Řešení problému obchodního cestujícího pomocí lokálního prohledávání a evolučních algoritmů

David Herel

1) Úvod

Problém obchodního cestujícího je obtížný optimalizační problem, který jsme se rozhodli vyřešit pomocí lokálního prohledávání a evolučních algoritmů. Tyto dva postupy jsme navzájem porovnali.

1.2) Reprezentace:

Jako reprezentaci řešení jsem zvolil pořadí vrcholů, které určuje pořadí v němž místa navštívím.

1.3) Inicializace řešení:

Má funkce *createRandomValidSolution* vytváří náhodné validní řešení.

Funguje tím způsobem, že z počátečního vrcholu hledá další náhodný vrchol do kterého vede cesta. Tak pokračuje dokud nevyčerpá všechny vrcholy. Nakonec ještě kontroluje zda-li poslední vrchol má cestu zpátky do prvního. Pokud ne, algoritmus začíná odznovu.

1.4) Účelová funkce:

Fitness řešení určuji pomocí znalosti správného řešení. Vydělím délku cesty optimálního řešení tím mým. Fitness je tedy v rozmezí 0-1, přičemž 1 je nejlepší. Tedy vzorec na fitness je: optimal_solution/my_solution.

Funkce se jmenuje *countFitness*

1.5) Mutační operátor:

Můj mutační operátor je ve funkci: *mutationTwoPlaces*

Ten prohodí dvě místa v řešení, tak aby byla platná.

1.6) Lokální prohledávání:

Lokální prohledávání probíha v funkci *localSearchSwap2Places.*

Pokud dva vrcholy zvýší fitness řešení pak jsou prohozeny. Tato funkce je ve while cyklu a prohazuje všechny možné vrcholy dokud už žádné další prohození nezvýší fitness funkci řešení.

1.7) Operátor křížení:

Využívá Order crossover, vybere se tedy část rodiče A a zbytek je doplněn rodičem B s tím, že se žádné dva vrcholy nesmí opakovat. Probíhá to ve funkci *order_crossover*

1.8) Evoluční algoritmus:

Probíhá ve funkci *ea_run*. Nainicializuje populaci náhodných řešení. Spočítá jejich fitness. Ten kdo má vyšší fitness má vyšší šanci na rozmnožení tj. je víckrát v „matingPool“. Nahrazení staré populace funguje tím způsobem, že se náhodně z matingPool vyberou dva rodiče. Ti order_crossoverem vytvoří potomka, který pak ještě může být s nějakou pravděpodobnosti trošku zmutován. Toto se děje pořád dokola dokud někdo nedosáhne fitness 1 nebo pokud přesáhneme určitý počet generací.

2) Výsledky:

Experimenty byly prováděné na 3 instancích TSP LIBU. Konkrétně a280, bays29, bays29.

Výsledky, ale ukázali, že je úplně jedno jaké z těchto tří instancí používáme, protože pro všechny tři případy jsou výsledky víceméně identické.

Každý experiment byl puštěn 10-krát a výsledky zde uvedené jsou jejich průměrem.

2.1) Local search:

Náhodné řešení má v průměru fitness okolo **0.3**. Po aplikaci local searche jsme schopni dostat téměř **2.5x** lepší výsledek tj**. 0.7**-**0.8**.

Zkusil jsem i inicializovat local searche z více pozic. Tedy stejně jako jsme říkali na přednášce. Snažil jsem se startovat a pak hill climbovat z více pozic, abych našel více lokálních maxim a z nich vybral to nejlepší.

Při inicializaci z **100** náhodných míst jsem se výsledku velmi přiblížil: fitness cca **0.93**

Po navýšení na **1000** jsem už dostal hodnotu **0.98**.

Pořád to ale není to neojptimálnější. Zkusil jsem tedy **5000**. A voalá – dostal jsem téměř optimální řešení – **0.999**:

```
Local search experiments:

Random naive solution:
TSP length: 5690
Fitness: 0.35500878734622143

Optimized solution:
TSP length: 2906
Fitness: 0.6951135581555402

Starting from multiple random solutions and optimize each of those and choose the best:
Number of multiple starts is: 5000
Best solution: 0.9990108803165183

Process finished with exit code 0
```

2.2) Evoluční algoritmus:

Od evolučního algoritmu jsem čekal, že pomalu dokonverguje k lokálnímu maximu.

