

AI based Software Engineering Assignment #1

20170344 Minjae Song

Intro

At beginning, I used usual evolutionary computation. For usual selection, I used tournament selection. For crossover, I used sequence crossover. For generational selection, I used elitism. My method's process is as follows.

```
<sudo code for evolutionary computation>
init population
loop for #generation{
    evaluate fitness
    crossover for children
    mutation
    select for next generation
}
```

I used python. My most significant element of testing was “time”. There are several approaches for TSP problem (e.g. ACO, PSO...). But, I think using ACO or PSO needs reasonable population size and generation number to see some improvements. Because ACO and PSO have merits in aspect of interaction between each individual and, with lower population size and generation number which is my case, I thought they don't have such thing to convey between individuals.

Result

Test for optimization and finding hyper parameter at first.

For understanding, I placed each parameter value and short description.

<Trial #1>

Population size = 30

Generation size = 100

Mutation rate = 0.2

Result = 86421898.99291812

Basic condition. Did not use elitism. Just put all children into next generation.

<Trial #2>

Population size = 100

Generation size = 10

Mutation rate = 0.2

Result = 86315003.07999954

By comparing #2 result with #1, I wanted to see the which parameter is more important between generation size and population size. Other than two parameter, all condition same as #1.

<Trial #3>

Population size = 10

Generation size = 200

Parent number = 2

Result = 86644643.55999997

With high generation size however, results in meaningless offspring.

<Trial #4>

Population size = 100

Generation size = 100

Parent number = 10

Elite number= 8

Mutation rate = 0.2

Result2 = 85949858.64000027

Brought in Elitism. From #3, it seems mutation does not bring offspring to development. So, I thought that bringing elite from parents generation may lead generation to better results. Actually, it improved very little.

<Trial #5> gradual replacement

Population size = 100

Generation size = 100

Parent number = 80

Elite number= 8

Mutation rate = 0.2

Result = 85630890.1094121

Used gradual replacement for generational selection. Thought to lead generation into proper (which brings more development) direction, not much children is needed.

Optimization

From several tests, I realized that just pushing generation into many loops or putting more population is like waiting for a miracle. Therefore, I decided to optimize in “init population” part.

Padding for initialized population

Using law of large number, I picked 100 individual and picked best of them. Then, they become one of the initialized population. By doing this, each init population is at least better than 100 different routes.

Local Search

For better initial population, I also used local search. Here comes fitness evaluation limit parameter for limiting local search.

Also, for finding local optima, which we want to find, after number of fitness evaluation limit, I used improvement evaluation. So, even we passed fitness evaluation limit, if local search is still in improvement, fitness evaluation is keep going on.

```
improve = (pre_fit - route_fit) / pre_fit
```

Crossover Method

To make evolutionary computation meaningful, I thought that effective crossover method was needed. Several ways to execute crossover.

1. Partially Mapped Crossover(PMC, PMX)

This approach was based on randomly picking points. After randomly picking, slice parent's gene into three parts and replace one part with other parent's part of same slice. Then, placing empty parts using relationship earned by two parents' slices which was exchanged.

2. Cycle Crossover(CX)

Main idea of CX is to conserve one parent's cycle. By picking cycle in one's parent, remaining part will be covered by other parent's gene.

There are other Crossover methods. But, from my search, many of them are based on either PMC or CX. Plus, even pure CX costs a lot of time. CX made about 10minutes for one generation which is clearly heavier than PMC or normal crossover.

Improvement Result

<Trial #6>

Population size = 5

Fitness evaluation limit = 2000 (only for local search per one init individual)

