6 Representation, Mutation and Recombination Part 1:
Binary and Integer Representations

Recap: General Scheme of an EA

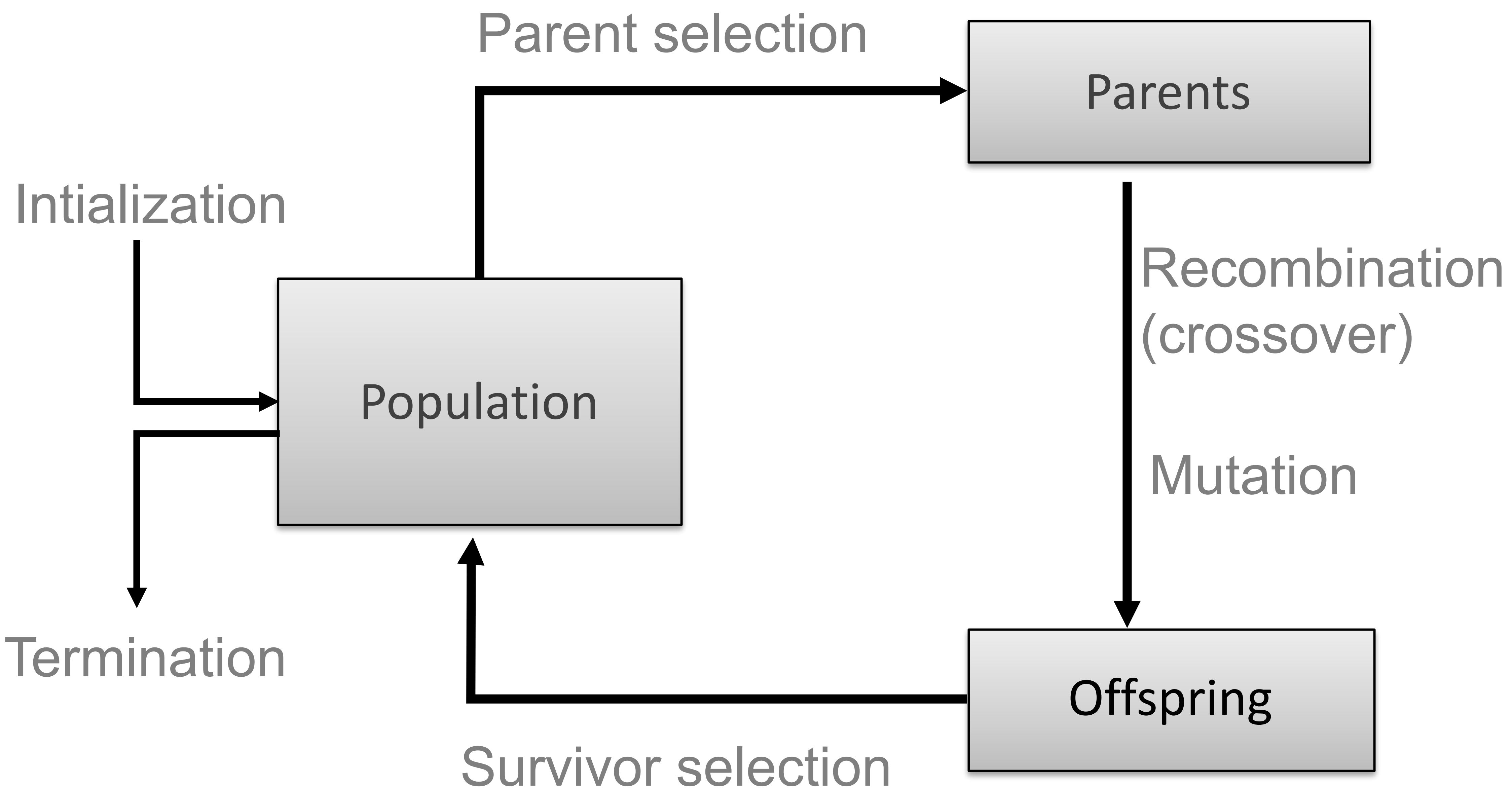


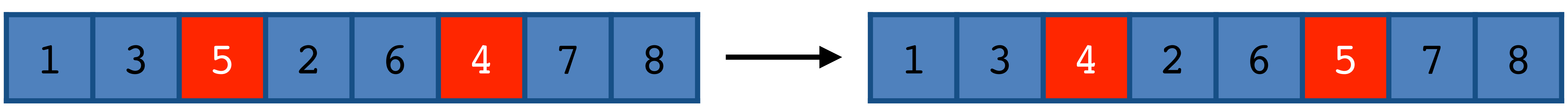
figure 3.2, Introduction to Evolutionary Computation

Recap: Representation

- want a representation that allows exploration *and* exploitation
- for exploration we must be able to represent all valid solutions
- for exploitation we need small changes in the representation to be able to lead to small changes in the fitness of an individual

