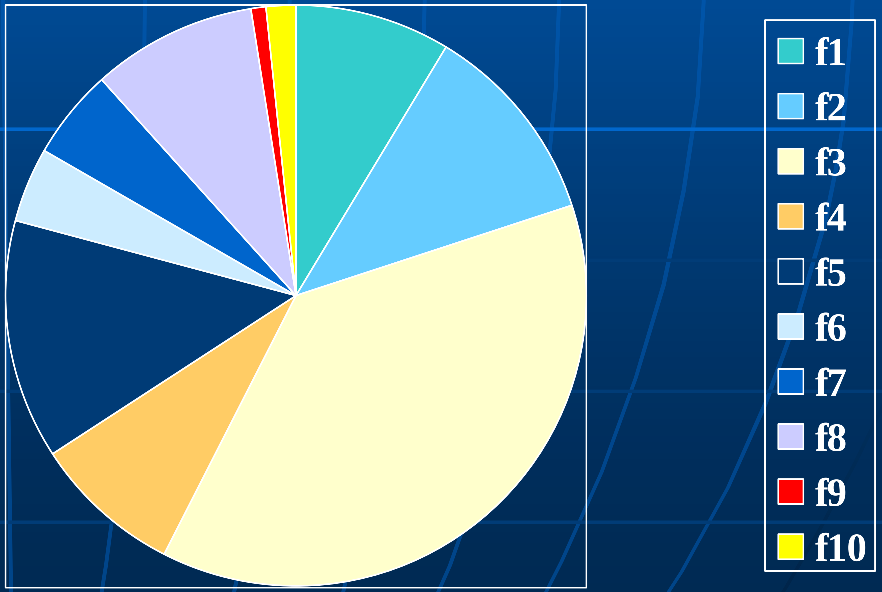
Roulette Wheel selection

Suppose there are P individuals with fitnesses $f1, f2, \ldots, fP$

The probability of selecting individual i is simply:

$$\frac{f_i}{\sum_{k=1}^{P} f_k}$$

This is equivalent to spinning a roulette wheel with sectors proportional to fitness



Problems with Roulette Wheel Selection

Having prob of selection directly proportional to selection has a nice ring to it. It is still used a lot, but:

What about when we are trying to *minimise* the `fitness' value?

What about when we may have negative fitness values?

- Suppose we are trying to maximise something, and we have a population of 5 fitnesses:

  100, 0.4, 0.3, 0.2, 0.1 -- the best is 100 times more likely to be selected than all the rest put together!
- A modified f(s) might give us:

  200, 100.4, 100.3, 100.2, 100.1 – a much more reasonable situation.

- Point is: Fitprop requires us to be very careful how we design the fine detail of fitness assignment.

- Other selection methods are better in this respect, and more used now.

Tournament Selection

Tournament selection: tournament size = *t*

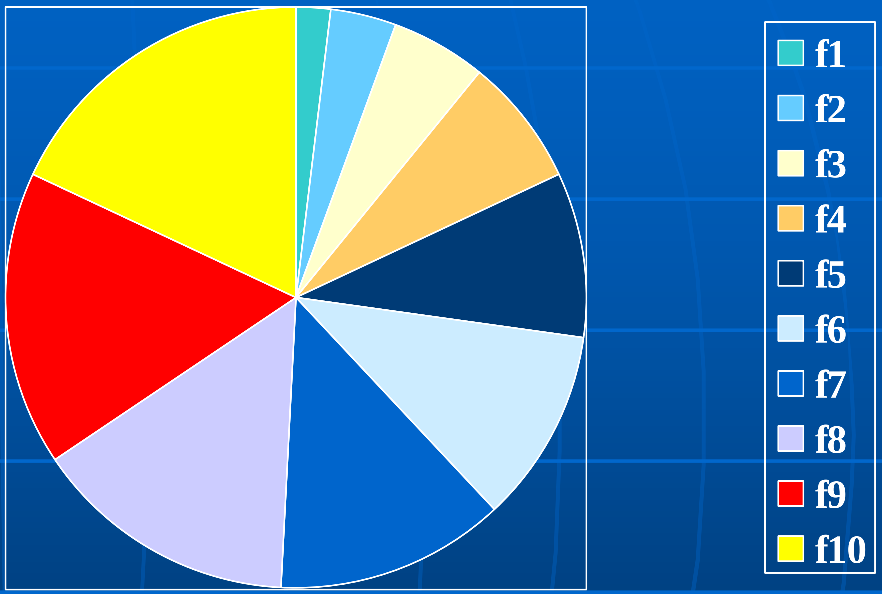
```
Repeat t times
   choose a random individual from the pop
   and remember its fitness

Return the best of these t individuals (BTR)
```