Generation size = 10

Parent number = 2

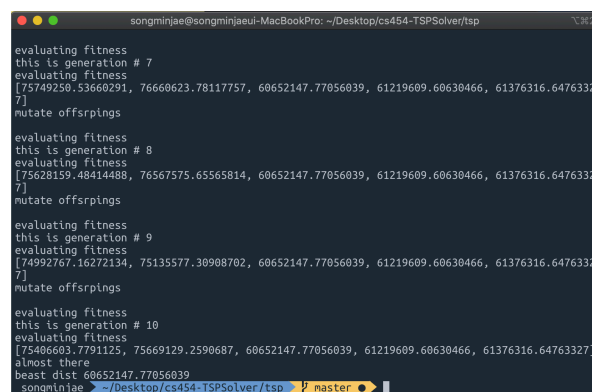
Elite number= 3

Mutation rate = 0.2

Neighbor size = 50

Result = 60652147.77056039

From now all test condition is having gradual replacement, padding for init population, local search with neighbors.



```
songminjae@songminjaeui-MacBookPro: ~/Desktop/cs454-TSPSolver/tsp
evaluating fitness
this is generation # 7
evaluating fitness
[75749259.53660291, 76660623.78117757, 60652147.77056039, 61219609.60630466, 61376316.6476332
7]
mutate offsprings
evaluating fitness
this is generation # 8
evaluating fitness
[75628159.48414488, 76567575.65565814, 60652147.77056039, 61219609.60630466, 61376316.6476332
7]
mutate offsprings
evaluating fitness
this is generation # 9
evaluating fitness
[74992767.16272134, 75135577.30908702, 60652147.77056039, 61219609.60630466, 61376316.6476332
7]
mutate offsprings
evaluating fitness
this is generation # 10
evaluating fitness
[75406603.7791125, 75669129.2590687, 60652147.77056039, 61219609.60630466, 61376316.64763327]
almost there
best dist 60652147.77056039
songminjae ~/Desktop/cs454-TSPSolver/tsp master
```

List of fitness for each generation

<Trial #7>

Population size = 5
 Fitness evaluation limit = 2000 (only for local search per one init individual)
 Generation size = 10
 Parent number = 2
 Elite number = 3
 Mutation rate = 0.2
 Neighbor size = 100
 Result = 57158022.04678965

Increased neighbor size to 100. I couldn't handle variance of neighbors. So, I increased possibility of diversity by picking more neighbors. Meaning that randomly pick 2 points to tweak route for 100 times makes routes neighbors more diverse than 50 times.

This method can control diversity and makes control of neighbor size in probability side, but makes calculation time significantly increase and anyway it is still probability approach.

However, it increases result. But, downgraded in approach of time consumed.

<Trial #8>

Fitness evaluation limit = infinite (only for local search per one init individual)
 Neighbor size = 100

each row (number of evaluation in local search, evaluated distance)

I wanted to find the limit of local search. To help this, I set fitness evaluation limit to infinity and monitored improvements.

The result was that when we approached about 8000th local search, it seems improvement came close to limit. The improvement of distance was about thousand units scale. This is significantly lower improvement compared to thousand -hundred thousand units scale improvement under 2000th local search.

<Trial #9>

Population size = 5
 Fitness evaluation limit = 500 (only for local search per one init individual)
 Generation size = 50
 Parent number = 2
 Elite number = 3
 Mutation rate = 0.2
 Neighbor size = 50
 Result = 77682530.84404702


```

python3 tsp_solver.py -f rl1849.tsp -f 200 -g 500 -m 0.0005
if city not in sub_r1:
KeyboardInterrupt

X songminjae ~/Desktop/cs454-TSPsolver/tsp P master
X songminjae ~/Desktop/cs454-TSPsolver/tsp P master python tsp_solver.py -f rl18
49.tsp -f 200 -g 500 -m 0.0005
init pop's fitness [86019756.16151282, 86007330.52625881, 85971040.81828415, 86036940.9812876
6, 86021020.66548653]
local search for init pops
improved 9.510601108947538e-05 pre_fit: 82708944.59086062 this_fit: 82782070.76942259
improved 9.227064921162800e-05 pre_fit: 82993697.05273941 this_fit: 82986039.17043188
improved 9.70844751729675e-05 pre_fit: 82286476.19085166 this_fit: 82278494.03441493
improved 8.86274592491072e-05 pre_fit: 83039676.15091802 this_fit: 83032316.57311827
improved 8.997251524938222e-05 pre_fit: 82997281.87139973 this_fit: 82989814.3971909
local fitness [82782070.76942259, 82986039.17043188, 82278494.03441493, 83032316.57311827, 82
989814.3971909]
this is generation # 1
evaluating fitness
[82782070.76942259, 82986039.17043188, 82278494.03441493, 83032316.57311827, 82989814.3971909]
mutate offsprings
evaluating fitness
this is generation # 2
evaluating fitness
[82999475.73345074, 82999475.73345074, 82278494.03441493, 82782070.76942259, 82986039.1704318
8]

