Beladung von LKW - Optimierung

Projektarbeit

des Studiengangs Informatik

an der Dualen Hochschule Baden-Württemberg Stuttgart

von

5320806, 1980527

20.12.2021

Bearbeitungszeitraum	Wintersemester 2021/22
Matrikelnummer, Kurs	5320806, 1980527 , INF19B
Dozent	Prof. Dr. Dirk Reichardt

Inhaltsverzeichnis

1 Einleitung

Evolutionäre Algorithmen werden in verschiedenen Bereichen verwendet. Die Idee dahinter ist es, natürliche Evolution in Software nachzubilden und die Individuen an die Umgebung, beziehungsweise an eine bestimmte Herausforderung, anzupassen.

Die Herausforderung im vorliegenden Projekt ist das Beladen von LKW's. Gegeben ist eine Liste von Aufträgen und eine Liste von verfügbaren LKW's. Es soll bestimmt werden, welcher LKW welche Aufträge zu übernehmen hat, damit sich möglichst viel Geld verdienen lässt. Dabei ist zu beachten, dass nicht jeder LKW jeden Auftrag übernehmen kann, da die LKW's beschränkte Kapazitäten bezügliche Ladegewicht und Ladevolumen haben. Außerdem sind bestimmte Aufträge zu priorisieren, da bei verspäteter Ablieferung Strafen drohen. Genauso locken Bonusvergütungen bei rechtzeitiger Belieferung.

Die grundlegenden Techniken evolutionärer Algorithmen, sowie deren konkrete Umsetzung zur Lösung des genannten Problems, sind Teil dieser Projektarbeit.

2 Schritte der Evolution

Nachfolgend werden gängige Strategien zur Modellierung evolutionärer Entwicklungen beschrieben. In Zusammenhang damit wird auch erklärt, wie diese bei dem gegebenen Optimierungsproblem anwendbar sind.

2.1 Fitness

Ein zentrales Maß für die Güte einzelner Individuen während der Evolution ist deren Fitness. In dem gegeben Problem ist die Fitness das Geld, welches sich mit einer Beladung verdienen lässt. Das Ziel der Entwicklung ist es dabei die Fitness über den Prozess hinweg zu maximieren.

2.2 Erstellen einer zufälligen Anfangspopulation

Zunächst muss mit einer initialen Population gestartet werden. Diese muss möglichst zufällig erstellt werden. In vielen Fällen reicht hier eine naive zufällige Zuweisung von Genen, da keine zusätzlichen Bedingungen vorliegen. Da bei dem vorliegenden Problem, aber die Randbedingungen der maximalen Kistenanzahl und der maximalen Gewichtskapazität gegeben sind, ist eine rein zufällige Zuweisung nicht der beste Ansatz. Außerdem muss beachtet werden, dass jeder Auftrag genau ein Mal zugewiesen wird. Nachfolgend werden Beladungen, die eine oder mehrere dieser Bedingungen verletzten, als invalide Beladungen bezeichnet.

Eine Strategie wäre es auch invalide Beladungen in den Prozess einzubeziehen und diese in der Selektierungphase, siehe Abschnitt 2.5, maximal schlecht zu bewerten, so dass sie entfernt werden. Somit muss man jedoch eine sehr große Anfangspopulation bereitstellen, da bei zufälliger Zuweisung nur sehr selten eine valide Beladung entsteht. Das ist sowohl speicher- als auch rechenintensiv.

Deshalb wurde sich dafür entschieden, die Populationsgenerierung so zu gestalten, dass stets nur valide Beladungen entstehen können.

2.3 Kreuzung

Nach dem Erstellen einer initialen Population wird mit einer Wahrscheinlichkeit $P_{kreuzung}$ eine Kreuzung von Individuen vorgenommen. Gängige Ansätze hierfür sind zum Beispiel ein *One-Point-Crossover* oder ein *Two-Point-Crossover*. Dabei werden die Gene von zwei benachbarten Individuen ab einem oder zwei zufälligen Punkten vertauscht. Auch hier tritt jedoch wieder das Problem auf, dass invalide Beladungen entstehen, wenn man solch einen Ansatz verfolgt.

Deswegen wurde eine eigene Strategie entworfen, welche beide Beladungen, durch Lernen von der jeweils anderen, verbessern soll. Prinzipiell kann davon ausgegangen werden, dass es suboptimal ist, wenn ein LKW keinen Auftrag zugewiesen bekommt. Deshalb wird in solche einem Fall in der anderen Beladung nachgeschaut, welche Aufträge prinzipiell von diesem LKW erledigt werden könnten. Diese Aufträge werden dann von anderen LKW's zu diesem verschoben. Das geschieht nacheinander in beide Richtungen, wodurch beiden Beladungen voneinander lernen. Da beide Beladungen im Voraus valide waren und die Aufträge bereits als machbar für den jeweiligen LKW eingestuft wurden, bleibt auch danach die Validität beider Beladungen bestehen.