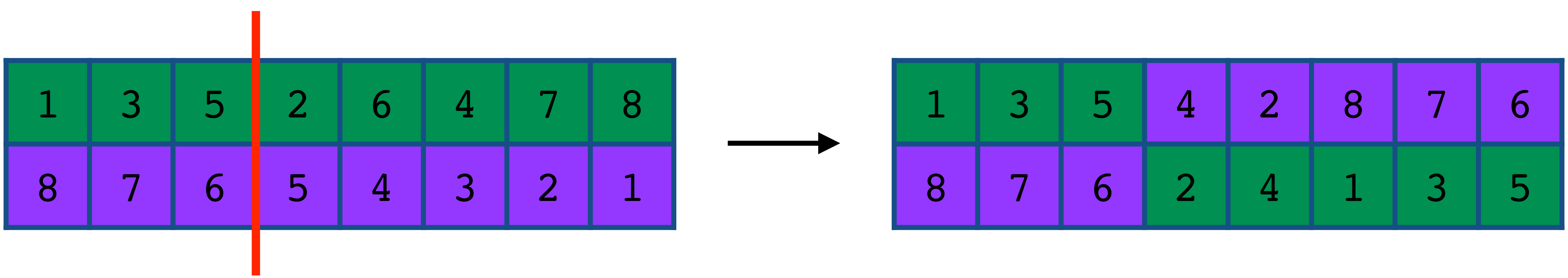
Recap: Mutation

- two aspects:
- mutation rate
  - the probability that mutation happens at all
- mutation operator
  - what actually happens if mutation occurs



Recap: Recombination

- usually two parents produce two children
- two aspects:
- recombination rate
  - if p_c is the recombination rate then:
    - P(children are different to parents) = p_c
    - P(children are same as parents) = $1-p_c$
- recombination operator
  - what actually happens if recombination occurs



