Final report

* Problem Description:

The Sequential Ordering Problem (SOP) with precedence constraints consists of finding a minimum weight Hamiltonian path on a directed graph with weights on the arcs and on the nodes, subject to precedence constraints among nodes.

* Instances Description:

Instances are provided by TSPLIB that it is a library of sample instances for the TSP (and related problems like SOP, ATSP, HCP) from various sources and of various types.

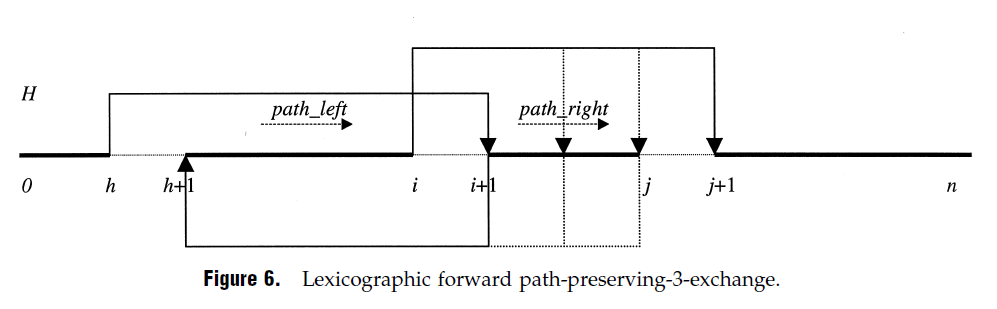
Each instance file consists of two part as **specification part** that contains information about the instance data and **data part**.

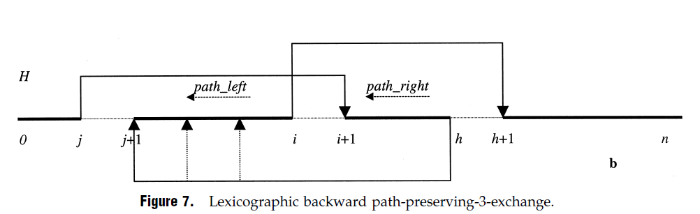
* Algorithm Description:

The algorithm designed base on the related paper as *(An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem).*

**Constructive heuristic** used for generating initial solution in the way that from the bingeing each time minimum possible length edge based on precedence condition selected.

For neighboring method to move from current solution to another, **Lexicographic Search** using **forwarding and back warding path-preserving-3-exchange** applied and best solution selected among them.





The only difference is that in this algorithm lexicographic search doesn’t applied on whole search space by iteratively change the parameters “h, i, j”, instead random “h” generated and according to that random “i,j” created to do the search.

With the use of loop with size half of dimension forward and with same size loop backward exchanging applied.

It means that in each simulated annealing iteration best solution selected from a list of solutions with size of problem dimension.

* + Initial Constructive heuristic
  + O(dimension/2) forward searching with random “h, i, j” parameters.
  + O(dimension/2) backward searching with random “h, i, j” parameters.
  + Selecting the best from search as next solution
* Algorithm time complexity:

for it in range(*int*(dimension/2)):

          h = randrange(0, dimension-3)

         i = h + 1

…

for j in range(i, len(solution)):

             for dep in deps[solution[j]]:

…

As code shows the forward and backward search consist of 3 loops so the time complexity is O(n3).

def get\_neighbor(problem, dependencies, state, cost):

    …

    new\_state1 = fpp3exchange(problem, dependencies, state)

    new\_state2 = bpp3exchange(problem, dependencies, state)

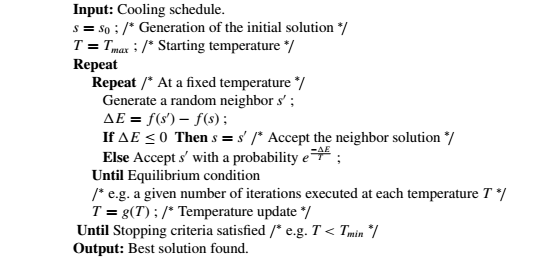
…

and the neighboring function calling both of them for selecting new solution so

the searching algorithm complexity is O(n3).

For updating the temperature 3 methods (**linear** and **logarithmic** and **exponential**) applied to find the best to work with.

