Algorytmy genetyczne

Wstęp

Algorytmy genetyczne to jedne z popularniejszych obecnie metod optymalizacji. Ich główną zaletą jest prostota implementacji oraz duża łatwość w zrównoleglenia. Te atuty powodują, że algorytmy genetyczne (GA) znalazły zastosowanie niemal we wszytkach problemach optymalizacyjnych. Na uwagę zasługuje tutaj fakt, iż cechują się one własnością odnajdywania ekstremów globalnych , czyli są to tak zwane globalne metody optymalizacji, które w teorii pozwalają na znalezienie ekstremum globalnego. Własności tej pozbawione sa np. metody gradientowe, które mają tendencje do utykania w ekstremach lokalnych.

Główna wadą GA jest konieczność wielokrotnej ewaluacji funkcji dopasowania (funkcji kosztu), co w pewnych zastosowaniach znacząco utrudnia ich wykorzystanie –np. w problemach optymalizacji urządzeń gdzie czas wykonania pojedynczej oceny - symulacji komputerowych dla zadanej konfiguracji urządzeń, sięga wielu godzin/dni.

Na jakość algorytmów genetycznych mają wpływ 3 operacje/operatory. Są nimi:

- selekcja – czuli dobór osobników stanowiących rodziców dla kolejnego pokolenia
- krzyżowanie – łączenie osobników w pary w celu wydobycia z nich najcenniejszych informacji i stworzenia potomstwa
- mutacja – losowe zmiany wprowadzane do poszczególnych osobników, których celem jest zapewnienie wspomnianej wcześniej właściwości globalności uzyskanego rozwiązania

Mikro kod GA przedstawia poniższy schemat[1]:

```
1:  popsize ← desired population size
2:  P ← {}
3:  for popsize times do
4:      P ← P ∪ {new random individual}
5:  Best ← □
6:  repeat
7:      for each individual P_i ∈ P do
8:          AssessFitness(P_i)
9:          if Best = □ or Fitness(P_i) > Fitness(Best) then
10:             Best ← P_i
11:     Q ← {}                                          ▷ Here's
12:     for popsize/2 times do
13:         Parent P_a ← SelectWithReplacement(P)
14:         Parent P_b ← SelectWithReplacement(P)
15:         Children C_a, C_b ← Crossover(Copy(P_a), Copy(P_b))
16:         Q ← Q ∪ {Mutate(C_a), Mutate(C_b)}
17:     P ← Q
18: until Best is the ideal solution or we have run out of time
19: return Best
```

[1] Ten jak i wszystkie pozostałe schematy pochodzą z książki Essentials of Metaheuristics Sean Luke

W powyższym schemacie linijki 2 do 4 odpowiadają za inicjalizację algorytmu, czyli wybór osobników populacji początkowej, a następnie począwszy od linii 6 rozpoczyna się główna pętla programu powtarzana aż do osiągnięcia określonego wyniku, lub też określonej liczby iteracji. W kolejnym kroku (linie od 7 do 10) odpowiedzialne są za wyznaczenie najlepszego z dotychczasowych rozwiązań.

Właściwy algorytm rozpoczyna się od linii 12 do 16, gdzie w pętli poszukiwanych jest para dwóch osobników stanowiących rodziców dla kolejnych pokoleń (linia 13 i 14) – proces ten stanowi implementację procesu selekcji, następnie rodzice są krzyżowani tak aby utworzyć dzieci (linia 15), które następnie poddawane sa operacji mutacji i dodawane do listy nowych osobników.

Typowym operatorem selekcji jest tak zwana selekcja turniejowa (ang. tournament Selection) realizowane poprzez:

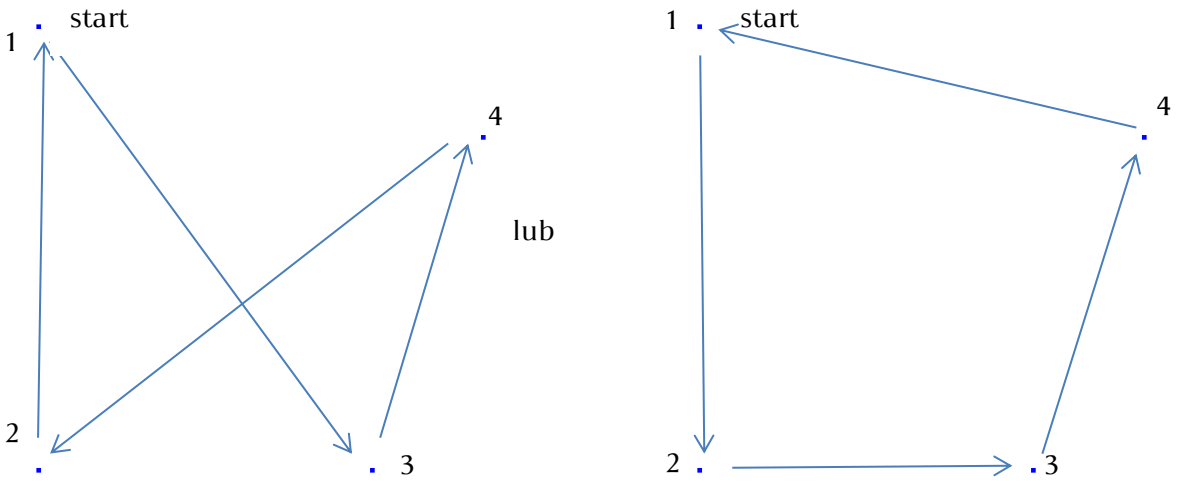
```
1: P ← population
2: t ← tournament size, t ≥ 1

3: Best ← individual picked at random from P with replacement
4: for i from 2 to t do
5:     Next ← individual picked at random from P with replacement
6:     if Fitness(Next) > Fitness(Best) then
7:         Best ← Next
8: return Best
```

Selekcja turniejowa polega na t krotnym wylosowaniu liczby odpowiadającej indeksowi w obecnej populacji i zapamiętaniu najlepszego spośród t rozwiązań.

