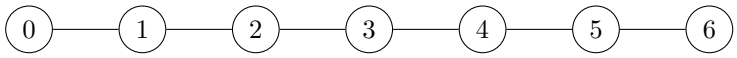
Prescriptive Analytics, Hausaufgabe 3

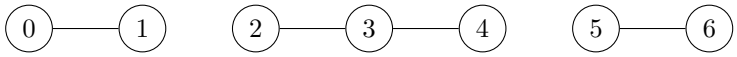
Henry Haustein

Aufgabe 1: Theorie I: TwoEdgeExchange Move

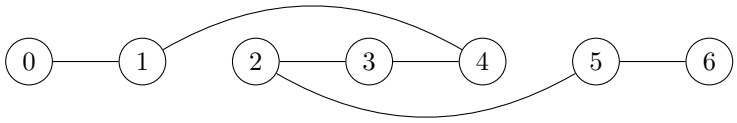
originale Reihenfolge:



Kanten entfernt:



neu verbunden:



Führt zu

$$M(\text{2-EdgeExchange}, s, 1, 4) = [0, 1, 4, 3, 2, 5, 6]$$

Aufgabe 2: Theorie II: TwoEdgeExchange Move

(a) Wären i und j nur 1 voneinander entfernt, so wären sie benachtbart $\rightarrow$ kein TwoEdgeExchange-Move
Wären i und j nur 2 voneinander entfernt, so gibt es nur ein Element in der "Mitte" und der TwoEdgeExchange-Move entspricht einem Swap-Move.
$\Rightarrow$ i und j müssen mindestens 3 voneinander entfernt sein

(b) In einer Sequenz gibt es $n-1$ Kanten.
Wenn ich die erste Kante entferne, dann habe ich noch $n-1-2 = n-3$ Kanten, die ich auch entfernen könnte.
Wenn ich die zweite Kante entferne, dann habe ich noch $n-4$ Kanten, die ich auch entfernen könnte.
Wenn ich die dritte Kante entferne, dann habe ich noch $n-5$ Kanten, die ich auch entfernen könnte.
Wenn ich die vierte Kante entferne, dann habe ich noch $n-5$ Kanten ($n-6$ Kanten rechts davon, eine mögliche Kante links davon), die ich auch entfernen könnte.
Wenn ich die fünfte Kante entferne, dann habe ich noch $n-5$ Kanten ($n-7$ Kanten rechts davon, zwei mögliche Kanten links davon), die ich auch entfernen könnte.
...
Wenn ich die $(n-3)$-te Kante entferne, dann habe ich noch $n-5$ Kanten ($n-6$ links, eine rechts), die ich entfernen könnte.
Wenn ich die $(n-2)$-te Kante entferne, dann habe ich noch $n-5$ Kanten, die ich links davon entfernen

könnte.
Wenn ich die $(n-1)$-te Kante entferne, dann habe ich noch $n-4$ Kanten, die ich links davon entfernen könnte.
Wenn ich die n-te Kante entferne, dann habe ich noch $n-3$ Kanten, die ich links davon entfernen könnte.
$\Rightarrow 2(n-3)+2(n-4)+(n-4)(n-5)$ Möglichkeiten, aber wenn $i > j$, dann ändert sich die Permutation nicht
$\Rightarrow |N(\text{2-EdgeExchange}, s)| = (n-3)+(n-4)+\frac{1}{2}(n-4)(n-5)$

Aufgabe 3: Implementierung: TwoEdgeExchange-Move

Hinzugefügter Code in `Neighborhood.py`

```
1  class TwoEdgeExchangeMove:
2    def __init__(self, initialPermutation, indexA, indexB):
3      self.Permutation = list(initialPermutation)
4      list1 = initialPermutation[:indexA]
5      list2 = initialPermutation[indexA:indexB]
6      list3 = initialPermutation[indexB:]
7      self.Permutation = list1 + list(reversed(list2)) + list3
8
9  class TwoEdgeExchangeNeighborhood(BaseNeighborhood):
10   def __init__(self, inputData, initialPermutation,
         evaluationLogic, solutionPool):
11     super().__init__(inputData, initialPermutation,
           evaluationLogic, solutionPool)
12     self.Type = "TwoEdgeExchange"
13
14   def DiscoverMoves(self):
15     for i in range(1, len(self.Permutation)):
16       for j in range(1, len(self.Permutation)):
17         if j-i >= 3 and j > i:
18           twoEdgeExchangeMove = TwoEdgeExchangeMove(self.
                 Permutation, i, j)
19           self.Moves.append(twoEdgeExchangeMove)
```

Aktualisierter Code in `ImprovementAlgorithm.py`

```
1  def CreateNeighborhood(self, neighborhoodType, bestCurrentSolution
       ):
2    if neighborhoodType == 'Swap':
3      return SwapNeighborhood(self.InputData, bestCurrentSolution.
           Permutation, self.EvaluationLogic, self.SolutionPool)
4    elif neighborhoodType == 'Insertion':
5      return InsertionNeighborhood(self.InputData,
           bestCurrentSolution.Permutation, self.EvaluationLogic,
           self.SolutionPool)
6    elif neighborhoodType == 'TwoEdgeExchange':
7      return TwoEdgeExchangeNeighborhood(self.InputData,
           bestCurrentSolution.Permutation, self.EvaluationLogic,
           self.SolutionPool)
8    else:
```

```
9       raise Exception(f"Neighborhood type {neighborhoodType} not
            defined.")
```