* Simulated annealing algorithm progress:
  + Algorithm template



T = 1

ALPHA = 0.8 (for using in temperature updating)

TEMP\_MODE = EXP (temperature updating method)

INIT\_HEURISTIC = True (using initial heuristic)

NUM\_ITERATIONS = 500

* + Algorithm progress plot for sample instances:



p43.4.sop jpeg.4753.54.sop

The whole results (main, max, avg) came at the end.

* Simulated annealing Initial methods comparison:

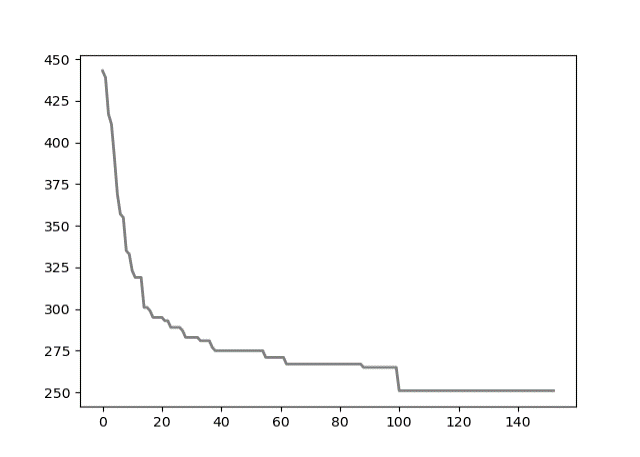
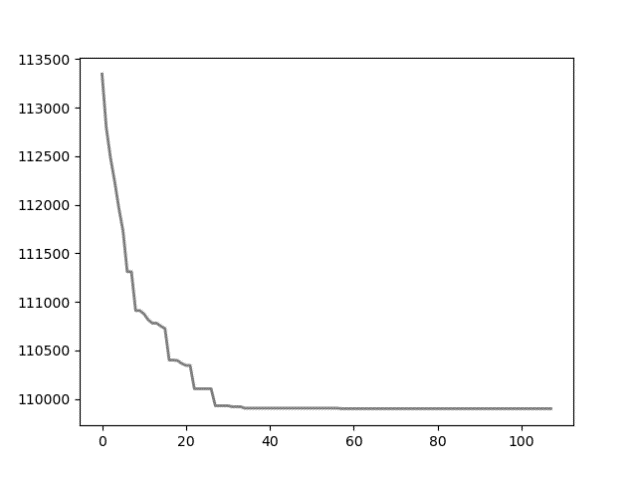
T = 1

ALPHA = 0.8

TEMP\_MODE = EXP

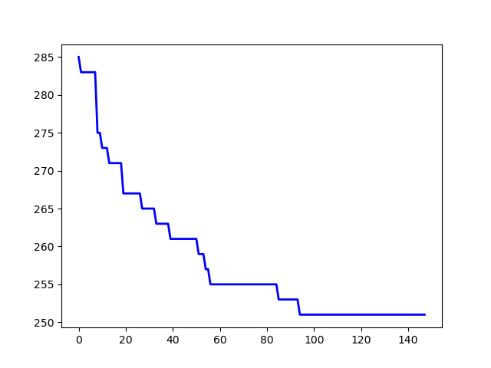
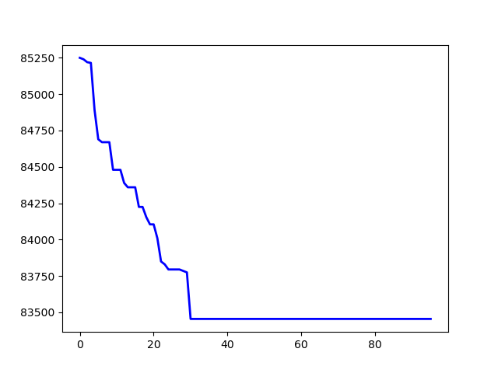
NUM\_ITERATIONS = 500

* + Random



*p43.4.sop* *jpeg.4753.54.sop*

* + Heuristic:



*p43.4.sop* *jpeg.4753.54.sop*

as result shows with heuristic method algorithm start from much better initial solution (lower value) and in some cases leads to better final solution.

For 10 instances as test heuristic method gave better solution.