- \+ Very tunable
- \+ Avoids the problems of superfit or superpoor solutions
- \+ Very simple to implement
- \- Requires another variable (tournament size)

Rank Based Selection

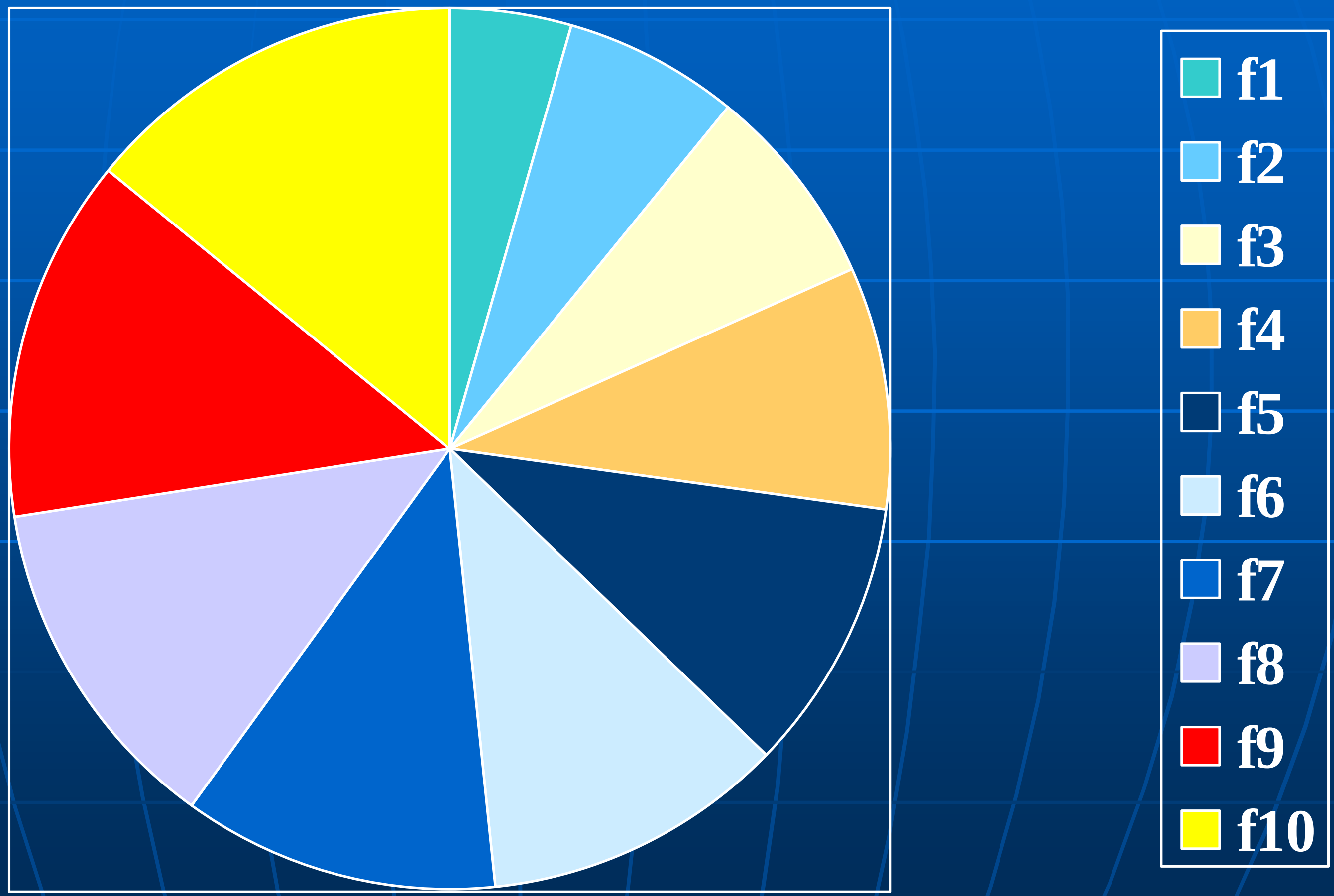
The fitnesses in the pop are *Ranked* from *Popsize* (fittest) down to 1 (least fit). The selection probabilities are proportional to rank.