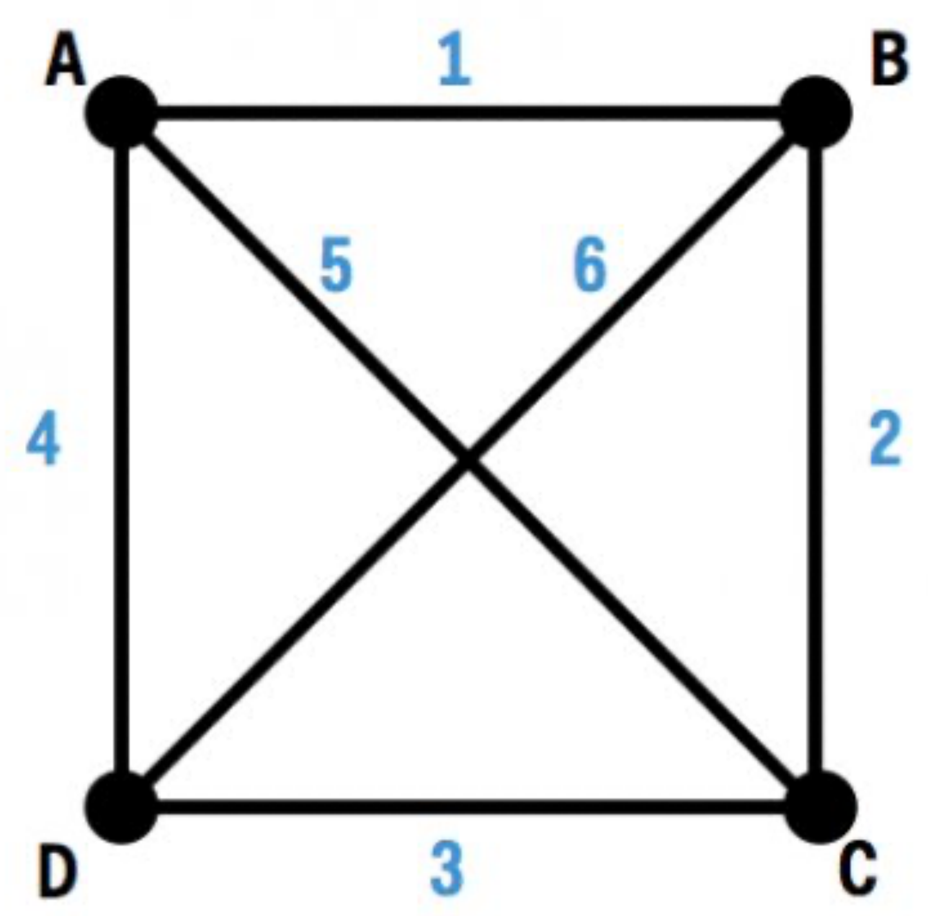
Role of Representation & Variation Operators

- choosing right representation for the problem is the first and most difficult stage of building an EA
- key variation operators are mutation and crossover
- the type of variation operators needed depends upon the chosen representation
- example: the TSP problem Travel Sales Man
  - what are possible representations?

TSP: How should we represent solutions?

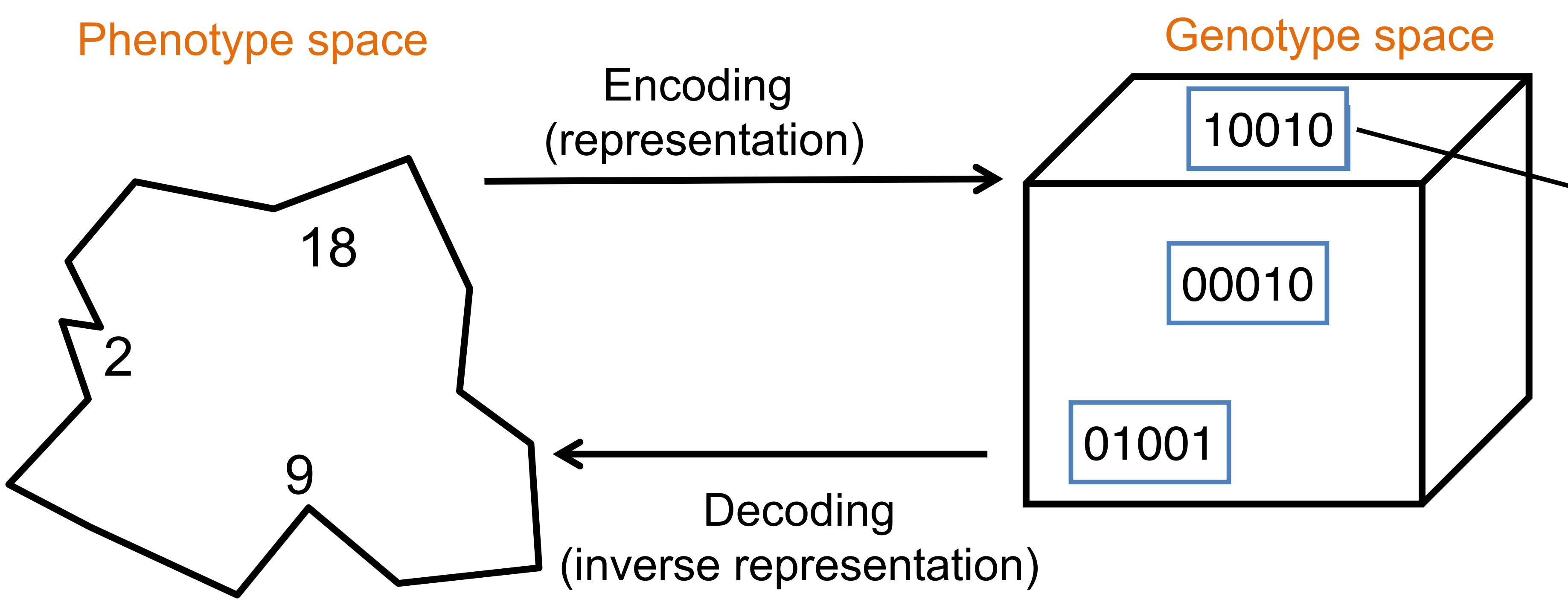
One way to do it...