* Simulated annealing temperature update methods comparison:

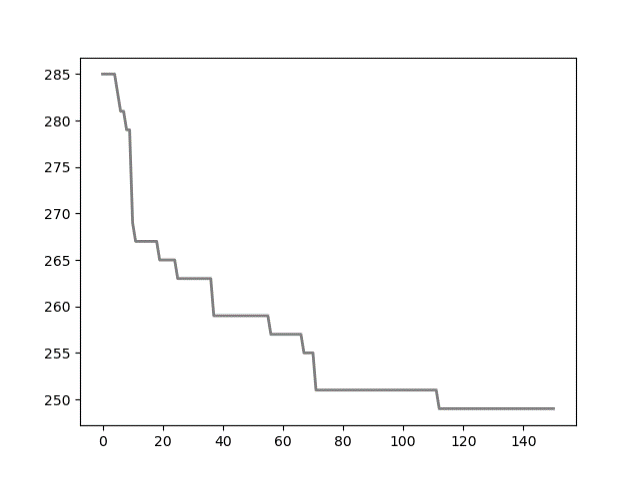
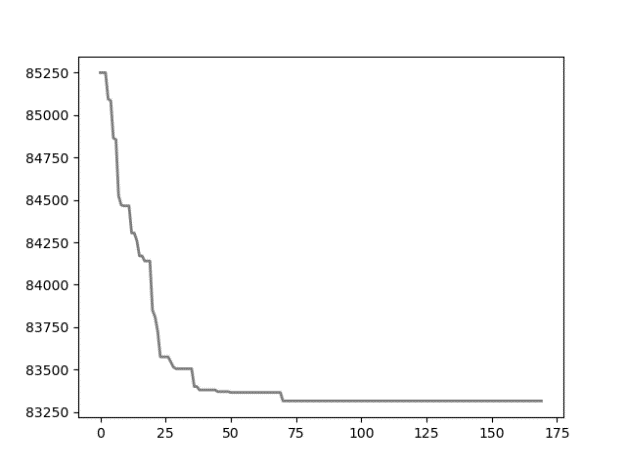
T = 1

ALPHA = 0.9

INIT\_HEURISTIC = True

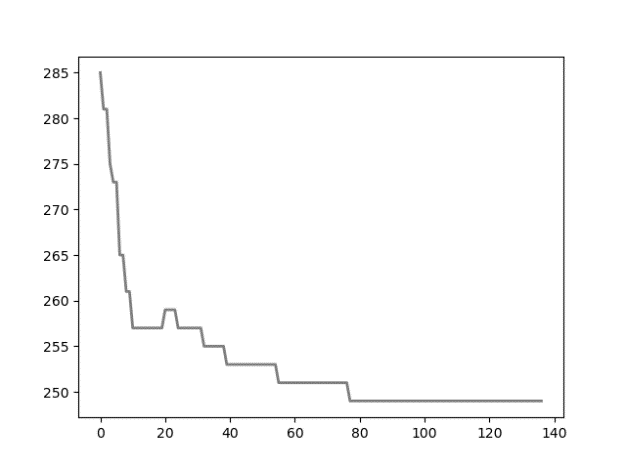
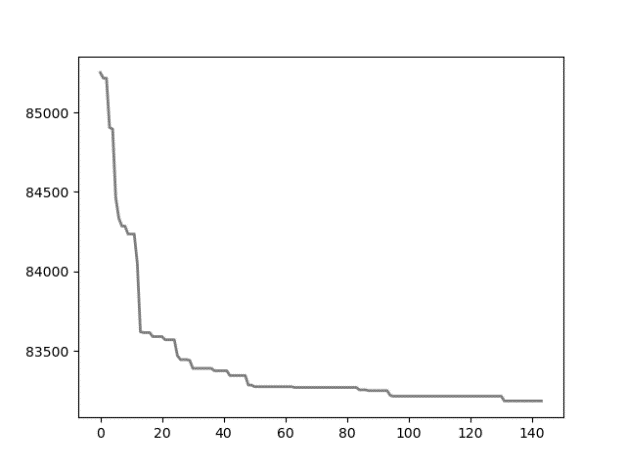
NUM\_ITERATIONS = 500

* + Linear (ALPHA \* T):



