```csharp
using System;
using System.Collections.Generic;
using System.Linq;

namespace _3P71_2
{
    class Crossover
    {
        GeneticTS geneticTS;
        public Crossover(GeneticTS geneticTS)
        {
            this.geneticTS = geneticTS;

        }
        /// <summary>
        /// Set up function for UOX crossover. Chooses parents
        /// </summary>
        /// <param name="tourList">list to perform the cross overs on</param>
        /// <returns>list containing new generation</returns>
        public List<Tour> UOXCrossover(List<Tour> tourList)
        {
            int[] mask = new int[tourList[0].Path.Length];

            //Generate bit mask
            for (int i = 0; i < mask.Length; i++)
            {
                mask[i] = geneticTS.random.Next(2);
            }

            int crossOverCount = (int)(geneticTS.crossoverRate * tourList.Count) / 2;
            for (int i = 0; i < crossOverCount; i++)
            {
                int P1Index = 0;
                int P2Index = 0;
                while(P1Index == P2Index)
                {
                    P1Index = geneticTS.TournementSelect(tourList);
                    P2Index = geneticTS.TournementSelect(tourList);
                }

                int[] ch1 = UOXLoop(mask, tourList[P1Index].Path, tourList[P2Index].Path);
                int[] ch2 = UOXLoop(mask, tourList[P2Index].Path, tourList[P1Index].Path);

                geneticTS.AddChild(tourList, ch1, P1Index);
                geneticTS.AddChild(tourList, ch2, P2Index);
            }

            return geneticTS.Prioritize(tourList);
        }
        /// <summary>
        /// Main loop for OUX, performs the magic
        /// </summary>
        /// <param name="mask"> mask to decide which parent to take from</param>
        /// <param name="P1">First parent </param>
        /// <param name="P2">Second parent</param>
        /// <returns>the generated child</returns>
        private int[] UOXLoop(int[] mask, int[] P1, int[] P2)
        {
            int[] child = new int[P1.Length];
            child[0] = geneticTS.startCityIndex;
            child[P1.Length - 1] = geneticTS.startCityIndex;

            List<int> valuesNotInChild = new List<int>();
            for (int i = 0; i < P1.Length; i++)
            {
                if (P2[i] != geneticTS.startCityIndex)
                {
                    valuesNotInChild.Add(P2[i]);
                }
            }

            //Take from first parent
            for (int i = 1; i < mask.Length - 1; i++)
            {
                if (mask[i] == 1)
                {
                    child[i] = P1[i];
                    valuesNotInChild.Remove(P1[i]);
                }
            }
            //Repair from second parent
            for (int i = 1; i < mask.Length - 1; i++)
            {
                if (mask[i] == 0)
                {
                    child[i] = valuesNotInChild[0];
                    valuesNotInChild.RemoveAt(0);
                }
            }
            return child;
        }
        /// <summary>
        /// Performs a partially mapped crossover
        /// crossOvers is calculated to dramatically redice the number of
          Math.Random calls that would be needed.
        /// On average, it will do the same number of crossOvers as just doing
          Math.Random.Next(2) < crossoverRate
        /// </summary>
        /// <param name="tourList">List to perform the crossover on</param>
        /// <returns>List containing the new generation</returns>
        public List<Tour> PMXCrossover(List<Tour> tourList)
        {
            //Calculate min value based on elietism
            int crossOvers = (int)(geneticTS.crossoverRate * tourList.Count) / 2;
            for (int i = 0; i < crossOvers; i++)
            {
                int P1Index = 0;
                int P2Index = 0;
                while (P1Index == P2Index)
                {
                    P1Index = geneticTS.TournementSelect(tourList);
                    P2Index = geneticTS.TournementSelect(tourList);
                }

                //Get parents
                int[] P1 = (int[])tourList[P1Index].Path;//.Clone();
                int[] P2 = (int[])tourList[P2Index].Path;//.Clone();


                //Make children
                int[] ch1 = new int[tourList[P1Index].Path.Length];
                int[] ch2 = new int[tourList[P2Index].Path.Length];

                //Make helper bool array for children
                bool[] ch1UsedValues = new bool[tourList[0].Path.Length + 1];
                bool[] ch2UsedValues = new bool[tourList[0].Path.Length + 1];
                ch1UsedValues[0] = true;
                ch2UsedValues[0] = true;


                //Randomize cut size
                int cutA = geneticTS.random.Next(1, ch1.Length / 2);
                int cutB = geneticTS.random.Next(cutA, ch1.Length - 1);

                //place crossed values into their spots
                for (int k = cutA; k < cutB; k++)
                {
                    ch1[k] = P2[k];
                    ch1UsedValues[P2[k]] = true;
                    ch2[k] = P1[k];
                    ch2UsedValues[P1[k]] = true;
                }

                ch1 = PMXRepair(P1, ch1, ch1UsedValues);
                ch2 = PMXRepair(P2, ch2, ch2UsedValues);
                geneticTS.AddChild(tourList, ch1, P1Index);
                geneticTS.AddChild(tourList, ch2, P2Index);
            }

            return geneticTS.Prioritize(tourList);
        }
        /// <summary>
        /// Repairs the given child using values from given parent
        /// usedValues array is for dramatically speeding up computation time
        /// </summary>
        /// <param name="parent">array to use for repair</param>
        /// <param name="child">array to be repaired</param>
        /// <param name="usedValues">array of all values in parent, set to true 
          if used and false if available</param>
        /// <returns>repaired child</returns>
        private int[] PMXRepair(int[] parent, int[] child, bool[] usedValues)
        {
            //Ensure start position is not compromised
            child[0] = child[child.Length - 1] = geneticTS.startCityIndex;


            for (int i = 1; i < parent.Length - 1; i++)
            {
                if (!usedValues[parent[i]])
                {
                    int index = System.Array.IndexOf(child, 0);
                    child[index] = parent[i];
                    usedValues[parent[i]] = true;
                }
            }
            return  child;
        }
    }
}

```