```

This makes evolutionary child better competitiveness.
And as generation flows, they make mutual improvements.

```

songminjae@songminjaeui-MacBookPro: ~/Desktop/cs454-TSPsolver/tsp
evaluating fitness
this is generation # 497
evaluating fitness
[77665958.25610068, 77665958.25610068, 77647942.66737477, 77647942.66737477, 77669541.2913879
9]
mutate offsprings
evaluating fitness
this is generation # 498
evaluating fitness
[77653142.5825647, 77653142.5825647, 77647942.66737477, 77647942.66737477, 77665958.25610068]
mutate offsprings
evaluating fitness
this is generation # 499
evaluating fitness
[77652229.91372627, 77652229.91372627, 77647942.66737477, 77647942.66737477, 77653142.5825647]
mutate offsprings
evaluating fitness
this is generation # 500
evaluating fitness
[77623961.06319308, 77623961.06319308, 77647942.66737477, 77647942.66737477, 77652229.9137262
7]
almost there
best dist 77623961.06319308
songminjae ~/Desktop/cs454-TSPsolver/tsp P master

```

<Trial #11>

Population size = 5
Fitness evaluation limit = 200 (only for local search per one init individual)
Generation size = 500
Parent number = 2
Elite number = 3
Mutation rate = 0.0005
Neighbor size = 50
Result = 76935479.384544

```

songminjae@songminjaeui-MacBookPro: ~/Desktop/cs454-TSPsolver/tsp
evaluating fitness
this is generation # 497
evaluating fitness
[76964207.38582887, 76964207.38582887, 76946484.43012753, 76946484.43012753, 76956233.6203929]
mutate offsprings
evaluating fitness
this is generation # 498
evaluating fitness
[76948879.41362514, 76948879.41362514, 76946484.43012753, 76946484.43012753, 76956233.6203929]
mutate offsprings
evaluating fitness
this is generation # 499
evaluating fitness
[76937966.09056951, 76937966.09056951, 76946484.43012753, 76946484.43012753, 76948879.4136251
4]
mutate offsprings
evaluating fitness
this is generation # 500
evaluating fitness
[76935479.384544, 76935479.384544, 76937966.09056951, 76937966.09056951, 76946484.43012753]
almost there
best dist 76935479.384544
songminjae ~/Desktop/cs454-TSPsolver/tsp P master

```

Everything is same, except I used PMC(partially mapped crossover)

But, there exists pair inside same generation, due to reduction of mutation rate, not much diversity exists.
Therefore, many best_child(which is best picked in crossedover, mutated child group) sticks in one generation.

To improve pair in same generation problem, I increased mutation rate a little from #12. It fixes little, as you can see from image, generation 2 has different 3th and 4th fitness at #12-3.

Also, since CX takes much time, so I reduced generation size to 20.

<Trial #12-1>

I tried normal crossover for comparison.

Population size = 5

Fitness evaluation limit = 20(only for local search per one init individual)