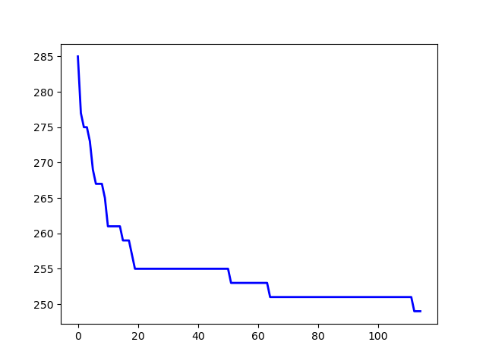
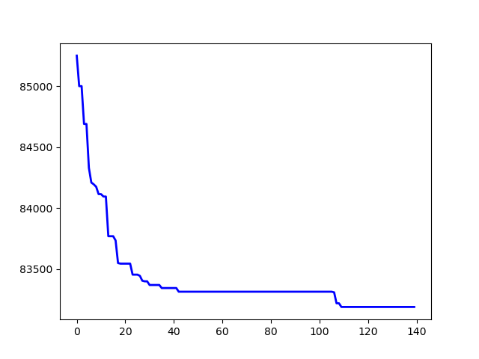
*p43.4.sop* *jpeg.4753.54.sop*

* + Logarithmic (T0 / math.log(step)):



*p43.4.sop*  *jpeg.4753.54.sop*

* + Exponential exp(-ALPHA \* step)\*T0:



*p43.4.sop* *jpeg.4753.54.sop*

as plots show exponential method perform a little bit better search in compare with other.

* Simulated annealing Comparison with BKSs:

Instances run with bellow config:

T = 1

ALPHA = 0.9

TEMP\_MODE = EXP

INIT\_HEURISTIC = True

NUM\_ITERATIONS = 500

Instances with run time under 30 seconds, ran for 20 times and other ran for 10 times.

On the “E” instances folder, results were near to the BK answers except bellow instance types: kro124p.\*, prob.100, prob.7. \*, rbg109a.sop.

it seems that from view of this algorithm, these problems were harder than other.

On the “H” instances folder, results were almost similar to the BK answers (with maximum difference equal to 7).

the “M” instances were much more time consuming and the results weren’t as good as “H” folder.

The whole results (main, max, avg) came at the end.