There are variants where the selection probabilities are a function of the rank.



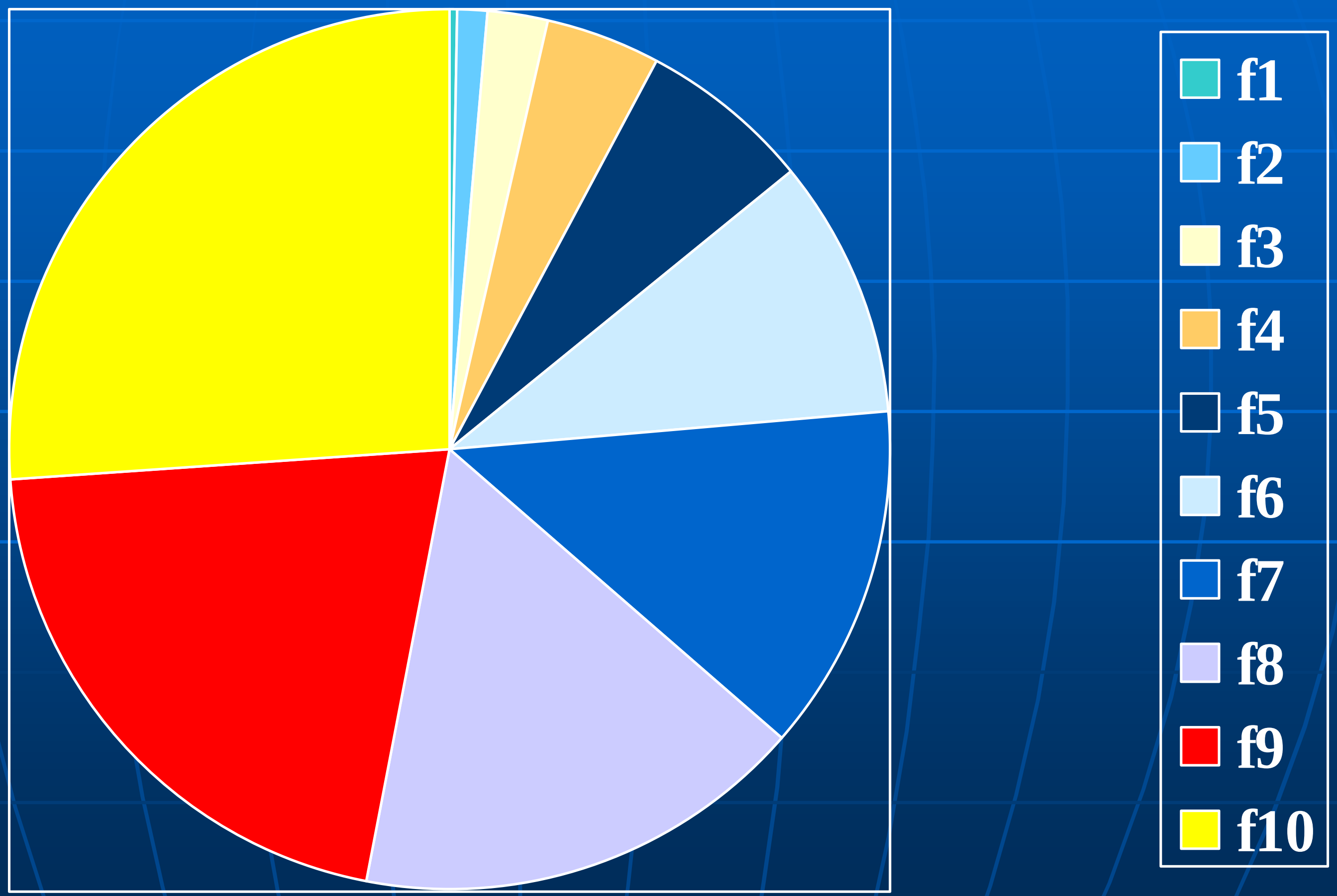
Rank with low bias

Here, selective fitnesses are based on *rank*$^{0.5}$



Rank with high bias

Here, selective fitnesses are based on *rank*^{2}



Tournament Selection

- Parameter: tournament size, t
- To select a parent, randomly choose t individuals from the population (with replacement).
- Return the fittest of these t

- What happens to selection pressure as we increase t?
- What degree of selection pressure is there if $t = 10$ and $popsize = 10{,}000$?

Truncation selection

- Applicable only in *generational* algorithms, where each generation involves replacing most or all of the population.
- Parameter *pcg* (ranging from 0 to 100%)

- Take the best *pcg*% of the population;
- produce the next generation entirely by applying variation operators to these.
- How does selection pressure vary with *pcg* ?

How to think of genetic operators

- ***Selection*** **represents our strategy for deciding which areas of the search space to focus our efforts on.**

- ***Operators*** **provide our means to generate new candidate solutions.**
  - *We want operators to have a fair chance of yielding good new solutions (small change, and/or combine bits from solutions we already know are good)*
  - *We also (obviously) want to be able to potentially explore the whole space.*

Genetic Operators / Variation Methods

Often we have used a k-ary encoding, in which a candidate solution is just a list of L numbers, each of which can be anything from 0 to k-1 (or 1 to k).

