Algorytm genetyczny dla problemu komiwojażera

Piotr Popis

Czerwiec 2020

1 Wprowadzenie

1.1 Problem komiwojażera(Traveling 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. Należy do problemów NP- trudnych.

1.2 Algorytm Genetyczny(Genetic Algorithm)

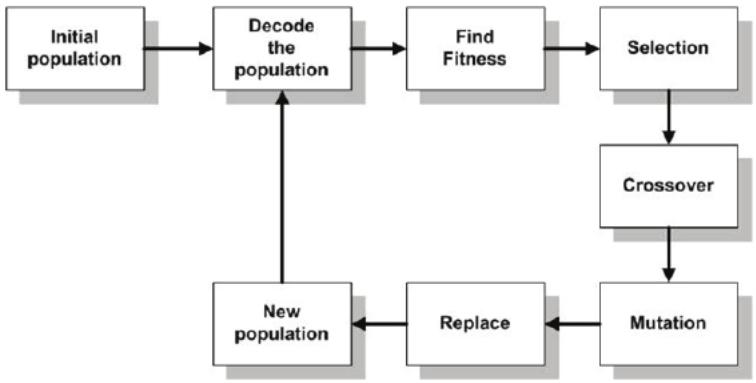
1.2.1 Opis

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.

1.2.2 Schemat



1.3 Kilka wybranych implementacji powyższych operacji(pseudokod)

1.3.1 Selekcja

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)
```

Teraz znając przykładowe implementacje operacji w algorytmie genetycznym głównie opierające się na permutacjach, możemy zacząć rozważania. Możemy postawić, że naszym celem jest znalezienie jak najbardziej optymalnego rozwiązania w jak najkrótszym czasie. Problemem stają się lokalne optima, których musimy pokonać odpowiednio dobierając operacje mutacji, krzyżowania oraz selekcji. Rozważmy kilka solucji problemu komiwojażera wykorzystując optymalny algorytm genetyczny.

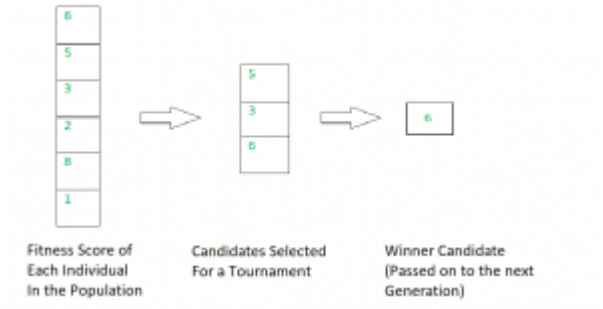
2 Różne strategie selekcji

Jedną z operacji w algorytmie genetycznym jest selekcja. Służy do wybrania osobników, które następnie chcemy skrzyżować. Przy doborze

rodziców musimy uważać, żeby nasze wybory nie doprowadziły do zamknięcia się na jeden genotyp

2.1 Tournament Selection

Bardzo popularną metodą jest selekcja turniejowa. Z większej populacji wybieramy x przedstawicieli losowo, wielkość x nazywamy rozmiarem turnieju. Binary Tournament zatem, to selekcja, w której do konkurencji wybieramy tylko 2 chromosomy. Następnie z wyodrębnionego zbioru wybieramy najlepszego. Takie podejście pozwala zachować różnorodność.

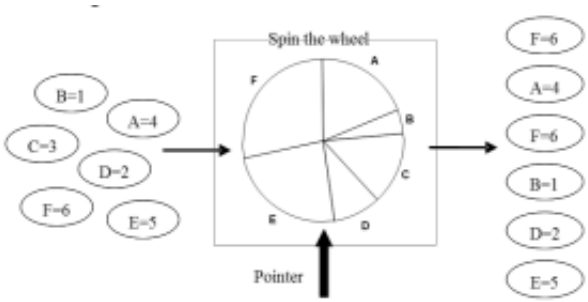



```
P - population
t- tounament size >0
best = random.choice(P)
for i from 2 to t:
  next = random.choice(P)
  if fitness(next)>fitness(Best)
    best=next
return best
```

2.2 Proportional Roulette Wheel Selection

Osobniki są wybierane z prawdopodobieństwem bezpośrednio proporcjonalnym do ich jakości (f -fitness value). Odpowiada to nieco kole ruletki.Koło możemy podzielić na segmenty odpowiadające proporcjom rodziców. Prawdopodobieństwo takie możemy wyznaczyć korzystając z wzoru $p_i = \frac{f_i}{\sum_{j=1}^{n} f_i}$. Główną zaletą jest to, że żaden z elementów populacji nie zostaje przekreślony. Ma też niestety swoje wady to znaczy, że jeśli populacja początkowa zawiera załóżmy, 2-3 silne genotypy, ale nie idealne, a pozostali członkowie populacji są bardzo słabi - zamknie się na tych dwóch najlepszych (prawdopodobnie). Z drugiej strony słabe rozwiązania są szybko eliminowane. Jednak jeśli rozważamy problem minimalizacji to jest nieco uciążliwe rozwiązanie, bo musimy zaimplementować funkcje

konwertującą funkcje minimalizującą do maksymalizującej. Wprowadza zatem lekkie zamieszanie.




