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 [1].

1.1 Problem

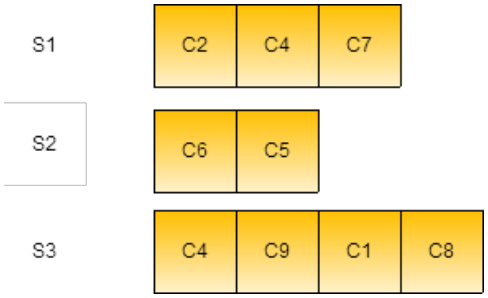
Rozważany problem można formalnie zdefiniować w następujący sposób: Zakładamy istnienie jednego, centralnego ośrodka 0, który wykorzystuje k niezależnych samochodów dostawczych. Każdy z pojazdów posiada identyczną ładowność C. Należy obsłużyć n klientów o d_i zapotrzebowaniach, $i = 1, 2, ..., n$. Samochody 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 [1] 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 NetworkX

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 [1] 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

W tej części przedstawiono wyniki przeprowadzonych eksperymentów na zaimplementowanym algorytmie. Badania były wzorowane na tych, które przedstawiono w [1]. Tabela 3.1 przedstawia parametry użyte podczas eksperymentowania. Ich modyfikacje są zaznaczone w tabelach 3.2 i 3.3.

Tabele pozwalają wnioskować, że ogólną tendencją jest poprawa wyników wraz ze wzrostem liczby generacji. Krzyżowanie poprawiało uzyskany wynik dla większych populacji i liczb generacji, przy obniżonym prawdopodobieństwie *insertion*. Okazało się także korzystne dla podwyższonych prawdopodobieństw *displacement i inversion* dla mniejszych populacji. W pozostałych przypadkach krzyżowanie ma nieznaczny bądź negatywny wpływ na wynik. Najkrótsze ścieżki otrzymywano przy zwiększonych prawdopodobieństwach wszystkich operatorów mutacji. W naszej implementacji krzyżowanie wprowadza różnorodność genetyczną gdy algorytm ma niewielką populację początkową i ograniczoną liczbę iteracji, co w niektórych przypadkach prowadzi do poprawy końcowego wyniku. Wyniki te są niezgodne z tymi uzyskanymi w [1], gdzie krzyżowanie w każdym przypadku prowadziło do poprawy rezultatu. Jest to spowodowane różnicami implementacyjnymi i założeniami

(normalizacja, generacja populacji początkowej), które zostały przyjęte ze względu na powierzchowny opis oryginalnego algorytmu.

Podczas implementacji algorytmu, zauważono, że tendencję do znajdowania minimum lokalnego gdy używany jest tylko jeden operator mutacji (*swap*). Dodanie pozostałych operatorów całkowicie rozwiązało ten problem. Fakt odporności na przedwczesną zbieżność do lokalnego optimum zauważają także autorzy [1].

Mimo różnic w otrzymanych wynikach, uzyskano poprawny algorytm genetyczny, optymalizujący rozwiązanie problemu znajdowania tras pojazdów.

Domyślne parametry							
crossover	swap	inversion	insertion	displacement	max_demand	max_capacity	depot_cnt
0.75	0.05	0.1	0.05	0.1	27	97	37

Tablica 3.1: Domyślne ustawienia algorytmu, na których przeprowadzono testy

		Insertion = 0.05			
Nr. of generations	Population size	Swap = 0		Swap = 0.1	
		Cross = 0	Cross = 0.75	Cross = 0	Cross = 0.75
200	200	964.0	954.0	722.0	753.0
400	200	745.0	870.0	614.0	681.0
200	400	720.0	729.0	678.0	683.0
400	400	719.0	656.0	598.0	551.0
2000	600	649.0	706.0	573.0	503.0
		Insertion = 0.15			
		Swap = 0		Swap = 0.1	
		Cross = 0	Cross = 0.75	Cross = 0	Cross = 0.75
200	200	699.0	739.0	636.0	726.0
400	200	607.0	602.0	533.0	624.0
200	400	588.0	745.0	623.0	775.0
400	400	511.0	695.0	620.0	632.0
2000	600	593.0	660.0	516.0	541.0

Tablica 3.2: Zależność między operatorami krzyżowania, insertion i swap

		Inversion = 0.1			
Nr. of generations	Population size	Disp = 0.1		Disp = 0.2	
		Cross = 0	Cross = 0.75	Cross = 0	Cross = 0.75
200	200	901.0	789.0	751.0	810.0
400	200	767.0	742.0	662.0	674.0
200	400	747.0	804.0	735.0	819.0
400	400	674.0	656.0	625.0	743.0
2000	600	479.0	490.0	562.0	600.0
		Inversion = 0.15			
		Disp = 0.1		Disp = 0.2	
		Cross = 0	Cross = 0.75	Cross = 0	Cross = 0.75
200	200	782.0	804.0	767.0	686.0
400	200	694.0	747.0	682.0	646.0
200	400	738.0	768.0	670.0	768.0
400	400	651.0	743.0	615.0	637.0
2000	600	516.0	573.0	497.0	729.0

Tablica 3.3: Zależność między operatorami krzyżowania, inversion i displacement

Bibliografia

[1] F. B. Pereira, J. Tavares, P. Machado, and E. Costa, "Gvr: a new genetic representation for the vehicle routing problem," in *Problem, Proceedings of the 13th Irish Conference on Artificial Intelligence and Cognitive Science*, pp. 95–102, Springer-Verlag, 2002.