Prescriptive Analytics, Hausaufgabe 4

Henry Haustein

Aufgabe 1: Zeitmessung

```python
from Solver import *
import timeit

data = InputData("VFR20_10_1_SIST.json")
solver = Solver(data, 1008)

localSearch = IterativeImprovement(data, 'BestImprovement', ['Insertion'])
iteratedGreedy = IteratedGreedy(
  inputData = data,
  numberJobsToRemove = 2,
  baseTemperature = 0.8,
  maxIterations = 10,
  localSearchAlgorithm = localSearch)

start = timeit.default_timer()
solver.RunLocalSearch(
  constructiveSolutionMethod='NEH',
  algorithm=iteratedGreedy)
ende = timeit.default_timer()
print(f"Runtime: {ende - start} seconds")
```

Aufgabe 2: Stoppkriterium

Code im Jupyter-Notebook:

```python
from Solver import *

data = InputData("VFR20_10_1_SIST.json")
solver = Solver(data, 1008)

localSearch = IterativeImprovement(data, 'BestImprovement', ['Insertion'])
iteratedGreedy = IteratedGreedy(
  inputData = data,
  numberJobsToRemove = 2,
  baseTemperature = 0.8,
  maxIterations = 10,
  maxIterationsWithoutImprovement = 2,
  localSearchAlgorithm = localSearch)

solver.RunLocalSearch(
  constructiveSolutionMethod='NEH',
  algorithm=iteratedGreedy)
```

Aktualisierter Code in ImprovementAlgorithm.py (__init__-Methode)

```python
def __init__(self, inputData, numberJobsToRemove, baseTemperature, maxIterations, maxIterationsWithoutImprovement = 1000000, localSearchAlgorithm = None):
  super().__init__(inputData)

  self.NumberJobsToRemove = numberJobsToRemove
  self.BaseTemperature = baseTemperature
  self.MaxIterations = maxIterations
  self.MaxIterationsWithoutImprovement = maxIterationsWithoutImprovement

  if localSearchAlgorithm is not None:
    self.LocalSearchAlgorithm = localSearchAlgorithm
  else:
    self.LocalSearchAlgorithm = IterativeImprovement(self.InputData, neighborhoodTypes=[]) # IterativeImprovement without a neighborhood does not modify the solution
```

Run-Methode (veränderte Zeilen: 6, 7, 15 und 24)

```python
1  def Run(self, currentSolution):
2    currentSolution = self.LocalSearchAlgorithm.Run(currentSolution)
3
4    currentBest = self.SolutionPool.GetLowestMakespanSolution().Makespan
5    iteration = 0
6    withoutImprovement = 0
7    while iteration < self.MaxIterations and withoutImprovement < self.MaxIterationsWithoutImprovement:
8      removedJobs, partialPermutation = self.Destruction(currentSolution)
9      newSolution = self.Construction(removedJobs, partialPermutation)
10
11     newSolution = self.LocalSearchAlgorithm.Run(newSolution)
12
13     if newSolution.Makespan < currentSolution.Makespan:
14       currentSolution = newSolution
15       withoutImprovement = 0
16
17       if newSolution.Makespan < currentBest:
18         print(f'New best solution in iteration {iteration}: {currentSolution}')
```

```
        self.SolutionPool.AddSolution(currentSolution)
        currentBest = newSolution.Makespan

    elif self.AcceptWorseSolution(currentSolution.Makespan, newSolution.Makespan):
      currentSolution = newSolution
      withoutImprovement += 1

    iteration += 1

  return self.SolutionPool.GetLowestMakespanSolution()
```

Aufgabe 3: Rechenstudie

Sammeln der Daten

```
from Solver import *
import timeit

data = InputData("VFR20_10_1_SIST.json")

rows = []
for neighboorhood in ["None", "Insertion", "TaillardInsertion"]:
  for numberJobsToRemove in [2, 3, 4]:
    for baseTemperature in [0.5, 1]:
      for maxIterations in [1, 10]:
        for iteration in range(3):
          seed = numberJobsToRemove * maxIterations * iteration
          solver = Solver(data, seed)
          if neighboorhood == "None":
            localSearch = None
          else:
            localSearch = IterativeImprovement(data, 'BestImprovement', [neighboorhood])
          iteratedGreedy = IteratedGreedy(
            inputData = data,
            numberJobsToRemove = numberJobsToRemove,
            baseTemperature = baseTemperature,
            maxIterations = maxIterations,
            maxIterationsWithoutImprovement = 100,
            localSearchAlgorithm = localSearch)

          start = timeit.default_timer()
          solver.RunLocalSearch(
            constructiveSolutionMethod='NEH',
            algorithm=iteratedGreedy)
          ende = timeit.default_timer()

          # build dict
          row = {}
          row["Iteration"] = iteration
```

```
            row["LocalSearch"] = neighboorhood
            row["NumberOfJobsToRemove"] = numberJobsToRemove
            row["BaseTemperature"] = baseTemperature
            row["MaxIterations"] = maxIterations
            row["Seed"] = seed
            row["Makespan"] = solver.SolutionPool.
                GetLowestMakespanSolution().Makespan
            row["Runtime"] = ende - start
            rows.append(row)
```

Verarbeitung mittels Pandas

```
import pandas as pd
df = pd.DataFrame(rows)
df
df.groupby(["LocalSearch", "NumberOfJobsToRemove", "
    BaseTemperature", "MaxIterations"]).mean()
```