Generation size = 20

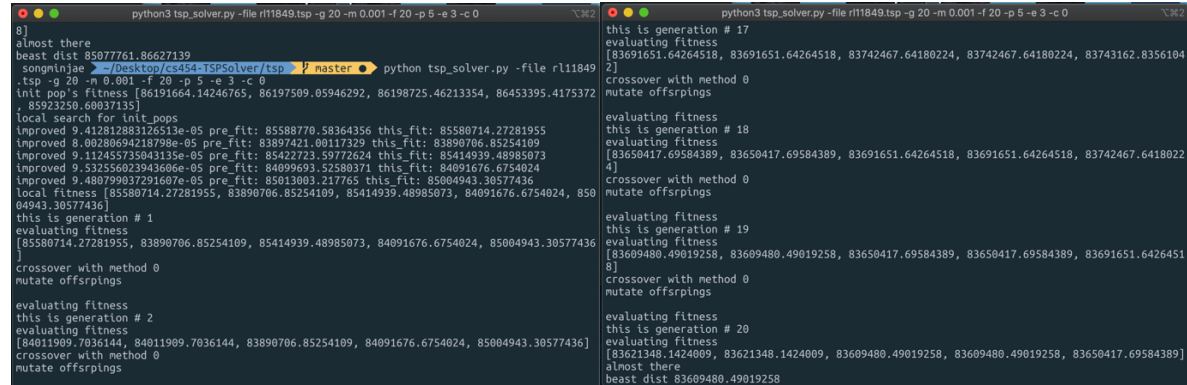
Parent number = 2

Elite number= 3

Mutation rate = 0.001

Neighbor size = 50

Result = 83609480.49019258



```
python3 tsp_solver.py -file rl11849.tsp -g 20 -m 0.001 -f 20 -p 5 -e 3 -c 0
almost there
beast dist 85077761.86627139
songminjae ~/Desktop/cs454-TSPSolver/tsp master python tsp_solver.py -file rl11849.tsp -g 20 -m 0.001 -f 20 -p 5 -e 3 -c 0
init pop's fitness [86191664.14246765, 86197509.05946292, 86198725.46213354, 86453395.4175372, 85923250.60037135]
local search for init pops
improved 9.412812883126513e-05 pre_fit: 85588770.58364356 this_fit: 85588714.27281955
improved 8.00280894218798e-05 pre_fit: 83897421.08117329 this_fit: 83898706.85254109
improved 9.112455735043135e-05 pre_fit: 85422723.59772624 this_fit: 85414939.48985073
improved 9.532556023943606e-05 pre_fit: 84099693.52580371 this_fit: 84091676.6754024
improved 9.480799037291607e-05 pre_fit: 85013003.217765 this_fit: 85004943.30577436
local fitness [85588714.27281955, 83898706.85254109, 85414939.48985073, 84091676.6754024, 85004943.30577436]
this is generation # 1
evaluating fitness
[85588714.27281955, 83898706.85254109, 85414939.48985073, 84091676.6754024, 85004943.30577436]
crossover with method 0
mutate offsprings
evaluating fitness
this is generation # 2
evaluating fitness
[8401909.7036144, 84011909.7036144, 83898706.85254109, 84091676.6754024, 85004943.30577436]
crossover with method 0
mutate offsprings
evaluating fitness
this is generation # 17
evaluating fitness
[83691651.64264518, 83691651.64264518, 83742467.64180224, 83742467.64180224, 83743162.83561042]
crossover with method 0
mutate offsprings
evaluating fitness
this is generation # 18
evaluating fitness
[83650417.69584389, 83650417.69584389, 83691651.64264518, 83691651.64264518, 83742467.64180224]
crossover with method 0
mutate offsprings
evaluating fitness
this is generation # 19
evaluating fitness
[83609480.49019258, 83609480.49019258, 83650417.69584389, 83650417.69584389, 83691651.64264518]
crossover with method 0
mutate offsprings
evaluating fitness
this is generation # 20
evaluating fitness
[83621348.1424009, 83621348.1424009, 83609480.49019258, 83609480.49019258, 83650417.69584389]
almost there
beast dist 83609480.49019258
```

<Trial #12-2>

I also tried TMC

Population size = 5

Fitness evaluation limit = 20(only for local search per one init individual)

