Podstawy Sztucznej Inteligencji - Projekt
Problem Komiwojażera

A. Białobrzeski, D. Bułak, M. Waszak

20 stycznia 2017

1 Treść projektu

Należy wybrać N miast polskich i znaleźć najkrótszy cykl łączący je wszystkie.
Odległości między miastami można znaleźć np. pod adresem http://www.odleglosci.pl/tabele-odleglosci.php
Algorytm: genetyczny lub ewolucyjny (μ+λ) i (μ,λ).
Należy ustalić w przybliżeniu najlepsze parametry działania obu algorytmów.
Interfejs: pokazuje postęp procesu optymalizacji, podaje ostateczną kolejność miast.
Należy sprawdzić program dla N=10, 20, 30.

2 Relizacja

Program wyświetla okno graficzne, w którym wybiera się, który algorytm ma być stosowany ((μ+λ) albo (μ,λ)), wprowadza się parametry: μ, λ i ilość miast (n). Po kliknięciu przycisku ıStart następuje wybranie n miast z pliku, który zawiera nazwy miast oraz ich pozycje (*latitude* i *longitude*), w tym momencie następuje uruchomienie algorytmu.
Algorytm uruchamiany jest w innym wątku niż GUI tak, aby swoim działaniem nie blokował interakcji użytkownika z oknem.
Działanie algorytmu opisano w następnym rozdziale.
Algorytm zwraca obiekt typu *Tour*, który zawiera uporządkowaną od początku do końca listę miast, które komiwojażer powinien odwiedzić.
Tak przygotowana lista jest wyświetlana w oknie za pomocą grafu.

3 Algorytm

Implementacja algorytmu znajduje się przede wszystkim w klasie TourCalculator, która to jest za niego odpowiedzialna. Po uruchomieniu metody *Run*() tej klasy wykonane zostają następujące kroki:

1. Zainicjalizowana przekazanymi do metody miastami zostaje nowa μ-elementowa populacja *initialPopulation*.

2. Dla liczby zdefiniowanych kroków (przyjęto 100) zostają wykonane operacje reprodukcji:

   (a) Losowana jest z populacji początkowej λ-elementowapopulacja tymczasowa *tempPopulation*

   (b) Populacja tymczasowa jest **krzyżowana**, a wyniki tego krzyżowania zapisywane są w populacji zreprodukowanej *repPopulation*

   (c) Populacja zreprodukowana *repPopulation* poddawana jest **mutacji**.

   (d) Nowa populacja początkowa (μ-elementowa) dla kolejnego kroku algorytmu wybierana jest z populacji przeznaczonej do tego wyboru poprzez losowanie z niej *mi* elementów. Populacja przeznaczona do tego losowania tworzona jest w zależności od wybranego algorytmu:

      i. Dla $(\mu + \lambda)$ są to połączone populacje początkowa i zreprodukowana.

      ii. Dla (μ, λ) jest to tylko populacja zreprodukowana.

3. Po zakończeniu wszystkich zadanych iteracji zwracany jest osobnik (*Tour*) o najlepszej funkcji przystosowania, czyli najkrótszej odległości łączącej wszystkie miasta.

3.1 Krzyżowania

Krzyżowanie zaimplementowane w metodzie *Crossover* klasy *Population* składa się z dwóch kroków:

1. Wybranie rodziców - metoda wybora to turniej. Polega on na stworzeniu niewielkiej populacji (przyjęto 5-elementową), do której osobniki losowane są z populacji, którą krzyżujemy. Następnie z członków turnieju wybierany jest ten o najlepszej funkcji przystosowania. Tworzymy dwa turnieje i po ich rozegraniu otrzymujemy dwoje rodziców.

2. Krzyżowanie rodziców ze sobą polega na stworzeniu nowego osobnika, którego część miast jest przeniesiona od pierwszego rodzica, a część od drugiego. Należy tutaj jednak pamiętać, że miasta nie mogą się powtarzać, więc przy przenoszeniu miast z drugiego osobnika musimy sprawdzać, czy takie miasto już w nim nie istnieje.

3.2 Mutacja

Mutacja zaimplementowana w metodzie *Mutate* klasy *Population* polega na zmianie kolejności niektórych miast w osobnikach w sposób losowy. Czy mutacja zachodzi dla danego miasta w osobniku czy nie wynika z wylosowanej liczby, która jeśli jest większa lub równa od czynnika *MutationRate* to mutacja zachodzi, w przeciwnym razie nie.
Jeśli okazuje się, że zachodzi wtedy losujemy drugie miasto w osobniku i zamieniamy miejscami z analizowanym.
Czynnik *MutationRate* określa prawdopodobieństwo wystąpienia mutacji i został przyjęty jako 0.015