E.g. our simple clustering representation for 100 genes and 5 groups, would be a 5-ary encoding of length L=100.

These might be candidate solutions from a k-ary encoding with L = 10 and k = 20:

[17, 2, 19, 1, 1, 1, 5, 11, 12, 2]

[16, 19, 2, 19, 2, 3, 4, 7, 5, 2]

Mutation in *k*-ary encodings

Single-gene mutation:

Choose a gene at random, and change it to a random new value. E.g. 352872 → 312872

M-gene mutation:

Multiple instances of single-gene mutation

Swap mutation:

Choose two genes at random, and swap them. E.g. 352872 → 372852

Why is this probably not much good in this context?

Mutation in Real-Valued Encodings

- Often a candidate solution is a vector of real numbers of length *L*
- Single-gene mutation:
  - Choose a gene at random, and add a small random deviation to it. Often chosen from a Gaussian distribution.
- Vector mutation:
  - Generate a small random vector of length *L*, and add it to the whole thing.

Mutation in Permutation-Based Encodings

Here is a permutation of length 10:

DEGJACBFIH

Can we do single-gene mutation?

Can we do swap mutation?

What else might we do?

Recombination Methods for *k*-ary Encodings

The standard recombination operators are:

1-point crossover:

Recombination Methods for *k*-ary Encodings

2-point crossover:

k-point crossover – again, what do you think it is?

Uniform Crossover

Uniform Crossover

Generate a random binary `mask’

Use it to decide which genes are swapped and which stay.