2.4 Mutation

Da das Austauschen von Genen zwischen zwei Individuen gewisse Grenzen nicht überschreiten kann, wird außerdem mit einer Wahrscheinlichkeit $P_{mutation}$ eine Mutation einzelner Individuen vorgenommen. Warum eine Mutation nötig ist, wird an folgendem Beispiel klar. Angenommen, die Gene der Individuen bestehen ausschließlich aus 0 und 1 und es sind zwei Individuen $i_1 = [0,1,0]$ und $i_2 = [0,0,1]$ gegeben. Dann lässt sich durch eine Kreuzung jedweder Art auch immer nur ein Individuum $i_p = [0, ...]$ erzeugen. In solch einem Fall wäre zum Beispiel eine Mutation durch Negieren zufällig ausgewählter Bits sinnvoll. Beispiel sinngemäß entnommen aus [Yan14].

In dem vorliegenden Problem, muss jedoch auch wieder darauf geachtet werden, dass durch eine Mutation keine invalide Beladung entsteht. Deshalb wurde folgende Technik angewendet:

Die einzelnen Aufträge für jeden LKW werden in ihrer Reihenfolge permutiert. Das kann dazu führen, dass Bonuszeiten beziehungsweise Strafzeiten besser eingehalten werden. Außerdem wird ein zufälliger Auftrag jedes LKW's zu einem zufällig ausgewählten anderen LKW geschoben, falls das nicht zu einer invaliden Beladung führt. Somit können einzelne Individuen sich zufällig in andere Richtungen entwickeln.

2.5 Auswahl der nächsten Generation

Nachdem Kreuzung und Mutation erfolgt sind werden solche Individuen erwählt, welche in der nächsten Generation fortbestehen sollen. Dabei können alle erwählt werden, so dass die Populationsgröße gleich bleibt oder nur ein paar, so dass sich die Population je Zyklus entweder verkleinert oder durch Individuen anderer Quelle angereichert wird.

Bei dem gegebenen Problem wurde sich dafür entschieden die schlechtesten 25% der Population durch neue, zufällig generierte Indivduen zu ersetzen. Dabei wird davon ausgegangen, dass sich diese Individuen auch über viele Generationen und durch wiederholte Anwendung von Kreuzung und Mutation nicht signifikant verbessern werden.

2.6 Terminierung

Eine Terminierungsbedingung bestimmt, wie viele Generationen simuliert werden. Das kann entweder direkt durch Angabe einer Zahl von Generationen gemacht werden oder zum Beispiel durch eine mindestens zu erreichende Fitness. Auch eine Mindestverbesserung zwischen aufeinander folgenden Generationen kann benutzt werden, d.h. das abgebrochen wird, sobald die Differenz der Fitness einzelner Generationen ein bestimmtes Maß unterschreitet.

Da eine Obergrenze oder einen minimale Verbesserung für die Beladung im allgemeinen Fall nicht bestimmbar ist, wurde sich im vorliegenden Fall für eine Anzahl von Generationen $n = 50$ entschieden. Dabei kann eine schnelle Auswertung und eine gute Optimierung erreicht werden.

3 Umsetzung

Zur Umsetzung der beschriebenen Techniken wurde sich für die Programmiersprache Python 3.9 entschieden. Ein Grund hierfür ist die schnelle Entwicklung und das gute Vorwissen der Teammitglieder. Außerdem verfügt Python über viele, gute Bibliotheken, welche die Arbeit mit vielen Themen erleichtern.

3.1 Erstellte Klassen

Zum Verständnis werden hier die erstellten Klassen und ihre Bedeutungen erklärt. Die Klasse *Truck* repräsentiert einen einzelnen LKW, seine Kennnummer, sowie seine maximale Beladung bezüglich Kisten und Masse. Je ein *Truck* gehört zur Klasse *Route*. Eine Route ist eine Liste von Auftragsnummern, die der zugehörige LKW in der gegebenen Reihenfolge abarbeiten soll. Das zu optimierende Individuum ist ein Objekt der Klasse *Loading*. Eine solche Beladung beinhaltet eine Route für jeden LKW und eine Liste aller Aufträge. Aufträge sind durch die Klasse *Assignment* repräsentiert. Mit den Routen und den Aufträgen kann anschließend berechnet werden, wie viel Geld sich mit der Beladung verdienen lässt. Dazu wird über die einzelne Routen iteriert. Für jede Route wird berechnet, zu welchem Zeitpunkt welcher Auftrag erfüllt wird und ob daraus Straf- oder Bonuszahlungen entstehen.

3.2 Distributed Evolutionary Algorithms in Python - DEAP

DEAP ist ein Framwork evolutionärer Algorithmen, welches besonderen Wert auf Transparenz und und schnelles Entwickeln legt [For+12].

