7: Permutation Representation

- Problems with permutation solutions
  - order-type
  - adjacency-type
- Mutations for permutation
  - swap, insert, scramble, inversion
- Crossover for permutation
  - PMX, edge, order
- Textbook Chapter 4.5

Prof. Ting Hu, School of Computing, Queen’s University

Permutation representation

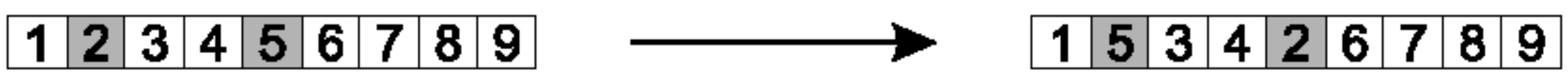
- Ordering / sequencing problems form a special type
- Task is to arrange some objects in a certain order
  - Example: sorting, scheduling, where *order* is important
  - Example: Traveling Salesperson Problem (TSP) where *adjacency* is important
- These problems are generally expressed as a permutation
  - if there are *n* variables then the representation is a list of *n* integers, each of which occurs exactly once

Mutation operators for permutation

- 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

Swap mutation

- Pick two allele values at random and swap their positions

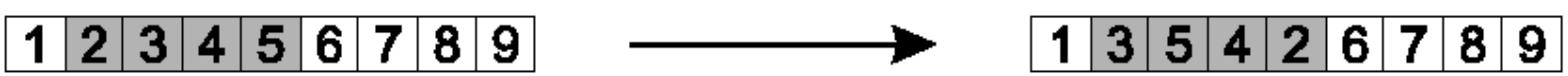


Insert mutation

- Pick two allele values at random
- Move the second to follow the first, shifting the rest along to accommodate

Scramble mutation

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



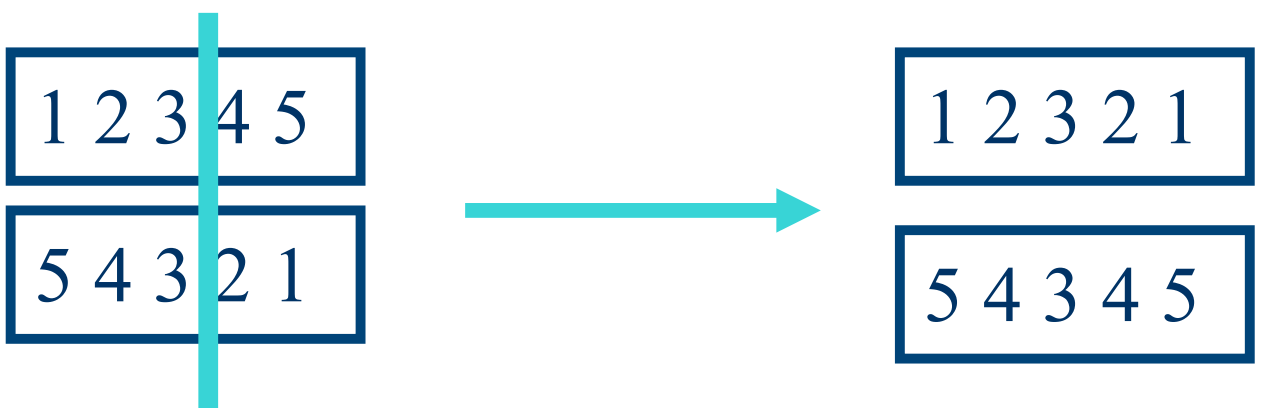
- Note subset does not have to be contiguous

Inversion mutation

- Pick two allele values at random and then invert the substring between them
- Particularly designed for TSP

Crossover operators for permutation

- “Normal” crossover operators will often lead to inadmissible solutions



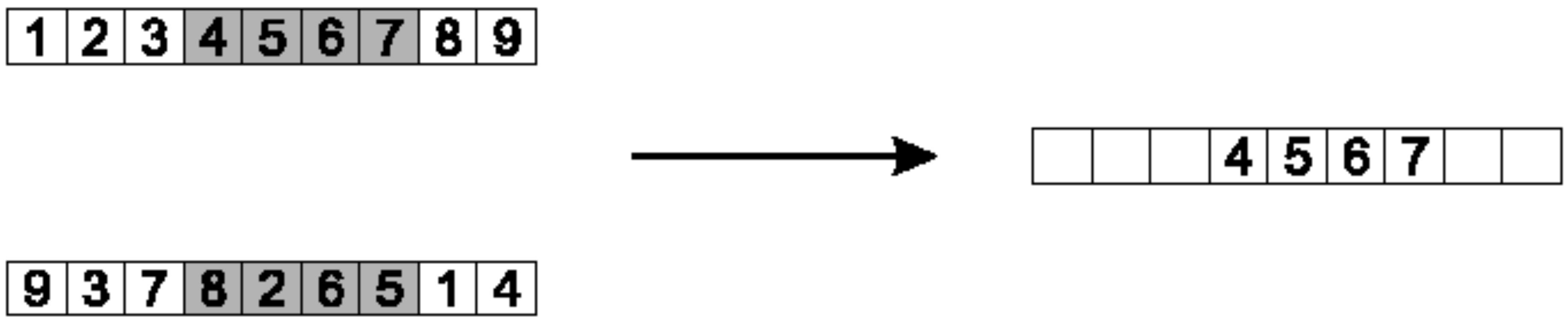
- Many specialized operators have been devised which focus on combining order or adjacency information from the two parents