Generation size = 20

Parent number = 2

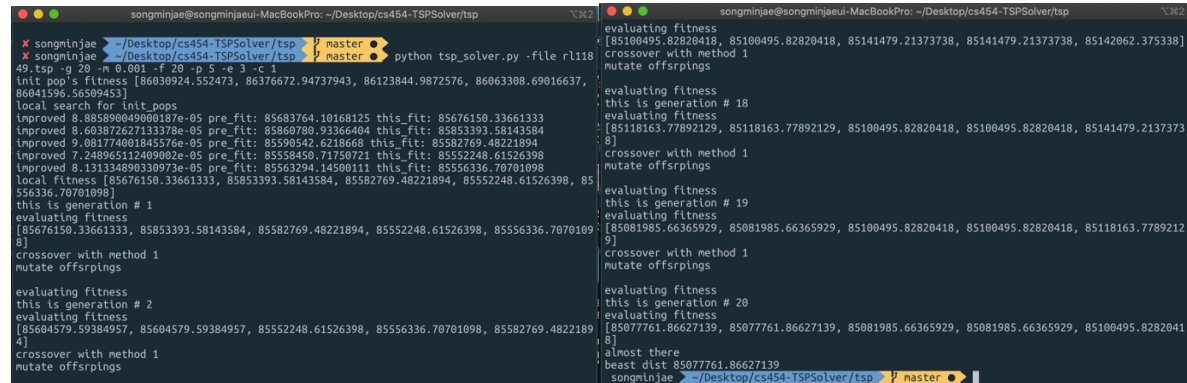
Elite number= 3

Mutation rate = 0.001

Neighbor size = 50

Result = 85077761.86627139

85552248.61526398



```
songminjae@songminjaeui-MacBookPro: ~/Desktop/cs454-TSPSolver/tsp
songminjae ~/Desktop/cs454-TSPSolver/tsp master python tsp_solver.py -file rl11849.tsp -g 20 -m 0.001 -f 20 -p 5 -e 3 -c 1
init pop's fitness [86030924.352473, 86376672.94737943, 86123844.9872576, 86063308.69016637, 86041596.56509453]
local search for init pops
improved 8.885890049000187e-05 pre_fit: 85683764.10168125 this_fit: 85676150.33661333
improved 8.00280894218798e-05 pre_fit: 85869780.93366404 this_fit: 85853393.58143584
improved 9.081774001845576e-05 pre_fit: 85598542.6218668 this_fit: 85582769.48221894
improved 7.24896511240902e-05 pre_fit: 8558450.71750721 this_fit: 85552248.61526398
improved 8.1313489030973e-05 pre_fit: 85563294.14500111 this_fit: 85556336.70701098
local fitness [85676150.33661333, 85853393.58143584, 85582769.48221894, 85552248.61526398, 85556336.70701098]
this is generation # 1
evaluating fitness
[85676150.33661333, 85853393.58143584, 85582769.48221894, 85552248.61526398, 85556336.70701098]
crossover with method 1
mutate offsprings
evaluating fitness
this is generation # 2
evaluating fitness
[85604579.59384957, 85604579.59384957, 85552248.61526398, 85556336.70701098, 85582769.48221894]
crossover with method 1
mutate offsprings
evaluating fitness
this is generation # 18
evaluating fitness
[85100495.82820418, 85100495.82820418, 85141479.21373738, 85141479.21373738, 85142862.375338]
crossover with method 1
mutate offsprings
evaluating fitness
this is generation # 19
evaluating fitness
[85081985.66365929, 85081985.66365929, 85100495.82820418, 85100495.82820418, 85141479.21373738]
crossover with method 1
mutate offsprings
evaluating fitness
this is generation # 20
evaluating fitness
[85077761.86627139, 85077761.86627139, 85081985.66365929, 85081985.66365929, 85100495.82820418]
almost there
beast dist 85077761.86627139
songminjae ~/Desktop/cs454-TSPSolver/tsp master
```

<Trial #12-3>

I also tried CX(cycle crossover)

Population size = 5

Fitness evaluation limit = 20(only for local search per one init individual)

Generation size = 20

Parent number = 2

Elite number= 3

Mutation rate = 0.001

Neighbor size = 50

Result = 83780285.66349736


```
python3 tsp_solver.py -file rl11849.tsp -g 200 -m 0.005 -f 500 -p 5 -e 3 -c 1
KeyboardInterrupt
songminjae ~/Desktop/cs454-TSPSolver/tsp master
songminjae ~/Desktop/cs454-TSPSolver/tsp master
songminjae ~/Desktop/cs454-TSPSolver/tsp master python tsp_solver.py -file rl118
49.tsp -g 200 -m 0.005 -f 500 -p 5 -e 3 -c 1
init pop's fitness [85926919.65695709, 85963986.67650938, 86245004.15502323, 86174641.9331864
6, 86316815.83879225]
local search for init pops
improved 9.445856677191577e-05 pre_fit: 78520554.22795711 this_fit: 78513137.2889426
improved 9.058297920632327e-05 pre_fit: 78614087.76126316 this_fit: 78606966.66298616
improved 8.10735006611748e-05 pre_fit: 78724038.95935749 this_fit: 78717656.52593286
improved 8.58929635897133e-05 pre_fit: 78456596.33139549 this_fit: 78449857.27880229
improved 9.470416526150055e-05 pre_fit: 78556195.33693436 this_fit: 78548755.73802885
local fitness [78513137.2889426, 78606966.66298616, 78717656.52593286, 78449857.27880229, 785
48755.73802885]
this is generation # 1
evaluating fitness
[78513137.2889426, 78606966.66298616, 78717656.52593286, 78449857.27880229, 78548755.73802885]
crossover with method 1
mutate offsprings
evaluating fitness
this is generation # 2
evaluating fitness
[79943079.2910503, 79943079.2910503, 78449857.27880229, 78513137.2889426, 78548755.73802885]
crossover with method 1
mutate offsprings
evaluating fitness
this is generation # 198
evaluating fitness
[78893414.32680471, 78893414.32680471, 78449857.27880229, 78513137.2889426, 78548755.73802885]
crossover with method 1
mutate offsprings
evaluating fitness
this is generation # 199
evaluating fitness
[79481675.68695028, 79481675.68695028, 78449857.27880229, 78513137.2889426, 78548755.73802885]
crossover with method 1
mutate offsprings
evaluating fitness
this is generation # 200
evaluating fitness
[80052103.00091937, 80052103.00091937, 78449857.27880229, 78513137.2889426, 78548755.73802885]
almost there
beast dist 78449857.27880229
songminjae ~/Desktop/cs454-TSPSolver/tsp master
```

