Algorytm genetyczny dla problemu komiwojażera

Piotr Popis

Czerwiec 2020

1 Wprowadzenie

1.1 Problem komiwojażera(Traceling Salesman Problem)

Jest jednym z najszerzej przebadanych problemów optymalizacji kombinatorycznej. Polega na zminimalizowaniu dystansu trasy Salesmana, gdzie trasa musi przechodzić przez zadane n (załóżmy, że) *miast* z jego listy, które powinien przejść dokładanie raz oraz znane są odległości pomiędzy miastami. Inaczej mówiąc szukamy minimalnego cyklu Hamiltona w pełnym grafie ważonym.
Problem możemy przedstawiać w różnych wariantach na przykład:

1. symetryczny sTSP
2. asymetryczny aTSP
3. wielokrotny mTSP
4. zgeneralizowany gTSP

W symetrycznym przypadku odległość z miasta A do B jest taka sama jak odległość z B do A. W asymetrycznym odległości te mogą być różne ($c_{ij} \neq c_{ji}$). W 3 przypadku wielokrotnym miasto może być odwiedzone więcej niż raz, a w przypadku zgeneralizowanym(uogólnionym) nie wszystkie miasta muszą być odwiedzone. Tego typu problemy powstają w różnych zastosowaniach ciekawymi przykładami są między innymi vehicle routing problem, printed circuit board punching sequence problems, wing nozzle design problem in air craft design.

1.2 Algorytm Genetyczny(Genetic Algorithm)

1.2.1 Opis

Jest heurystyką, a mianowicie członkiem grupy algorytmów ewolucyjnych. Jest jedną z metod inteligencji obliczeniowej. Metodologia jest inspirowana efektywnością naturalnej selekcji w ewolucji biologicznej. W przeciwieństwie do innych metaheurystyk takich jak tabu serach czy simulated annealing nie generuje jednego rozwiązania, a grupę rozwiązań (generację). Bieżąca generacja nazywana jest populacją. Każdego członka bieżącej grupy nazywamy nieraz chromosomem. Do stworzenia nowej generacji używamy genotypów z poprzedniej generacji. Utworzenie nowego chromosomu, w zasadzie grupy chromosomów - generacji następuje zgodnie z różnymi operacjami.
Trzy najczęściej używane to:

1. selekcja(selection),
2. krzyżowanie(crossover),
3. mutacja(mutation).

Operacja selekcji pozwala wybrać odpowiednich osobników do nowych generacji. Wybiera zazwyczaj tych najlepszych, ale warto zostawić dla tych gorszych pewne prawdopodobieństwo wybrania w celu ucieczki z lokalnych optimum. Istnieje też możliwość rekombinacji czyli bezpośrednim przekazywaniu najlepszym osobników z populacji do nowej generacji w celu zachowania rozwiązań o bardzo wysokiej jakości. Krzyżowanie pozwala na wymieszanie genów wybranych w selekcji osobników. Mutacja natomiast polega na zazwyczaj losowym zmienieniu niektórych genów. Każda z powyższych operacji ma wiele efektywnych implementacji, które są zależne od problemu. Znaczy to tyle, że die da się jednoznacznie określić najlepszej implementacji, tylko jedna jest bardziej dokładna obliczeniowo, inna znacznie szybsza. Kolejnym problemem naszego algorytmu są lokalne optima, z których w inteligentny sposób musimy uciekać.

1.3 Kilka wybranych implementacji powyższych operacji(pseudokod)

1.3.1 Selekcja

Fitness-Proportionale Selection (Roulette Selection) Wybieramy osobników proporcjonalnie do ich jakości.

```
do once per generation
  global p = [ind_1,...,ind_ps]
  global f = [fitness(pi) for pi in p]
  if f is all 0.0s:
    convert f to all 1.0s
  for i from 2 to l:
    fi = fi+ f_{i-1}
perform each time
  n = random(0,fl)
  for i from 2 to l:
    if  f_{i-1}<n<=fi:
      return pi
      return p1
```

Tournament Selection Wybieramy losowo t osobników i z nich najlepszego + replace.

```
P- population
t- tounament size >0
best = random.choice(P).replace()
for i from 2 to t:
  next = random.choice(P).replace()
  if fitness(next)>fitness(Best)
    best=next
return best
```

1.3.2 Krzyżowanie

One-point crossover -Zamiana suffixów dwóch osobników.

```
v,w - vectors ( paths)
c = randomint(1,l)
if c!=1:
  for i in (c,l):
    swap(vi,wi)
return v,w
```

Two-point crossover Wymiana genów na zadanym przedziale c,d.

```
v,w - vectors ( paths)
c,d = randomint(1,l),randomint(1,l)
if c>d:
  c,d=d,c
if c!=d:
  for i in (c,d-1):
    swap(vi,wi)
return v,w
```

Uniform crossover Zamiana każdego genu z prawdopodobieństwem p.

```
p - probab of swapping an index
v,w - vectors ( paths)
c,d = randomint(1,l),randomint(1,l)
for i in (1,l):
  if p>random.uniform(0,1):
    swap(vi,wi)
return v,w
```

1.3.3 Mutacja

Swap Zamiana genów na pozycjach (losowych i,j).

```
i,j - random cities range(0,n)
return swap(path,i,j)
```