```
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
```

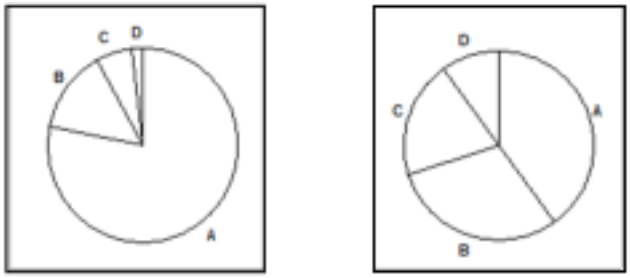
2.3 Rank-based Roulette Wheel Selection

Prawdopodobieństwo wybrania chromosomu jest tutaj zależna od fitness value oraz jest relatywna do całej populacji. Najpierw osobnicy zostają posortowani zgodnie z ich jakością, a następnie wyliczane jest prawdopodobieństwo na podstawie ich rankingu ,a nie bezpośrednim fitness. Potrzebujemy funkcji mapującej indeksy na posortowanej liście do listy jej prawdopodobieństw selekcji. Funkcja ta może, ale nie musi być liniowa. Bias może być regulowany przy użyciu nacisku wyboru SP dla liniowej np 2>SP>1. Pozycja zatem może być skalowana zgodnie z formułą:

$$Rank(Pos) = 2 - SP + (2.(SP - 1).\frac{Pos - 1}{n - 1})$$

Można uniknąć skalowania, ale to może stać się bardziej kosztowne obliczeniowo z powodu sortowania. Takie podejście pozwala zablokować sytuację powodującą utknięcie w najlepszych chromosomach początkowych.

Różnica pomiędzy kołami ruletki dla odpowiednio proporcjonalnej i rank-based selekcji.



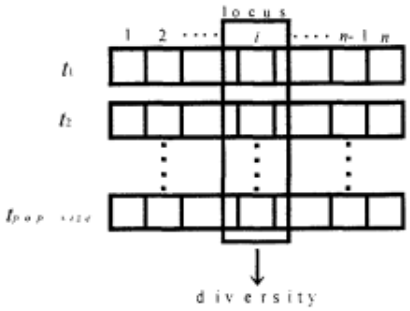
Porównanie selekcji:

Instances	Known optimal solution	Tournament	Proportional	Rank-based
10-city	-	2.8567	2.8567	2.8567
20-city	-	4.0772	4.0772	4.0772
30-city	-	4.8352	4.9075	4.6683
40-city	-	6.1992	6.5127	5.7311
burma14	30.8785	30.8785	30.8785	30.8785
bay29	9074	9077	9079	9074
dantzig42	679	725	760	679
eil51	425	470	513	430

3 Algorytm genetyczny bazujący na entropii

3.1 Opis

Aby poradzić sobie z problemem lokalnego optimum Yasuhiro TSUJIMURA and Mitsuo GEN zaprezentowali rozwiązanie bazujące na entropii. Wykorzystali operację cycle crossover, swap mutation oraz selekcję - roulette wheel selection. Jak wiemy zwiększenie się ilości podobnych chromosomów w populacji może doprowadzić do utknięcia w lokalnym optimum. W EBGA mierzymy różnorodność chromosomów w każdej nowej generacji i ulepszamy populacje o małej różnorodności. Jak wyznaczyć zatem różnorodność?



w tak ustawionych chromosomach przechodzimy po każdej kolumnie i sprawdzamy w niej różnorodność. Locus diversity H_i i-tego locus możemy wyznaczyć ze wzoru:

$$H_i = \sum_{c \in C} pr_{ic} ln(pr_{ic})$$

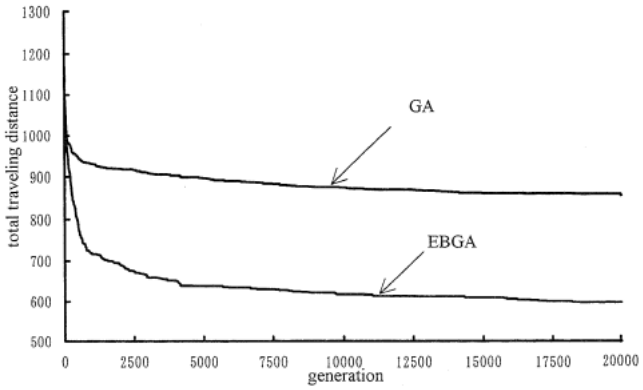
, gdzie $pr_{ic} = \frac{na_{ic}}{population_{size}}$
C - zbiór miast
na_{ic} -ilość wystąpień miasta c na(index kolumny) locus i H_i osiąga maksymalną wartość $ln(population_{size})$, gdy każde miasto z C występuje jednolicie, a 0 gdy jedno z miast występuje znacznie częściej niż pozostałe. H_i musimy oceniać na podstawie jakiejś stałej (próg(ang. threshold)) w tym przypadku $ln2$.Jeśli próg jest większy lub równy H_i to mamy niską różnorodność. Każde umiejscowienie porównujemy z progiem i jeśli jest większe od $\frac{n}{a}$ to zwiększamy podzielność. Parametr a to po prostu integer z przedziału [2,5]. Jak zwiększyć różnorodność? Korzystamy z procedury Med - Pop to znaczy, wybieramy $m = random(\frac{pop_{size}}{a}, pop_{size} - 1)$ chromosomów z populacji. W każdym wybranym chromosomie wymieniamy geny wzdłuż loca z innym mającym mniejszą różnorodność na locus niż próg.

