Rozwiązanie zadania komiwojażera przy pomocy algorytmu genetycznego. - wersja sekwencyjna działająca na pojedynczym procesorze.

Onaszkiewicz Przemysław, Gadawski Łukasz

30 listopada 2015

1 Cel zadania

Celem pierwszego etapu projektu jest implementacja wersji sekwencyjnej algorytmu genetycznego działającego na pojedynczym procesorze do rozwiązywania problemu komiwojażera.

2 Problem komiwojażera

Problem zdefiniowany jest następująco: mając listę miast oraz odległości między nimi, znajdź najkrótszą, dozwoloną drogę (początkiem oraz końcem drogi jest ten sam wierzchołek), zawierającą każde miasto dokładnie raz. Inaczej problem polega na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym. Problem należy do zbioru problemów NP-trudnych.

3 Algorytm genetyczny

W 1975 roku przez Johna Hollanda został wynaleziony algorytm genetyczny, którego zadaniem było działanie analogiczne do procesu ewolucji na ziemi. Pseudokod prostego algorytmu genetycznego (ang. *Simple Genetic Algorithm*) został przedstawiony poniżej:

```
procedure "Simple Genetic Algorithm"
begin
        t := 0
        initilize P^{t}
        evaluate P^{t}
        while (not stop_condition)
        begin
                O^{t} = reproduce P^{t}
                crossover O^{t}
                mutate O^{t}
                evaluate O^{t}
                P^{t + 1} := O^{t}
                t := t + 1
```

```
        end
end
```

sum−

4 Implementacja

4.1 Tworzenie mapy miast

W naszej implementacji mapa została odwzorowana jako obiekt posiadający wielkość, zbiór miast oraz strukturę danych *mapa* gdzie kluczem jest para miasto-miasto oraz odległość między taką na mapie.

4.1.1 Wczytanie z pliku

4.1.2 Losowa generacja mapy

4.2 Konfiguracja programu

Konfiguracja programu jest możliwa przy pomocy pliku *app.properties*. Umożliwia on zdefiniowanie następujących parametrów:

- **population_size** - wielkość populacji w każdej iteracji algorytmu,
- **propability_of_crossover** - prawdopodobieństwo krzyżowania,
- **propability_of_mutation** - prawdopodobieństwo mutacji,
- **generation_number** - liczba pokoleń, czyli iteracji algorytmu, w przypadku naszej implementacji jest to warunek stopu.

Przykładowa zawartość pliku konfiguracyjnego została zamieszczona poniżej.

```
population_size=5
propability_of_crossover=0.20
propability_of_mutation=0.01
generation_number=3
```

5 Wnioski

6 Podsumowanie