- genotype: integer or binary?
    - integer is better:
    - A→D→B→C ⇒ `[0,3,1,2]`
- values (alleles) allowed?
    - permutations of `{0,1,2,3}`
- mutation operator?
    - swap two genes' values
- recombination method?
    - cut-and-crossfill (see slides #5)



(Recap:) Binary Representation

- example: represent integer values by their binary code
- one of the earliest representations



- each binary string is a genotype
- each bit is a gene
- the value of each bit is an allele

Binary Rep: Mutation

- bitwise mutation is most common technique:
  - alter each gene independently with a probability p_m
  - p_m is called the mutation rate
  - typically has a value between `1/pop_size` and `1/chromosome_length`
- example: $p_m = 0.5$

parent	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
child	0	1	0	0	1	0	1	1	0	0	0	1	0	1	1	0	0	1

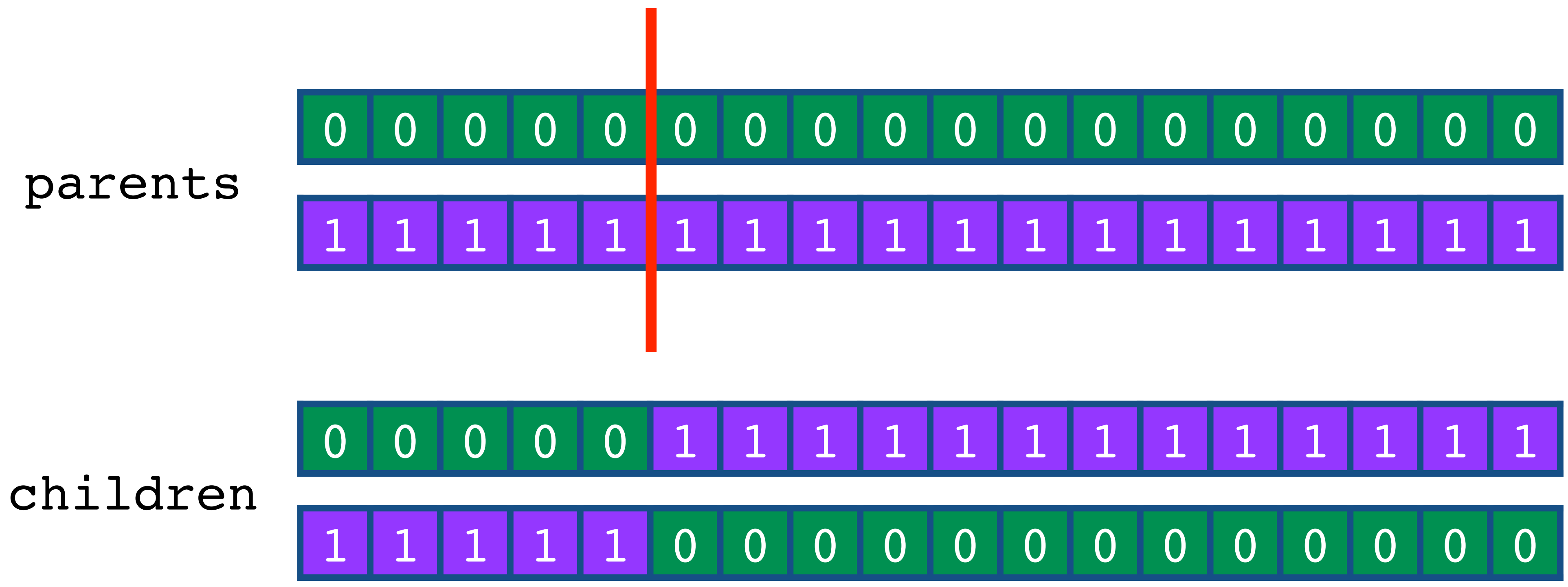
Issues with Bitwise Mutation

- using bit strings to represent non-binary solutions is usually a mistake
- why?
  - because different bits can have different significance
  - so the effect of a single bit mutation is highly variable
- can use Gray Coding to ensure that consecutive integers have Hamming Distance of 1

Decimal	Binary	Gray
0	0	0
1	1	1
2	10	11
3	11	10
4	100	110
5	101	111
6	110	101
7	111	100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Binary Rep: 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)