Partially mapped crossover (PMX)

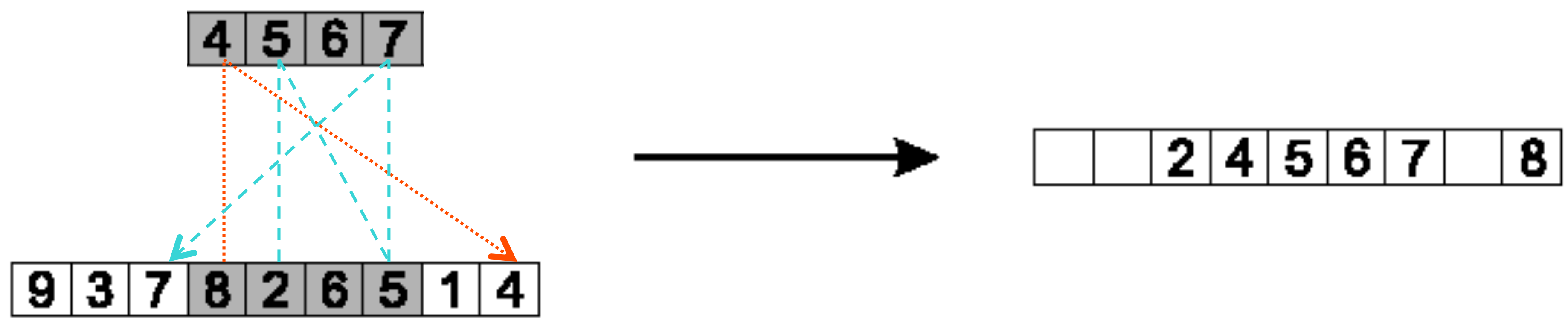
- Idea is to preserve adjacency information
- For parents P1 and P2:
  1. Choose a random segment and copy it from P1
  2. Starting from the first crossover point look for elements in that segment of P2 that have not been copied
  3. For each of these *i* look in the offspring to see what element *j* has been copied in its place from P1
  4. Place *i* into the position occupied by *j* in P2, since we know that we will not put *j* there (as is already in offspring)
  5. If the place occupied by *j* in P2 has already been filled in the offspring by *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.
  7. Second child is created similarly.

Partially mapped crossover: example

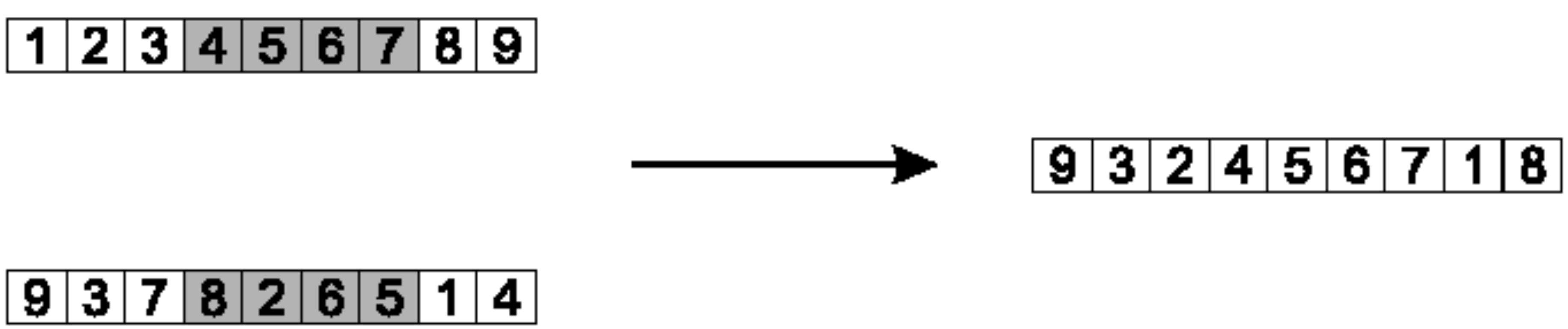
- Step 1



- Step 2



- Step 3



Edge crossover

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+		

- Two parents [1 2 3 4 5 6 7 8 9] and [9 3 7 8 2 6 5 1 4]