Neben vielen verfügbaren Funktionen bezüglich Mutation, Kreuzung oder Auswahl stellt DEAP Möglichkeiten bereit, eigene Datentypen zu erstellen und benötigte Funktionen in

einer so genannten *Toolbox* zu sammeln. So wurde ein *Individual*-Typ erstellt, welcher von *Loading* erbt und zusätzlich ein zu maximierendes *Fitness*-Objekt als Feld hält.

Dann werden übliche Funktionen wie *mate*, *mutate*, *select*, *evaluate* in der Toolbox registriert. Das hat unter anderem auch den Vorteil, dass sich die Funktionen, die hinter den einzelnen Bezeichnungen stecken, durch eine Änderung an der Toolbox-Registrierung ändern lassen, statt den Aufruf an jeder Stelle im Code anpassen zu müssen.

Ein weiteres Feature der Bibliothek, welches genutzt wurde, ist die *Hall of Fame*. Hierin werden die n besten Individuen über alle Generationen hinweg gespeichert. Im vorhanden Code ist $n = 3$.

3.3 Parameter

Die Ergebnisse hängen wesentlich von den verschiedenen Parametern ab. Bei der Anzahl der Generationen und der anfänglichen Populationsgröße gilt: Je größer, desto besser die Ergebnisse. Allerdings geht damit auch eine Erhöhung des Speicher-, Rechen- und Zeitbedarfs einher. Somit muss ein Kompromiss gefunden werden, der beide Vorteile widerspiegelt.

Daneben gibt es folgenden andere Parameter: $P_{kreuzung}$, siehe Abschnitt 2.3, und $P_{mutation}$, siehe Abschnitt 2.4, welche in Betracht auf [YCT15] gesetzt wurden.

Nach diesen Abwägungen ergab sich für die Parameter folgende Konfiguration:

- Anfängliche Populationsgröße: 1000
- Generationen: 50
- $P_{kreuzung}$: 0.8
- $P_{mutation}$: 0.01

4 Ergebnisse

Mit den in 3.3 definierten Parametern ergab sich folgende Änderung zwischen anfänglicher und abschließender Population:

Zeitpunkt	Maximale Fitness	Durchschnittliche Fitness
Anfang	80270	66445.92
Ende	86970	68643.60

Durch die Ausgabe der Hall of Fame lassen sich außerdem direkt die besten Individuen und deren Konfiguration erkennen, so zum Beispiel folgende für die erste LKW- bzw. Auftragsdatei:

```
Truck 1: 15 10 21 22 27
Truck 2: 26 16 25 2 23 14
Truck 3: 19 17 24 11
Truck 4: 12 8 1 28
Truck 5: 18 29
Truck 6: 30
Truck 7: 13 9 5 4 6
Truck 8: 7 3 20
```

Diese Konfiguration bringt es auf insgesamt 86970€.

5 Fazit

Es konnte ein evolutionärer Algorithmus entwickelt werden, welcher in der Lage ist, die Beladung der LKW's immer weiter zu verbessern.

Dabei wurden die Konzepte des Evolutionary Computing verstanden und umgesetzt. Es wurden eigene Kreuzungs- und Mutationsfunktionen entworfen und implementiert und es wurde ein gängiges Framework eingesetzt und verstanden.

Eine Verbesserung wäre eventuell in der Anpassung der Parameter zu finden, sowie in der Entwicklung beziehungsweise Verwendung von besseren Funktionen für die einzelnen Teilschritte. Dabei könnten diese performanter sein oder ganz prinzipiell zu einem besseren Ergebnis führen.

Anhang

Literatur

[For+12] Félix-Antoine Fortin u. a. „DEAP: Evolutionary Algorithms Made Easy“. In: *Journal of Machine Learning Research* 13 (2012), S. 2171–2175.

[Yan14] Xin-She Yang. „Chapter 5 - Genetic Algorithms“. In: *Nature-Inspired Optimization Algorithms*. Hrsg. von Xin-She Yang. Oxford: Elsevier, 2014, S. 77–87. ISBN: 978-0-12-416743-8. DOI: https://doi.org/10.1016/B978-0-12-416743-8.00005-1. URL: https://www.sciencedirect.com/science/article/pii/B9780124167438000051.

[YCT15] Xin-She Yang, Su Fong Chien und Tiew On Ting. „Chapter 1 - Bio-Inspired Computation and Optimization: An Overview“. In: *Bio-Inspired Computation in Telecommunications*. Hrsg. von Xin-She Yang, Su Fong Chien und Tiew On Ting. Boston: Morgan Kaufmann, 2015, S. 1–21. ISBN: 978-0-12-801538-4. DOI: https://doi.org/10.1016/B978-0-12-801538-4.00001-X. URL: https://www.sciencedirect.com/science/article/pii/B978012801538400001X.