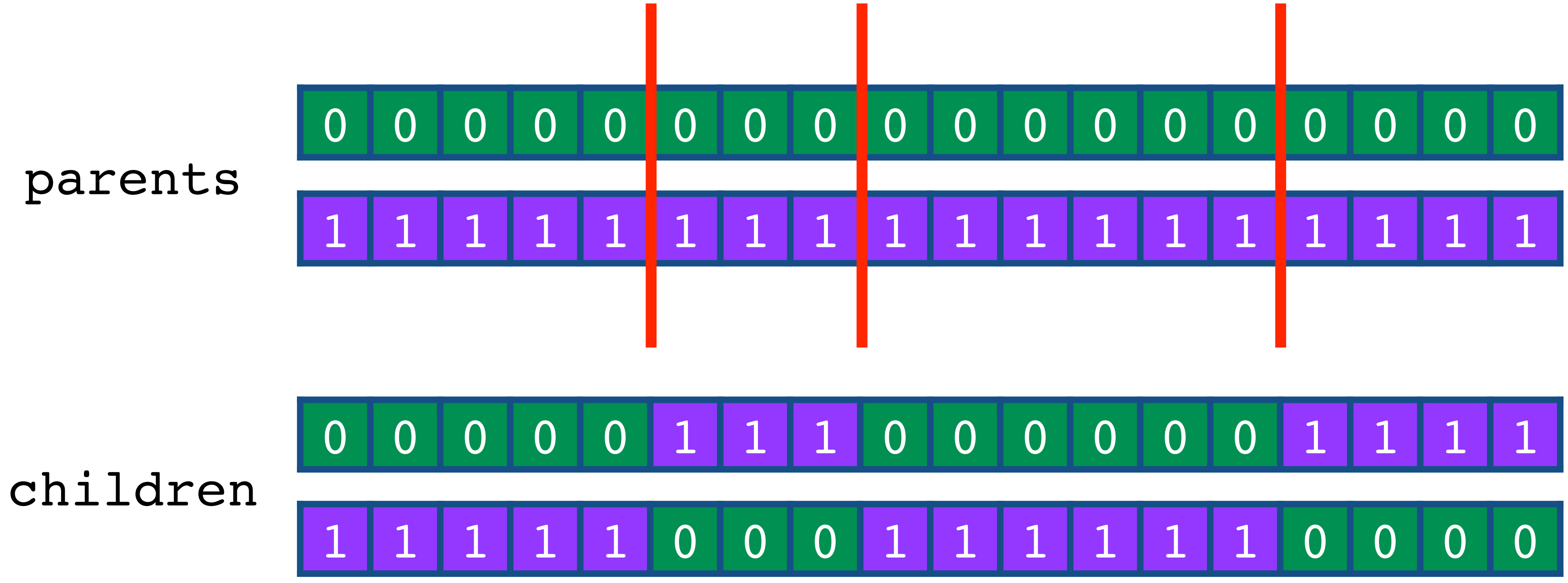


Binary Rep: Alternative Crossover Operators

- why do we need other types of crossover?
- 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

Binary Rep: n-point Crossover

- choose n random crossover points
- split along those points
- glue parts, alternating between parents
- because it's a generalisation of 1-point crossover it still has some positional bias



Binary Rep: 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
  - so no positional bias

parents	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
children	0	1	0	0	1	0	1	1	0	0	0	1	0	1	1	0	0	1
	1	0	1	1	0	1	0	0	1	1	1	0	1	0	0	1	1	0

Binary Rep: Crossover OR mutation?

- a decade long debate: which one is better or necessary?
- wide agreement that it depends on the problem
- but in general, it is good to have both
- mutation-only-EA is possible
- but crossover-only-EA would not work
- why?

Binary Rep: Crossover OR mutation?

- recall two key activities that an EA needs to support:
- exploration:
    - discovering promising areas in the search space
    - *gaining information on the problem*
- exploitation:
    - optimising within a promising area
    - *using information*
- there is co-operation AND competition between these two activities
- 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

Binary Rep: Crossover OR mutation?

- only crossover can combine information from two parents
- but crossover does not change the allele frequencies of the population
  - consider an initial population where every individual's first gene had the value 0…
- only mutation can introduce new information (new alleles)
- to hit the optimum you often need a 'lucky' mutation

Integer Representation

- it is generally accepted that it is better to encode numerical variables directly
    - as integers or floating point variables