<Trial #13-3>

Population size = 5

Fitness evaluation limit = 500(only for local search per one init individual)

Generation size = 200

Parent number = 2

Elite number= 3

Mutation rate = 0.001

Neighbor size = 50

Result = 76324734.3507345

```
python3 tsp_solver.py -file rl11849.tsp -g 200 -m 0.001 -f 500 -p 5 -e 3 -c 1
evaluating fitness
this is generation # 200
evaluating fitness
[80052103.00091937, 80052103.00091937, 78449857.27880229, 78513137.2889426, 78548755.73802885]
almost there
beast dist 78449857.27880229
songminjae ~/Desktop/cs454-TSPSolver/tsp master python tsp_solver.py -file rl11849
.tsp -g 200 -m 0.001 -f 500 -p 5 -e 3 -c 1
init pop's fitness [86286605.10995771, 86026282.31973334, 86250460.05155355, 86074471.3764883
, 85880176.7194515]
local search for init pops
improved 9.332387304964748e-05 pre_fit: 78778364.16183496 this_fit: 78771012.25977886
improved 7.222749378926941e-05 pre_fit: 78852992.30543092 this_fit: 78647311.39691767
improved 9.411949886359449e-05 pre_fit: 78832240.4959538 this_fit: 78824820.84498402
improved 9.59586868109181e-05 pre_fit: 77800584.79193437 this_fit: 77793119.14999472
improved 8.232153630762475e-05 pre_fit: 78626672.76519279 this_fit: 78620200.096696
local fitness [78771012.25977886, 78647311.39691767, 78824820.84498402, 77793119.14999472, 78
620200.096696]
this is generation # 1
evaluating fitness
[78771012.25977886, 78647311.39691767, 78824820.84498402, 77793119.14999472, 78620200.096696]
crossover with method 1
mutate offsprings
evaluating fitness
this is generation # 2
evaluating fitness
[78464656.48857377, 78446456.48857377, 77793119.14999472, 78620200.096696, 78647311.39691767]
evaluating fitness
this is generation # 197
evaluating fitness
[76343575.4037028, 76343575.4037028, 76359271.33333936, 76359271.33333936, 76403755.97511016]
crossover with method 1
mutate offsprings
evaluating fitness
this is generation # 198
evaluating fitness
[76324734.3507345, 76324734.3507345, 76343575.4037028, 76343575.4037028, 76359271.33333936]
crossover with method 1
mutate offsprings
evaluating fitness
this is generation # 199
evaluating fitness
[76331239.60169572, 76331239.60169572, 76324734.3507345, 76324734.3507345, 76343575.4037028]
crossover with method 1
mutate offsprings
evaluating fitness
this is generation # 200
evaluating fitness
[76374222.26418586, 76374222.26418586, 76324734.3507345, 76324734.3507345, 76331239.60169572]
almost there
beast dist 76324734.3507345
songminjae ~/Desktop/cs454-TSPSolver/tsp master
```

<Trial #13-4>

Population size = 5

Fitness evaluation limit = 500(only for local search per one init individual)

Generation size = 200

Parent number = 2

Elite number= 3

Mutation rate = 0.0005

Neighbor size = 50

Result = 76428599.97131659

```
songminjae@songminjae-MacBookPro: ~/Desktop/cs454-TSPSolver/tsp
5]
crossover with method 1
mutate offsprings
evaluating fitness
this is generation # 198
evaluating fitness
[76456409.70881085, 76456409.70881085, 76448348.46415688, 76448348.46415688, 76474888.0595482
4]
crossover with method 1
mutate offsprings
evaluating fitness
this is generation # 199
evaluating fitness
[76428599.97131659, 76428599.97131659, 76448348.46415688, 76448348.46415688, 76456409.7088108
5]
crossover with method 1
mutate offsprings
evaluating fitness
this is generation # 200
evaluating fitness
[76460840.70099849, 76460840.70099849, 76428599.97131659, 76428599.97131659, 76448348.4641568
8]
almost there
beast dist 76428599.97131659
songminjae ~/Desktop/cs454-TSPSolver/tsp master
```


Conclusion

Generational Selection

Comparing #4, #5 to #1, #2, #3 generational selection with elitism and gradual replacement makes little improvement.

