Philipps Universität Marburg

Philipps-Universität Marburg
Fachbereich Mathematik und Informatik

Prof. Gabriele Taentzer
Alexander Lauer

24. August 2023

Programmier-Praktikum

1 Das Rucksack-Problem

Ein beliebtes Beispiel für Optimierungsprobleme ist das sogenannte Rucksack-Problem[1]. Für dieses sollen Sie nun eine zum Framework aus Aufgabe 1 passende Implementierung schreiben, die es ermöglicht, konkrete Instanzen des Rucksackproblems mithilfe des Frameworks zu lösen.

Die Implementierung von Optimierungsproblemen soll dabei in einem separaten IntelliJ-Modul erfolgen. Zudem soll dieses, genau wie das Framework, als Modul im Sinne von JPMS entwickelt werden. Es ist dabei wichtig zu beachten, dass JPMS und das IntelliJ-Modulsystem zwei unabhängige Systeme zur Verwaltung von Modulen darstellen. Um ein korrektes Zusammenspiel zwischen Framework- und Problemimplementierung zu ermöglichen, müssen Abhängigkeiten in beiden Systemen korrekt spezifiziert werden!

Fügen Sie Ihrem Projekt für die Implementierung des Rucksackproblems zunächst ein neues IntelliJ-Java-Modul *ga.problems* hinzu. Legen Sie in diesem Modul ein Paket *ga.problems.knapsack* an, welches später alle Klassen des Problems enthalten soll. Die im Folgenden genannten Klassen sollen jeweils die entsprechenden Interfaces bzw. abstrakten Klassen der vorgegebenen Schnittstelle implementieren.

1.1 Problem und Lösung [4 Punkte]

Implementieren Sie die Klassen *KnapsackProblem* und *KnapsackSolution*. Die *Problem*-Klasse soll generelle Informationen über das Rucksackproblem (z.B. Kapazität des Rucksacks) halten und über *createNewSolution()* neue *KnapsackSolutions* anlegen können. Beim Erstellen einer neuen Lösung soll dabei solange ein zufälliger Gegenstand in den Rucksack gepackt werden, bis keiner der noch übrigen Gegenstände mehr hinein passt. Für den Fall, dass alle im Problem spezifizierten Gegenstände zu schwer für den Rucksack sind, also kein einziger Gegenstand hineingepackt werden kann, soll eine *NoSolutionException* geworfen und die Optimierung abgebrochen werden.

Die *Solution*-Klasse soll lösungsspezifische Informationen halten (z.B. welche Gegenstände sich im Rucksack befinden). Gegebenenfalls macht es Sinn, zusätzliche (redundante) Informationen zu halten, um die Performanz des Programms zu verbessern.

Selbstverständlich können Sie auch weitere Klassen anlegen, falls Sie diese zur Modellierung von Problem und Lösung benötigen.

[1] https://de.wikipedia.org/wiki/Rucksackproblem

1.2 Fitness [2 Punkte]

Implementieren Sie die Klasse *KnapsackFitnessEvaluator*, welche die Fitness aller Lösungen einer Population berechnet. Nutzen Sie zur Berechnung der Fitness einer Lösung Java-Streams und Lambda-Ausdrücke!

1.3 Mutationsoperator [5 Punkte]

Implementieren Sie für das Rucksackproblem *einen* evolutionären Operator *KnapsackMutation*, der zufällig auf eine der folgenden Arten eine neue Lösung erzeugt:

- Ein zufälliger Gegenstand wird aus dem Rucksack entfernt.
- Aus den Gegenständen, die noch nicht gewählt wurden und noch in den Rucksack passen, wird ein zufälliger gewählt und in den Rucksack gelegt.

Wichtig ist dabei, dass hierbei die ursprüngliche Lösung nicht verändert wird. Veränderungen sollten also auf einer Kopie durchgeführt werden. Verwenden Sie zur Erzeugung von Kopien einer Lösung einen konkreten Copy-Konstruktor für *KnapsackSolutions*.

Achten Sie auch darauf, dass zur Erstellung einer neuen Lösung immer exakt *eine* der genannten Veränderungen durchgeführt wird. Sollte keine der beiden Varianten anwendbar sein, ist dies als Fehler (*EvolutionException*) zu behandeln und die Optimierung insgesamt abzubrechen.

Es soll nur *ein* evolutionärer Operator entstehen, der beide Varianten integriert. Verwenden Sie innere Klassen, um die unterschiedlichen Varianten der Mutation zu implementieren.

Hinweis

Die Teilaufgaben sind weitestgehend unabhängig voneinander lösbar. Für die Erstellung der Fitnessfunktion und des Mutationsoperators benötigt man jedoch zumindest eine minimale Implementierung (Stub) der *KnapsackSolution*.

2 Ausführung einer Optimierung [2 Punkte]

Erstellen Sie eine Klasse *ConcreteProblem*, die eine Optimierung anhand eines konkreten *KnapsackProblems* demonstriert. Das Beispielproblem soll dabei aus einem Rucksack mit einer Kapazität von 11 Gewichtseinheiten und zehn Gegenständen bestehen:

- `g1`(Gewicht:5, Wert:10)
- `g2`(Gewicht:4, Wert:8)
- `g3`(Gewicht:4, Wert:6)
- `g4`(Gewicht:4, Wert:4)
- `g5`(Gewicht:3, Wert:7)
- `g6`(Gewicht:3, Wert:4)
- `g7`(Gewicht:2, Wert:6)
- `g8`(Gewicht:2, Wert:3)
- `g9`(Gewicht:1, Wert:3)
- `g10`(Gewicht:1, Wert:1)

Initialisieren Sie in *ConcreteProblem* das Problem und führen Sie, mit einer Population der Größe 4, eine Optimierung über 10 Iterationen durch. Verwenden Sie dabei *TopK-Survival* und, sofern Sie diese implementiert haben, *TournamentSelection*. Geben Sie auf der Konsole für jede Lösung der finalen Population deren Fitness und die im Rucksack enthaltenen Gegenstände aus. Vergleichen Sie die Ergebnisse für unterschiedliche Werte des Parameters k des *TopKSurvival*.