

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4
5 namespace _3P71_2
6 {
7     class Crossover
8     {
9         GeneticTS geneticTS;
10        public Crossover(GeneticTS geneticTS)
11        {
12            this.geneticTS = geneticTS;
13
14        }
15        /// <summary>
16        /// Set up function for UOX crossover. Chooses parents
17        /// </summary>
18        /// <param name="tourList">list to perform the cross overs on</param>
19        /// <returns>list containing new generation</returns>
20        public List<Tour> UOXCrossover(List<Tour> tourList)
21        {
22            int[] mask = new int[tourList[0].Path.Length];
23
24            //Generate bit mask
25            for (int i = 0; i < mask.Length; i++)
26            {
27                mask[i] = geneticTS.random.Next(2);
28            }
29
30            int crossOverCount = (int)(geneticTS.crossoverRate *      ↗
31                tourList.Count) / 2;
32            for (int i = 0; i < crossOverCount; i++)
33            {
34                int P1Index = 0;
35                int P2Index = 0;
36                while(P1Index == P2Index)
37                {
38                    P1Index = geneticTS.TournamentSelect(tourList);
39                    P2Index = geneticTS.TournamentSelect(tourList);
40                }
41                int[] ch1 = UOXLoop(mask, tourList[P1Index].Path, tourList      ↗
42                    [P2Index].Path);
43                int[] ch2 = UOXLoop(mask, tourList[P2Index].Path, tourList      ↗
44                    [P1Index].Path);
45
46                geneticTS.AddChild(tourList, ch1, P1Index);
47                geneticTS.AddChild(tourList, ch2, P2Index);
48            }
49            return geneticTS.Prioritize(tourList);
50        }
51    }
52 }
```

```

50     /// <summary>
51     /// Main loop for OUX, performs the magic
52     /// </summary>
53     /// <param name="mask"> mask to decide which parent to take from</param>
54     /// <param name="P1">First parent </param>
55     /// <param name="P2">Second parent</param>
56     /// <returns>the generated child</returns>
57     private int[] UOXLoop(int[] mask, int[] P1, int[] P2)
58     {
59         int[] child = new int[P1.Length];
60         child[0] = geneticTS.startCityIndex;
61         child[P1.Length - 1] = geneticTS.startCityIndex;
62
63         List<int> valuesNotInChild = new List<int>();
64         for (int i = 0; i < P1.Length; i++)
65         {
66             if (P2[i] != geneticTS.startCityIndex)
67             {
68                 valuesNotInChild.Add(P2[i]);
69             }
70         }
71
72         //Take from first parent
73         for (int i = 1; i < mask.Length - 1; i++)
74         {
75             if (mask[i] == 1)
76             {
77                 child[i] = P1[i];
78                 valuesNotInChild.Remove(P1[i]);
79             }
80         }
81         //Repair from second parent
82         for (int i = 1; i < mask.Length - 1; i++)
83         {
84             if (mask[i] == 0)
85             {
86                 child[i] = valuesNotInChild[0];
87                 valuesNotInChild.RemoveAt(0);
88             }
89         }
90         return child;
91     }
92     /// <summary>
93     /// Performs a partially mapped crossover
94     /// crossOvers is calculated to dramatically reduce the number of ➤
95     /// Math.Random calls that would be needed.
96     /// On average, it will do the same number of crossOvers as just doing ➤
97     /// Math.Random.Next(2) < crossoverRate
98     /// </summary>
99     /// <param name="tourList">List to perform the crossover on</param>
100    /// <returns>List containing the new generation</returns>
101    public List<Tour> PMXCrossover(List<Tour> tourList)

```

```
100     {
101         //Calculate min value based on elitism
102         int crossOvers = (int)(geneticTS.crossoverRate * tourList.Count) / 2;
103         for (int i = 0; i < crossOvers; i++)
104         {
105             int P1Index = 0;
106             int P2Index = 0;
107             while (P1Index == P2Index)
108             {
109                 P1Index = geneticTS.TournamentSelect(tourList);
110                 P2Index = geneticTS.TournamentSelect(tourList);
111             }
112
113             //Get parents
114             int[] P1 = (int[])tourList[P1Index].Path;//.Clone();
115             int[] P2 = (int[])tourList[P2Index].Path;//.Clone();
116
117
118             //Make children
119             int[] ch1 = new int[tourList[P1Index].Path.Length];
120             int[] ch2 = new int[tourList[P2Index].Path.Length];
121
122             //Make helper bool array for children
123             bool[] ch1UsedValues = new bool[tourList[0].Path.Length + 1];
124             bool[] ch2UsedValues = new bool[tourList[0].Path.Length + 1];
125             ch1UsedValues[0] = true;
126             ch2UsedValues[0] = true;
127
128
129             //Randomize cut size
130             int cutA = geneticTS.random.Next(1, ch1.Length / 2);
131             int cutB = geneticTS.random.Next(cutA, ch1.Length - 1);
132
133             //place crossed values into their spots
134             for (int k = cutA; k < cutB; k++)
135             {
136                 ch1[k] = P2[k];
137                 ch1UsedValues[P2[k]] = true;
138                 ch2[k] = P1[k];
139                 ch2UsedValues[P1[k]] = true;
140             }
141
142             ch1 = PMXRepair(P1, ch1, ch1UsedValues);
143             ch2 = PMXRepair(P2, ch2, ch2UsedValues);
144             geneticTS.AddChild(tourList, ch1, P1Index);
145             geneticTS.AddChild(tourList, ch2, P2Index);
146         }
147
148         return geneticTS.Prioritize(tourList);
149     }
150     /// <summary>
151     /// Repairs the given child using values from given parent
```

```
152     /// usedValues array is for dramatically speeding up computation time
153     /// </summary>
154     /// <param name="parent">array to use for repair</param>
155     /// <param name="child">array to be repaired</param>
156     /// <param name="usedValues">array of all values in parent, set to true ↗
157         if used and false if available</param>
158     /// <returns>repaired child</returns>
159     private int[] PMXRepair(int[] parent, int[] child, bool[] usedValues)
160     {
161         //Ensure start position is not compromised
162         child[0] = child[child.Length - 1] = geneticTS.startCityIndex;
163
164         for (int i = 1; i < parent.Length - 1; i++)
165         {
166             if (!usedValues[parent[i]])
167             {
168                 int index = System.Array.IndexOf(child, 0);
169                 child[index] = parent[i];
170                 usedValues[parent[i]] = true;
171             }
172         }
173         return child;
174     }
175 }
176 }
177 }
```