3.2 Podsumowanie

1. Wygeneruj pop_{size} chromosomów losowo.
2. Ewaluacja wyznacz jakość każdego chromosomu (fitness) i oceń, który jest najlepszy $eval(t_k) = \frac{1}{\sum_{i=1}^{n} d(c_i,c_{i+1}) + d(c_n,c_1)}$
3. Krzyżowanie CX cycle crossover na wybranych chromosomach z prawdopodobieństwem pr
4. Mutacja zamiana miast z prawdopodobieństwem pm

5. selekcja wybierz pop_{size} chromosomów korzystając z roulette wheel selection.

6. Ulepsz różnorodność populacji korzystając z procedury med-pop.



Jak widać EBGA znajduje dużo lepsze rozwiązania niż GA.

4 Algorytm genetyczny metodą losowych kluczy dla GTSP

Tak jak już wspomnieliśmy GTSP to zgeneralizowany wariant powszechnie znanego TSP. Różnica polega na tym, że nie każde z miast musi zostać odwiedzone. Mianowicie zbiór miast C jest dzielony na m zbiorów rozłącznych tak, by ich suma była równa C, tzn każde miasto musi należeć do jakiegoś zbioru C_i. Tsp jest zatem specjalnym przypadkiem GTSP takim, że C jest podzielone na |C| podzbiorów. W tym przypadku skorzystamy z operacji reprodukcji, czyli skopiowaniu najlepszych elementów z populacji do nowej generacji. W tym problemie skorzystamy również z random keys w celu kodowania rozwiązań. Użycie ich jest możliwe, gdy nasz problem może opierać się na permutacjach integerów i , w których one- lub two- point crossover stwarza problemy. Przeanalizujmy metodę random- key na przykładzie. Załóżmy, że mamy ścieżkę 4 2 1 5 3 , nadajemy kolejnym elementom losową wartość (random key) na przedziale (0,1) i teraz do naszych kluczy przyporządkujemy wartości ze zbioru 1,...,n rosnąco. to znaczy, że jeśli mamy 0.1 0.42 0.3 0.9 0.7 to nasz ciąg zostanie zakodowany na 1 3 2 5 4. Wróćmy do naszego zbioru C podzielonego na załóżmy 3 podzbiory C_i, C_j, C_k. Najpierw wykorzystując random keye

decydujemy w jakiej kolejności odwiedzimy grupy miast załóżmy, że i -> j -> k. Następnie losujemy klucze dla grup miast i decydujemy o kolejności w tych podgrupach. W tym przypadku 20% populacji będzie przekazane do nowej generacji, 70% stworzone przez crossover, a pozostała część zostanie wygenerowana przez imigracje. Przy reprodukcji korzystamy z strategii elitizmu. Pozwala to na zachowanie odpowiednich do mieszania genów. Do stworzenia potomków używamy uniform crossover. Operacja imigracji co jakiś czas dodajemy po prostu generujemy nowe losowe rozwiązania, które dołączamy do nowej populacji. Ulepszenie każde nowe rozwiązanie uzyskane w GA staramy się poprawić swapując losowe miasta. Druga operacja to 2-opt próbująca usunąć dwa miejsca i wstawić je w inne miejsca tak, aby uzyskać pojedynczy tour o niższym koście. Swap pozwala zamienić miasta z innych komponentów C_i. Populacja zarządzamy w taki sposób, aby nie dodawać duplikatów, duplikaty to takie chromosomy, których ulepszenia są takie same. Zalety takiej implementacji to prostota implementacji, jest łatwa do przekształcenia dla innych problemów np MTP lub TCP. Schemat byłby niemal identyczy.