Akademia Górniczo Hutnicza

im. Stanisława Staszica

Wydział Informatyki, Elektroniki i Telekomunikacji

Projekt z przedmiotu

Inteligencja Obliczeniowa

Zastosowania algorytmów ewolucyjnych w problemach optymalizacji transportu - implementacja wybranego algorytmu oraz eksperymentalne badanie efektywności

Autorzy:
Marek Sadowski
Marcin Jaroch

Prowadzący:
dr inż. Rafał Dreżewski

Streszczenie

Celem projektu było zaimplementowanie wybranego algorytmu ewolucyjnego z zakresu optymalizacji transportu. Dokument ten przedstawia zasadę działania wybranego algorytmu i proces implementacji z uwzględnieniem wykorzystanych narzędzi oraz przeprowadzone eksperymenty wraz z wynikami.

Spis treści

Rozdział 1

Definicja problemu i wybrany algorytm

W tym rozdziale znajduje się opis problemu i algorytmu go rozwiązującego zgodny z artykułem *GVR: a New Genetic Representation for the Vehicle Routing Problem* autorstwa *Francisco B. Pereira, Jorge Tavares, Penousal Machado, Ernesto Costa.*

1.1 Problem

Rozważany problem można formalnie zdefiniować w następujący sposób: Zakładamy istnienie jednego, centralnego ośrodka 0, który wykorzystuje k niezależnych samochodów dostawczych. Każdy z pojazdów posiada identyczną ładowność C. Należy obsłużyć n klientów o d_i zapotrzebowaniach, $i = 1, 2, ..., n$. Samochody muszą wykonać dostawy do klientów z minimalną całkowitą długością trasy, gdzie długość c_{ij} to odległość pomiędzy klientami i i j gdzie $i \in [1, n]$. Każdy klient musi być odwiedzony dokładnie jeden raz. Rozwiązaniem problemu jest zbiór k tras $R_1, ..., R_k$ gdzie dla każdej trasy

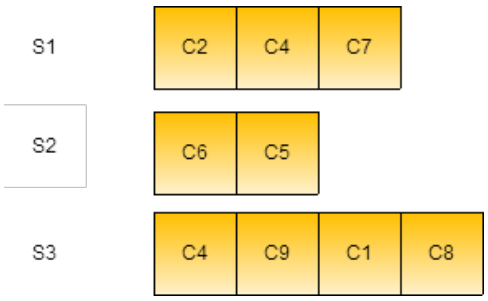
R_q suma zapotrzebowań nie przekracza C.

1.2 Algorytm

Podobnie jak w innych algorytmach ewolucyjnych, możemy w tym przypadku zdefiniować chromosom (będący reprezentacją pojedynczego rozwiązania) oraz operatory genetyczne.

1.2.1 Reprezentacja rozwiązania

Chromosom składa się z kolekcji ścieżek. Każda ścieżka odpowiada jednemu samochodowi. Przedstawia to rysunek 1.1.



Rysunek 1.1: Reprezentacja chromosomu. Sx i Cx oznaczają odpowiednio samochód i klienta.

1.2.2 Operatory

Krzyżowanie

Sposób krzyżowania osobników I_1 i I_2 można wyrazić jako listę kroków:

1. Wybierz losową pod-ścieżkę SR z I_2
2. Wybierz klienta c, nie znajdującego się w SR, który ma najmniejszy dystans do pierwszego klienta w SR

3. Wstaw SR do I_1 zaraz po c

4. Usuń zduplikowanych klientów w I_1 otrzymując potomka D

Mutacja

Autorzy artykułu, na którym się wzorowano przy realizacji projektu, przewidzieli cztery rodzaje mutacji. Są to operatory działające na pojedynczym osobniku.

Swap losowo wybiera dwóch klientów i zamienia ich miejscami.

Inversion wybiera pod-ścieżkę i odwraca jej kolejność.

Insertion wybiera klienta i wstawia go do losowej ścieżki (może tworzyć nową ścieżkę).

Displacement wybiera pod-ścieżkę i wstawia ją w inne, losowo wybrane, miejsce (może tworzyć nową ścieżkę).

1.2.3 Selekcja

Posłużono się selekcją turniejową o rozmiarze turnieju równym 5.

Rozdział 2

Implementacja

Ta część traktuje zarówno o procesie wykorzystanych narzędziach jak i szczegółach implementacyjnych.

2.1 Narzędzia

Językiem programowania, jaki został wykorzystany jest Python 2.7.8. Oprócz biblioteki standardowej tego języka, użyto także frameworka DEAP, wspomagającego implementację algorytmów genetycznych. Główne cechy DEAP:

- Predefiniowane chromosomy
- Typowe rodzaje krzyżówek i mutacji
- Operatory selekcji
- Programowanie genetyczne
- Integracja z NumPy i NetworX

2.2 Implementacja

Problem został wyrażony za pomocą klas języka Python. Wyróżniono klasy reprezentujące klienta, ścieżkę oraz osobnika. Graf odległości między klientami reprezentowany jest przez macierz symetryczną. Wszelkie dane wejściowe (macierz odległości, ładowność samochodu, maksymalne zapotrzebowanie klienta) mogą zostać zainicjowane losowo, lub wprowadzone ręcznie w pliku konfiguracyjnym. Pomimo użycia biblioteki DEAP, wszystkie potrzebne operatory (z wyjątkiem selekcji), należało zaimplementować od podstaw. Zostało to wykonane zgodnie z opisem w rozdziale 1. Ponadto, w trakcie prac implementacyjnych, zaszła potrzeba modyfikacji lub dodania nowych elementów do bazowego algorytmu, z uwagi na niewystarczająco szczegółowy opis. Kolejne sekcje skupiają się na wspomnianych elementach.

2.2.1 Normalizacja osobnika

Operatory mutacji i krzyżowania mogą wprowadzić osobnika w nieprawidłowy stan. Możliwe jest powielenie klienta lub przekroczenie limitu ładowności na pojedynczej ścieżce. Aby zredukować ten problem zaimplementowano procedurę normalizacji, zapisaną poniżej.
Dla każdej ścieżki w osobniku wykonuj:

1. Jeśli suma zapotrzebowań na ścieżce przekracza limit ładowności, wyznacz minimalną liczbę pod-ścieżek spełniających warunki.

2. Zastąp ścieżkę wyznaczonymi pod-ścieżkami.

Procedura uruchamiana jest po każdej krzyżówce bądź mutacji i zapewnia spójność osobników populacji.

2.2.2 Generacja wstępnej populacji

Analogiczna procedura opisana w artykule bazowym nie gwarantowała znormalizowanych osobników. Zmodyfikowane tworzenie wstępnej populacji przebiega następująco:

while *nieodwiedzeni klienci* **do**

1. Wybierz losowego klienta c
2. Dodaj go do aktualnej ścieżki R, jeżeli po dodaniu suma zapotrzebowań na R nie przekroczy limitu C
3. W przeciwnym wypadku umieść c w nowej ścieżce

end

Rozdział 3

Eksperymenty

Badano zachowanie algorytmu