A	B	C	D	E	F	G	H	Parent 1
K	L	M	N	O	P	Q	R	Parent 2
0	1	0	0	1	1	0	1	Binary Mask
A	L	C	D	O	P	G	R	Child 1
K	B	M	N	E	F	Q	H	Child 2

Replacement

- Some different strategies exist for replacement in steady-state algorithms:

- Replace Weakest:

Initial Pop

Name	F(S)
S1	0.1
S2	0.5
S3	0.3
S4	0.2
S5	0.9
S6	0.7
S7	0.3
S8	0.4
S9	0.4
S10	0.1

Name	F(S)
S11	0.1
S12	0.5

Replace weakest solutions in population

Generation 1

Name	F(S)
S12	0.5
S2	0.5
S3	0.3
S4	0.2
S5	0.9
S6	0.7
S7	0.3
S8	0.4
S9	0.4
S11	0.1

Replacement II

- Replace First Weakest:

Initial Pop

Name	F(S)
S1	0.3
S2	0.5
S3	0.3
S4	0.2
S5	0.9
S6	0.7
S7	0.3
S8	0.4
S9	0.4
S10	0.1

New Solutions

Name	F(S)
S11	0.2
S12	0.5

Find first solution which is weaker than the new solution

Generation 1

Name	F(S)
S12	0.5
S2	0.5
S3	0.3
S11	0.2
S5	0.9
S6	0.7
S7	0.3
S8	0.4
S9	0.4
S10	0.1

Encoding / Representation

Maybe the main issue in (applying) EC

Note that:

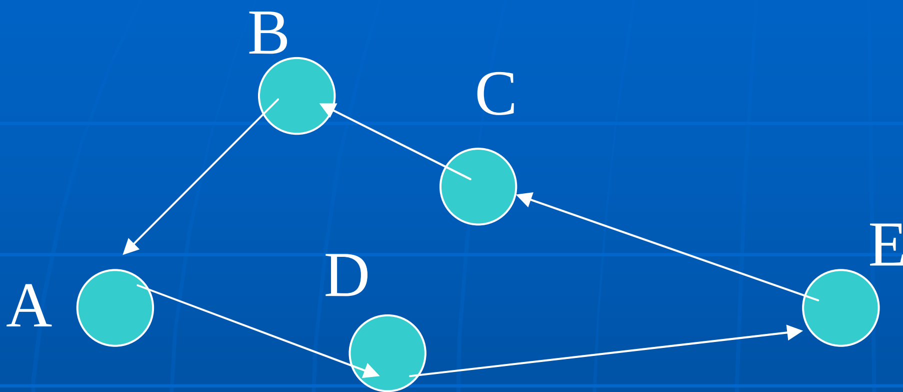
- Given an optimisation problem to solve, we need to find a way of encoding candidate solutions
- There can be many very different encodings for the same problem
- Each way affects the shape of the landscape and the choice of best strategy for climbing that landscape.

Representation

- The chromosome of a solution can be made up of (for example):
  - Integers
  - Real Values
  - Binary Strings
- The choice of representation and encoding can influence the operators needed to successfully run the optimisation.
- Some encodings require specialised operators (e.g. permutation based problems)