Local Search

Local search improves distance much.

Appropriate threshold for improvement was about 0.0001. This caused about 30 -40 extra search when I placed fitness evaluation limit for 2000.

Local search is surely good optimization approach, but just putting it too much will not improve as expected. From my test, about 8000 number of local search makes improvement rate dull.

8000 local search for each individual, will improve init_pop's fitness to about 25000000.

Crossover

From crossover part, we calculated improvement each since 12-1 has too good initial population in aspect of fitness. Improvement was calculated by

$$\text{improvement} = (\text{generation \#1 best} - \text{generation \#20 best}) / \text{generation \#1 best}$$

Improvement(%)

#12-1 : 0.003352294585416

#12-2 : 0.00554616338755

#12-3 : 0.006269424578703

Results shows that CX is best at improvements. But, comparing #12-2, #12-3, CX takes too much time. So, when we consider "time consumed" for decision, TMC is not a bad choice.

Mutation Rate

As mentioned above, mutation rate 0.2 is too high, which makes children not competitive. But, also pushing rate into too low (0.0005), make same pair children in one generation problem. This will make population size meaningless.

From #13, we can find that high mutation rate like 0.005 makes uncompetitive child. Mutation rate less than 0.002 was effective for evolutionary computation.

To short, from above results, I expect result to be best when I use [fitness limit(only for local search per one init individual) : 8000, crossover : CX, mutation rate : 0.002, gradual replacement]

Future Work

From my search about crossover methods, there are diverse methods improved from TMC and CX. From TMC, UTMX (uniform partially-mapped crossover), which uses probability of correspondence for each iteration and using it to slice genes. Also, CX2 which was introduced from paper may also improve fitness.

For testing, time cost was too much burden for me. Especially, CX crossover, widening neighbor size takes really much time for testing several times even though they improved fitness a lot. Generally, most of test were hard to execute multiple times. Approximately, for one individual's local search with fitness limit 2000 took about 33minute. This leads population initialization to take about 3hours when we have 5 population size.

From my friend's case, he had GPU server and ran his codes in parallel way. If I have some chance to ran in that kinds of condition, I want to run my code with parameters that I think will lead best result.

Also, bio-inspired algorithms like PSO, ACO may improve results.

<Reference>

- Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator
<https://www.hindawi.com/journals/cin/2017/7430125/>
- A Comparative Study of Adaptive Crossover Operators for Genetic Algorithms to Resolve the Traveling Salesman Problem
<https://arxiv.org/pdf/1203.3097.pdf>

***details about parameters in my code.

Args

1. -file, --file_name

2. -p, --population_size, population size of one generation.
3. -f, --fitness_evaluation, number of fitness evaluation for one individual's "local search"
4. -g, --generation_size, number of generation size in evolutionary loop
5. -parent, --parent_number, number of parent which crossover and makes offsprings
6. -e, --elite_number, number of elites in generation and elites remains after generational selection
7. -m, --mutation_rate, percent of mutated gene in mutation operation
8. -n, --neighbor_size, number of neighbor's size in local search
9. -c, --crossover_method, 0 for normal crossover 1 for TMC and 2 for CX
10. -padd, --padding_number, number of padding when init population
11. -i, --improve_limit, limit for improvements

Number of fitness evaluation is spreaded in my code.

Fitness evaluation executed

1. Local search => fitness_limit * pop_size * neighbor_size
2. init pop => padding * pop_size
3. evaluate fitness in evolutionary loop => pop_size * generation_limit
4. mutate_off => pop_size * generation_limit

+ Adding these 4 parts will result in total fitness evaluation number.

Furthermore, there is improvement limit which extends fitness evaluation. If improvement does not meet this value, they keep doing local search. If you want to turn off this functionality, just put enormous number to improve_limit(e.g. 100.1, 99.444...)