    - recall the slide *Issues with Bitwise Mutation*
- some problems naturally have integer variables
    - such as image processing parameters
    - where the ordering of the values is natural, or *ordinal*
- others take categorical values from a fixed set
    - such as {blue, green, yellow, red} for the k-colouring problem
    - where the ordering is arbitrary, or *cardinal*

Integer Rep: Recombination and Mutation

- recombination
    - n-point and uniform crossover operators work the same as with binary reps
- mutation
    - the bit-flipping principle for binary reps can be extended to work for integer reps in one of two ways:
    - random resetting:
        - a new value is chosen for each gene with probability p_m
        - most suitable for cardinal attributes, because all other gene values are equally likely to be chosen
    - creep:
        - add a small positive or negative value v to each gene with probability p_m
        - v is sampled randomly from a distribution centred around zero
        - most suitable for ordinal attributes

Integer Rep: Mutation Examples

- random resetting with cardinal attributes:

parent	N	N	E	N	N	E	E	S	S	E	E	N	N	W	S	W	E	E
child	S	N	W	W	N	E	E	S	N	S	W	N	W	W	S	E	N	E

set to any value

- creep with ordinal attributes:

parent	8	5	1	2	4	8	3	3	7	6	2	2	3	9	3	4	5	2
child	7	5	2	1	4	9	3	3	8	7	1	2	2	9	3	3	4	2

set to parent_value ± 1

Reading & References

- slides largely based on and adapted from, Chapter 4 slides for Eiben & Smith's *Introduction to Evolutionary Computing*
- W.M. Spears: Evolutionary Algorithms: The Role of Mutation and Recombination, Springer 2000
- K. Deb: Representations. Part 4 of T. Bäck, D. Fogel and Z. Michalewicz (editors) Evolutionary Computation 1: Basic Algorithms and Operators, Institute of Physics Press
  - *note that above link leads directly to a .pdf download*