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

Edge crossover: example

- Two parents [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+		

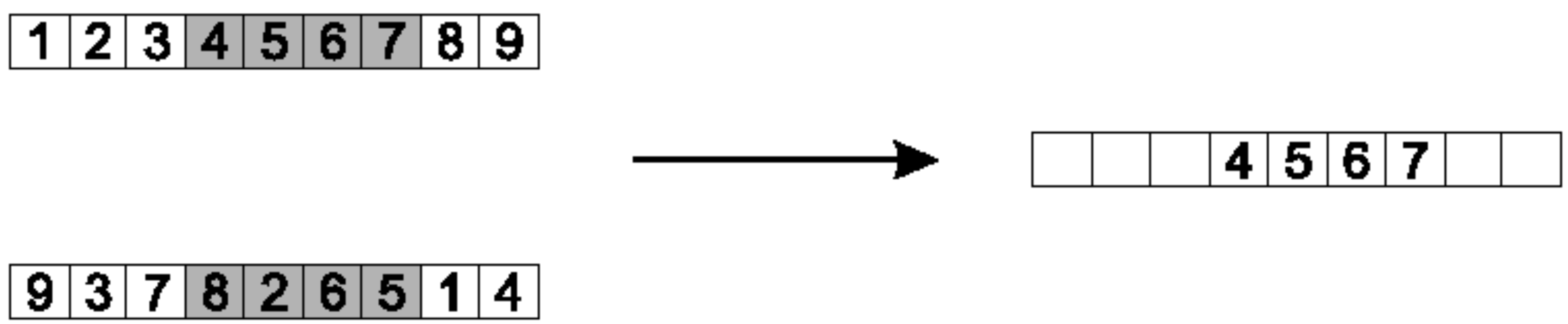
Choices	Element selected	Reason	Partial result
All	1	Random	[1]
2,5,4,9	5	Shortest list	[1 5]
4,6	6	Common edge	[1 5 6]
2,7	2	Random choice (both have two items in list)	[1 5 6 2]
3,8	8	Shortest list	[1 5 6 2 8]
7,9	7	Common edge	[1 5 6 2 8 7]
3	3	Only item in list	[1 5 6 2 8 7 3]
4,9	9	Random choice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

Order crossover

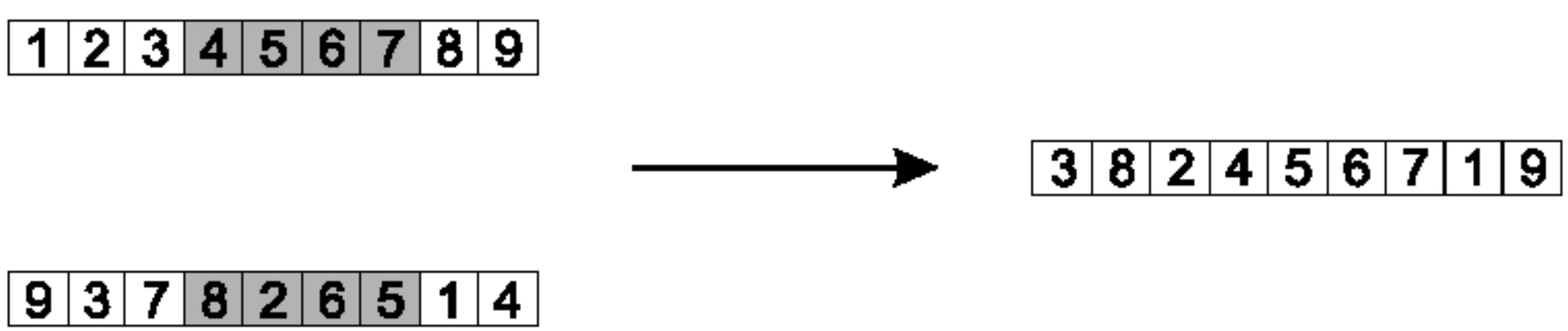
- Idea is to preserve relative ***order*** that elements occur
- Informal procedure:
  1. Chose an arbitrary part from the first parent
  2. Copy this part to the first child
  3. Copy the numbers that are not in the first part, to the first child:
     1. starting right from cut point of the copied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

Order crossover: example

- Copy randomly selected set from the first parent



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



Cycle crossover

- Two parents [1 2 3 4 5 6 7 8 9] and [9 3 7 8 2 6 5 1 4]
- Idea is to preserve absolute ***position*** in which elements occur
- Informal procedure:
  1. Start from the first unused position and allele of P1
  2. Look at the allele in the same position in P2
  3. Go to the position with the same allele in P1
  4. Add this allele to the cycle
  5. Repeat step 2 through 4 until you arrive at the first allele of P1

Cycle crossover: example

- Three cycles of elements