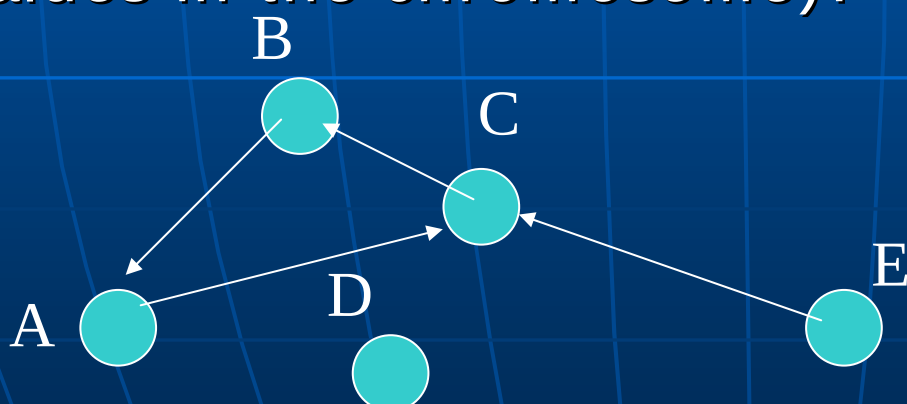
Representation Example

- Revisiting the TSP



A	D	E	C	B

- Standard Mutation (randomly change one of the values in the chromosome):

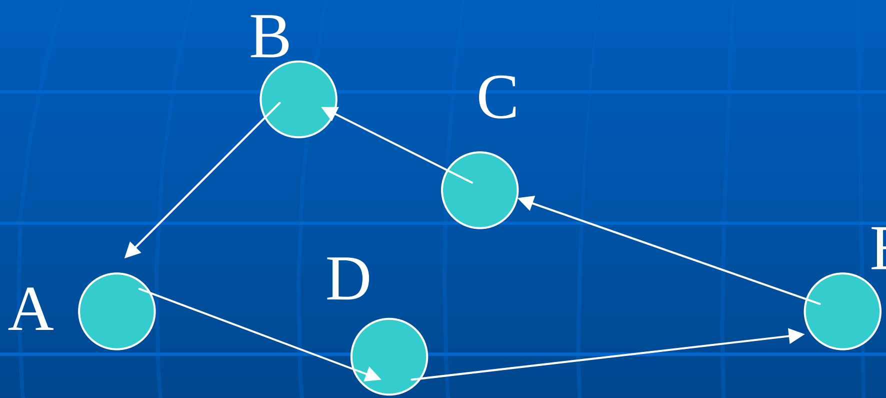


A	C	E	C	B

So we now visit C twice and D not at all!!

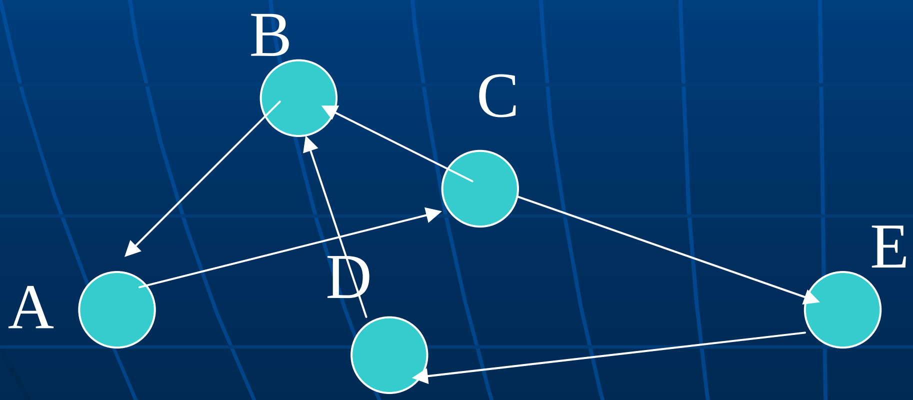
Representation Example (contd.)

- This is clearly not what is intended, so we must replace the standard mutation with something which does the job properly.



A	D	E	C	B

- The swap operator would seem to work:



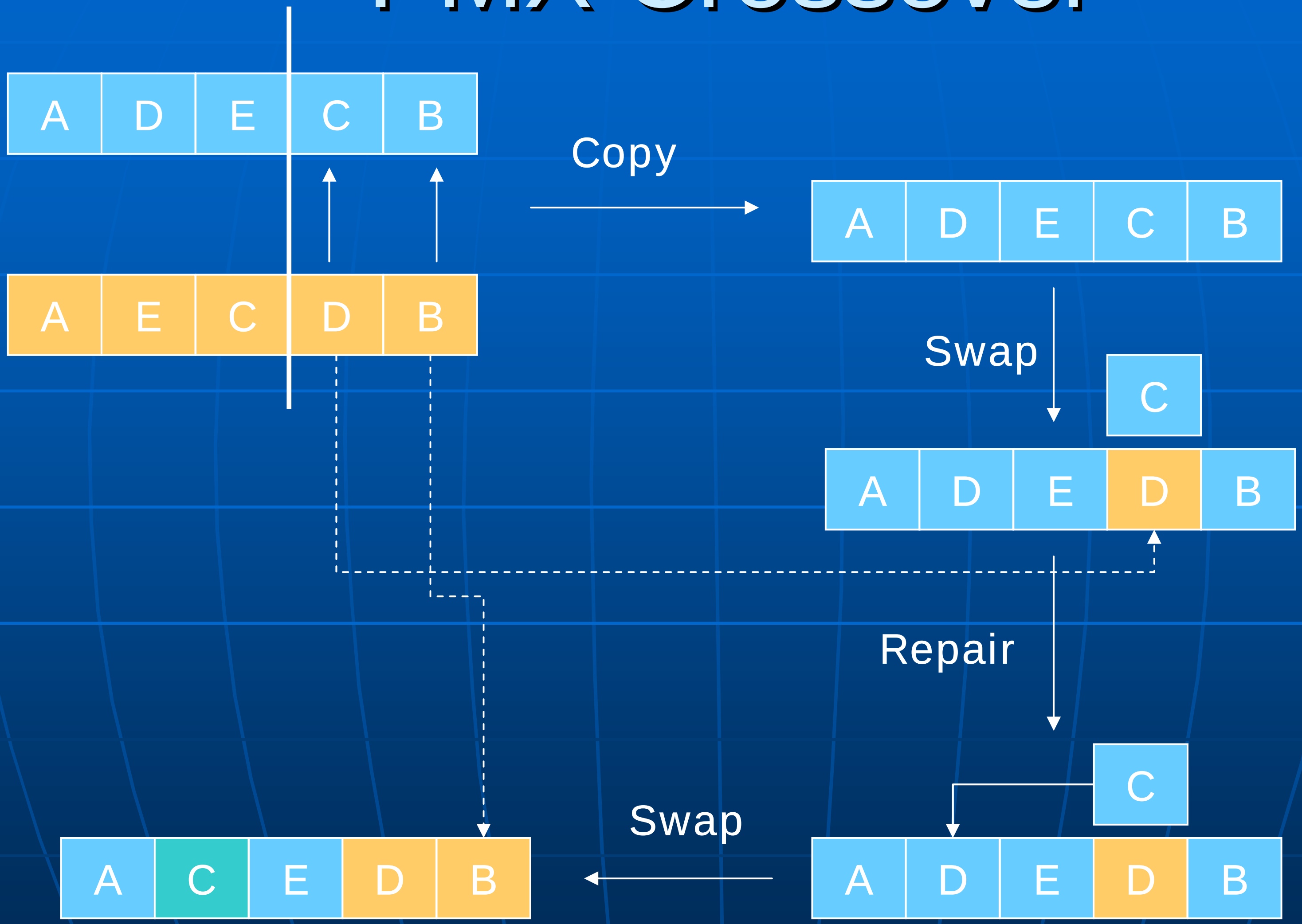
A	C	E	D	B

Representation Example (contd.)

- The same problem exists with crossover.
- Standard crossover

- This gives 2 invalid children which visit the same node twice
- 2 Strategies can help:
  1. Have a better cross-over in the first place
  2. Introduce a ‘fix’ to make the crossed-over solutions valid again.

PMX Crossover



Summary

- Selection should have a bias towards the fittest solutions
- There are varieties of genetic operators for different problem types
- Representation of the problem to the algorithm and genetic operator selection are linked
- Next time more on representations/encodings