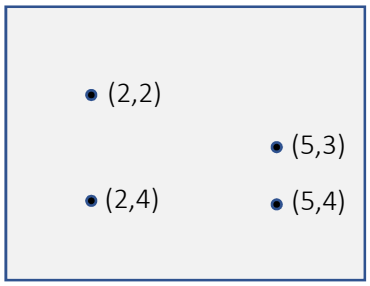
PCBMILL Problem Specification

An important step in the manufacture of PCBs (printed circuit boards) is computer numerical control (CNC) milling (i.e. drilling). This is where tiny holes are drilled in PCBs according to customer specifications. These holes allow the creation of *vias.* This is where two electrically conductive layers on the PCB that are separated by an insulator can be connected by drilling a hole through the insulator and lining the hole with conductive material. Holes also allow the mounting of electronic components on the PCB.

It takes time to move the drill-head of a PCB mill from one drill point on the PCB to another, and this time cost should be minimised in order to maximise production. Assume, for this *pcbmill* problem, the length of time to move the drill-head depends only on the distance of the move. An optimal drill-head movement plan should minimise the total distance the drill-head has to move between drill points. The movement plan could be followed in one direction for the first PCB board, and then performed in the reverse direction for the next PCB board, and so on.

The data file for the *pcbmill* problem is a series of input vectors, where each vector contains the coordinates of a drill point. Each input vector should therefore contain two *vector-elements*. As a simple example, if a PCB board had the following drill points:



… the input file could be:

```
InputVector:0(2,2)
InputVector:1(2,4)
InputVector:2(5,3)
InputVector:3(5,4)
```

A possible drill-head movement plan for the above example could be: (5,3),(2,4),(2,2),(5,4). This plan could be described by simply listing the input *vector-number* for each coordinate: 2,1,0,3.

Note that at least one, and probably multiple input vectors is read for any given *pcbmill* problem, and all chromosomes in the population are candidate solutions for the same *pcbmill* problem.

The task of *pcbmill* is to try and determine a drill-head movement plan that minimises the total distance that the drill-head moves (and thus the time required to perform the movement plan).

PCBMILL Chromosome representation

The length of a *pcbmill* chromosome equals the number of drill point coordinates (i.e. the number of input vectors in the input file). The chromosome is a sequence of input vector *vector-numbers*, each representing a drill point.

For example, a *pcbmill* chromosome for the drill-head movement plan (5,3),(2,4),(2,2),(5,4) using the above input vector data would be:

2	1	0	3

Each element of a *pcbmill* chromosome is called an *allele*.

Valid values for a *pcbmill* allele are positive integers from the set of input vector *vector-numbers*, where each *vector-number* is represented exactly once in the chromosome.

Function for creating a random PCBMILL chromosome

The function for creating a 'random' *pcbmill* chromosome is `create_pcbmill_chrom`

The function should create a chromosome of length *N*, where *N* is the number of input vectors (i.e. drill points), and each input vector *vector-number* is represented once in the chromosome, in some 'random' order.

PCBMILL Evaluation function

The *pcbmill* evaluation function calculates the raw score for a *pcbmill* chromosome.

The raw score for a *pcbmill* chromosome is the sum of the distances between the drill points. The distance between two points (x_1,y_1) and (x_2,y_2) is $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

So, for example, given the *pcbmill* chromosome

2	1	0	3

… and input vectors of

```
InputVector:0(2,2)
InputVector:1(2,4)
InputVector:2(5,3)
InputVector:3(5,4)
```

.. the raw score of the chromosome would be approximately 8.767

The function for implementing *pcbmill* evaluation is `eval_pcbmill`

Mutation operation for PCBMILL chromosome

The mutation operation must always result in a legal chromosome. Just changing a *pcbmill* chromosome allele index to another (legal) allele value is not appropriate, as the resulting drill-head movement plan may skip a drill point or visit the same drill point more than once. Mutation must therefore 'randomly' change only the *order* of the drill points in the plan.

Mutation of a *pcbmill* chromosome involves selecting two allele indexes at 'random' and swapping the associated chromosome values.

For example, if allele indexes 1 and 3 were chosen on the following chromosome:

3	1	2	0

… the resulting mutated chromosome would be:

3	0	2	1

The function for implementing *pcbmill* mutation is `mutate_pcbmill`

Crossover operation for PCBMILL chromosomes

Crossover must always produce a valid drill-head movement plan. The following crossover process will be used for the *pcbmill* problem, where crossover of two 'parent' *pcbmill* chromosomes produces a new 'child' *pcbmill* chromosome as follows:

Assume two 'parent' *pcbmill* chromosomes, p1 and p2:

p1:

5	1	2	3	4	0

p2:

0	1	2	3	5	4

'Randomly' choose two allele index positions such that index1 <= index2 (in this example index1 = 2 and index2 = 4). Index positions 2..4 inclusive of p1 are copied to the child chromosome starting from child index position 0. The remaining child positions are filled by going through parent p2 from position 0, and finding and copying allele values that are not present in the child. For example, once the child has had alleles 2,3,4 copied into it, examination of position 0 of p2 shows the associated allele 0 is not present in the child, and this is copied over to the child. Examination of position 1 of p2 shows the associated allele 1 is not present in the child and so is copied to the child. Position 2 of p2 has the value 2, which is present in the child and so is not copied over. Similarly, for the value at position 3 of p2. The value at position 4 of p2 (5) is not present in the child, and so is copied over, and so on until the child is 'full'.

child:

2	3	4	0	1	5

The function for implementing *pcbmill* crossover is `crossover_pcbmill`