Kluczową rolę w zastosowaniach GA stanowi dobór operatorów krzyżowania i mutacji w zależności od postawionego problemu. W niniejszym laboratorium GA zostaną wykorzystane do realizacji problemu komiwojażera. Problem komiwojażera to problem w którym komiwojażer ma za zadanie odwiedzić n miast, przy czym poszczególne miasta nie powinny się powtarzać, a jednocześnie powinien powyższe wykonać tak aby trasa jego odwiedzin była jak najkrótsza.

Przyjmując dla uproszczenia że poruszamy się po liniach prostych, wówczas powyższy problem dla 4 miast można przedstawić graficznie jako:



Tak postawiony problem jest więc problemem optymalnego kolejkowania, tzn. zbudowania odpowiedniego harmonogramu odwiedzin. Dla przedstawionych powyżej przykładów zaproponowane sekwencje to [1 3 4 2 1] oraz [1 2 3 4 1]

Do realizacji tak postawionego zadania poprzez wykorzystanie algorytmów genetycznych najlepszym rozwiązaniem jest zakodowanie osobników w postaci opisanych powyżej sekwencji liczb całkowitych. W takim przypadku operator mutacji można najprościej zaimplementować jako:

1: $\vec{v} \leftarrow$ integer vector $\langle v_1, v_2, \ldots v_l \rangle$ to be mutated
2: $p \leftarrow$ probability of randomizing an integer

3: **for** i from 1 to l **do**
4: **if** $p \geq$ random number chosen uniformly from 0.0 to 1.0 inclusive **then**
5: $v_i \leftarrow$ new random legal integer
6: **return** $\vec{v}$

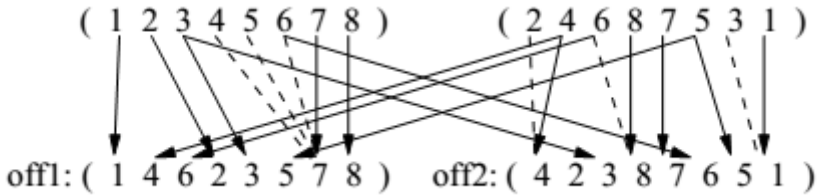
A operator krzyżowania jako tak zwany *Position based crossover* czyli

1) Wylosowanie punktu krzyżowania
2) Odczytanie wartości z osobników
3) Wymiana informacji pomiędzy osobnikami na poszczególnej pozycji
4) Konwersja zduplikowanych wartości.

Powyższe można zapisać jako:

```
ids = pozycje punktów krzyżowań
for i = ids;
    v1 = Pa(i);
    v2 = Pb(i);
    Pa(Pa==v2) = v1;
    Pb(Pb==v1) = v2;
    Pa(i) = v2;
    Pb(i) = v1;
end;
```

A przykładowo dla osobników [1 2 3 4 5 6 7 8] oraz [2 4 6 8 7 5 3 1] można również przedstawić jak na rys[2]:



Zadanie

1) Zaimplementuj funkcję kosztu, która dla zadanego osobnika wyznacza poziom jego dopasowania licząc sumaryczną odległość do przebycia. Uwaga jako dane przyjmij na wejściu macierz odległości

```
function costVal = costValFunction(param,state)
```

gdzie :

- Param – parametry funkcji kosztu, tutaj macierz odległości

[2] Źródło eneti
Algorithms for the Travelling Salesman Problem: A Review of Representations and Operator. P.Larra~naga,C.M.H.Kuijpers,R.H.MurgaI.InzaandS.Dizdarevi, Department of Computer Science and Artificial Intelligence, University of theBasque, Spain

- State – zmienna reprezentująca osobnika

2) Operator mutacji opisany wyżej w części teoretycznej w postaci funkcji
```
function state = mutate(param, state)
```
gdzie:
   - Param – zbiór parametrów operatora mutacji, w tym prawodpodobieństwo mutacji
   - State – osobnik poddany mutacji
3) Operator krzyżowania
```
function [state1,state2] = crossover(param,state1,state2)
```
Opisany wyżej operator krzyżowania który jako parametry wywołnia przyjumje:

   - param – zbiór parametrów konfiguracyjnych w tym prawdopodobieństwo krzyżowania, liczba punktów krzyżowania

   - state1 – pierwszy z rodziców

   - state2 – drugi z rodziców

4) Zaimplementuj opisany powyżej algorytm genetyczny w postaci funkcji
```
function [bestState,bestFitness] = GA(param,init)
```
gdzie:
   - param – zbiór parametrów konfiguracyjnych\
   - init – populacja początkowa
   - jako wynik zwracane sa najlepszy osobnik oraz poziom jego dopasowania
5) Zbuduj główny program który optymalizuje drogą w problemie komiwojażera z wykorzystaniem zaimplementowanego wcześniej algorytmu genetycznego
Powyższe wykonaj dla problemu 4 miast, gdzie macierz odległości opisana jest jako:
D = [0.0 3.0 5.0 4.0
  3.0 0.0 3.162277 5.0
  5.0 3.162277 0.0 4.1231055
  4.0 5.0 4.1231055 0.0]
6) Dokonaj modyfikacji parametrów konfiguracyjnych i sprawdź jak one wpływają na jakość rozwiązania. Optymalna trasa (bez powrotu do miejsca ztartu) powinna liczyć 10.1623
7) Spróbuj rozwiązać problem komiwojażera dla problemu 15 miast, gdzie macierz odległości dostępna jest na stronie prowadzącego.