Nejdříve jsem zkusil dosáhnout řešení jen za pomocí order crossoveru. Ale i po tisíci generacích při populaci **500** řešení, se mi zdálo, že se průměrná fitness vůbec neposouvá.

```
Best fitness globally: 0.5702992659514399
----------
Generation: 999
Best fitness: 0.4717421765530126
Average fitness: 0.36385429535261465
Best fitness globally: 0.5702992659514399
----------
Generation: 1000
Best fitness: 0.4706430568499534
Average fitness: 0.3628049391663673
Best fitness globally: 0.5702992659514399
----------

Process finished with exit code 0
```

Bylo mi to divné – zkusil jsem experimentovat s velikosti populace – zvýšil ji 10-krát na 5000.

```
Best fitness globally: 0.6151035322777101
----------
Generation: 999
Best fitness: 0.515832482124617
Average fitness: 0.3621930116757166
Best fitness globally: 0.6151035322777101
----------
Generation: 1000
Best fitness: 0.5044955044955045
Average fitness: 0.3634806692742941
Best fitness globally: 0.6151035322777101
----------

Process finished with exit code 0
```

Nejlepší řešení se trošku zvýšilo, ale průměrné zůstalo téměř stejné.

Přišlo mi, že i když někdo dosáhne fitness o hodně větší než zbytek populace tak se to v dalších generacích vůbec neprojeví. Došlo mi, že musím zvýšit šanci na rozmnožení těch nejlepších.

Dříve jsem do matingPoolu přidával řešení tolikrát kolik mělo fitness * 100.

Vzorec: **fitness*100**

Rozhodl jsem se, že fitness každého po vynásobení stem budu dávat na druhou a zvýším tím, tedy pravděpodobnost, že ti s nejlepším fitness budou vybraní.

Vzorec: **(fitness*100)^2**

```
Best fitness globally: 0.6400506970849176
----------
Generation: 999
Best fitness: 0.5471289274106176
Average fitness: 0.38400093868576485
Best fitness globally: 0.6400506970849176
----------
Generation: 1000
Best fitness: 0.5359511806845317
Average fitness: 0.3838418501655228
Best fitness globally: 0.6400506970849176
----------

Process finished with exit code 0
```

Trochu to také zabralo, ale pořád to nebyly výsledky jaké jsem čekal, průměrná i nejlepší fitness se zvýšila jen o trochu.

Zkusil jsem přidat **mutaci**. Mutaci jsem zvolil jako prohození dvou náhodných míst.

Každého nového člena populace zmutuji –každý jeho vrchol s určitou pravděpodobnosti vyměním s nějakým náhodně zvoleným.

Pravděpodobnost mutace jsem zvolil jako 0,01 (tedy každá 100-tý vrchol se prohodí)

```
Best fitness globally: 0.6029850746268657
----------
Generation: 999
Best fitness: 0.5162279580884233
Average fitness: 0.37368386449609764
Best fitness globally: 0.6029850746268657
----------
Generation: 1000
Best fitness: 0.572562358276644
Average fitness: 0.37412998926760443
Best fitness globally: 0.6029850746268657
----------

Process finished with exit code 0
```

Ale výsledky jsem dostal ještě horší.

Zkusil jsem si ještě pohrát s hodnotou mutace, ale výsledky mě frustrovali čím dál více. Nevím, kde dělám chybu, ale i po fine tunování parametru jsem se dostal na maximální hodnoty průměrné fitness 0.38 a maximální 0.65.

3) Závěr

Z experimentů jasně vyplývá, že lokální prohledávání se chová lépe ve všech případech, bez ohledu na dataset.

Musíme ovšem zmínit, že výsledky našeho evolučního algoritmu, z mně neznámých důvodů, nejsou tak optimální jak by být mohli. A věřím, že evoluční algoritmus by se dokázal dostat do řešení také, ovšem otázkou je jestli rychleji?