* Simulated annealing results:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instance | BKS | best | average | worst | min\_time | avg\_time | max\_time | Diff of Best |
| ft53.2.sop | 8026 | 9473 | 10312.8 | 11041 | 1.1050 | 1.2013 | 1.3194 | 1447 |
| ft70.2.sop | 40419 | 44076 | 45597.4 | 46640 | 2.0604 | 2.3552 | 2.7556 | 3657 |
| kro124p.1.sop | 39420 | 46934 | 48537.15 | 49917 | 7.9588 | 8.9360 | 9.8073 | 7514 |
| kro124p.3.sop | 49499 | 59438 | 62762.4 | 66501 | 3.4378 | 4.8182 | 5.6962 | 9939 |
| p43.1.sop | 28140 | 28290 | 28598.75 | 28810 | 0.8178 | 0.9116 | 1.0894 | 150 |
| p43.4.sop | 83005 | 83140 | 83270.5 | 83445 | 0.3390 | 0.3805 | 0.4697 | 135 |
| prob.100.sop | 1123 | 3158 | 3870.05 | 4849 | 4.0428 | 4.2812 | 4.5359 | 2035 |
| prob.5.sop | 243 | 421 | 548.15 | 682 | 0.7549 | 0.8216 | 0.9136 | 178 |
| prob.7.40.sop | 1071 | 1788 | 2301.3 | 2981 | 0.6532 | 0.7296 | 0.8986 | 717 |
| prob.7.60.sop | 912 | 1952 | 2545.5 | 2968 | 1.4327 | 1.5482 | 1.6655 | 1040 |
| prob.7.70.sop | 879 | 2310 | 2881.25 | 3525 | 1.9058 | 2.1013 | 2.2778 | 1431 |
| rbg050a.sop | 400 | 407 | 439.4 | 478 | 0.8008 | 0.9093 | 1.0082 | 7 |
| rbg050b.sop | 397 | 403 | 432.75 | 463 | 0.8028 | 0.9041 | 1.1270 | 6 |
| rbg050c.sop | 467 | 468 | 480.9 | 494 | 0.7749 | 0.8580 | 0.9614 | 1 |
| rbg105a.sop | 1023 | 1064 | 1104.55 | 1143 | 1.9378 | 2.1143 | 2.4140 | 41 |
| rbg118a.sop | 1423 | 1424 | 1450.35 | 1507 | 1.8151 | 1.9227 | 2.1562 | 1 |
| rbg124a.sop | 1361 | 1366 | 1397.25 | 1436 | 1.8051 | 1.9065 | 2.0226 | 5 |
| rbg126a.sop | 1381 | 1398 | 1421.6 | 1481 | 1.9942 | 2.2430 | 2.6624 | 17 |
| rbg143a.sop | 1765 | 1774 | 1801.35 | 1832 | 2.1233 | 2.2379 | 2.4345 | 9 |
| rbg219a.sop | 2544 | 2578 | 2605.35 | 2632 | 6.6901 | 7.2614 | 7.8829 | 34 |
| rbg247a.sop | 3062 | 3101 | 3140.35 | 3187 | 8.1642 | 8.7001 | 9.8681 | 39 |
| rbg341a.sop | 2568 | 3117 | 3217.9 | 3342 | 26.782 | 29.577 | 33.403 | 549 |
| ry48p.2.sop | 16666 | 18290 | 20884.7 | 23105 | 0.9273 | 1.0249 | 1.1865 | 1624 |
| ry48p.3.sop | 19894 | 22029 | 23826.5 | 25251 | 0.7129 | 0.8665 | 1.0328 | 2135 |
| prob.7.65.sop | 915 | 1649 | 1930.65 | 2188 | 1.6960 | 2.2015 | 2.6070 | 734 |
| rbg109a.sop | 198 | 1046 | 1081.1 | 1110 | 2.0425 | 2.5092 | 2.7716 | 848 |
| rbg117a.sop | 1494 | 1497 | 1516.55 | 1548 | 1.4388 | 1.6960 | 1.8951 | 3 |
| rbg150a.sop | 1750 | 1783 | 1829.5 | 1866 | 3.7965 | 4.4883 | 5.2027 | 33 |
| rbg174a.sop | 2033 | 2059 | 2114.5 | 2146 | 5.1369 | 6.4195 | 7.2307 | 26 |
| rbg190a.sop | 2241 | 2269 | 2290.0 | 2311 | 5.2462 | 6.6285 | 7.6026 | 28 |
| rbg285a.sop | 3482 | 3557 | 3604.55 | 3668 | 14.519 | 15.807 | 18.102 | 75 |
| rbg358a.sop | 2545 | 2884 | 3001.15 | 3141 | 37.397 | 41.883 | 47.849 | 339 |
|  |  |  |  |  |  |  |  |  |
| gsm.153.124.sop | 1109 | 1110 | 1121.05 | 1129 | 0.6336 | 0.7397 | 0.9579 | 1 |
| gsm.462.77.sop | 577 | 578 | 581.45 | 587 | 0.5404 | 0.5812 | 0.6931 | 1 |
| jpeg.3184.107.sop | 791 | 798 | 808.0 | 817 | 0.8498 | 0.9889 | 1.1526 | 7 |
| jpeg.4753.54.sop | 245 | 247 | 256.5 | 269 | 0.3554 | 0.4670 | 0.7355 | 2 |
| susan.260.158.sop | 1016 | 1022 | 1035.65 | 1055 | 1.7578 | 2.1034 | 2.4285 | 6 |
| typeset.15577.36.sop | 155 | 155 | 160.65 | 171 | 0.2309 | 0.2598 | 0.3679 | 0 |
| typeset.1723.25.sop | 64 | 64 | 69.85 | 78 | 0.1578 | 0.1951 | 0.3143 | 0 |
| typeset.19972.246.sop | 2018 | 2018 | 2021.6 | 2034 | 1.3684 | 1.4857 | 1.7511 | 0 |
| typeset.4724.433.sop | 3466 | 3468 | 3478.2 | 3496 | 6.0954 | 6.8138 | 8.1372 | 2 |
| typeset.16000.68.sop | 84 | 84 | 85.2 | 90 | 0.6667 | 0.8752 | 1.1691 | 0 |
| typeset.10835.26.sop | 127 | 127 | 130.9 | 137 | 0.1950 | 0.2186 | 0.2806 | 0 |
|  |  |  |  |  |  |  |  |  |
| R.200.100.1.sop | 61 | 340 | 402.3 | 453 | 27.408 | 28.773 | 30.956 | 279 |
| R.200.100.60.sop | 71749 | 72804 | 74300.15 | 75808 | 1.8221 | 1.9594 | 2.2568 | 1055 |
| R.200.1000.30.sop | 41196 | 46190 | 49303.0 | 52981 | 2.2330 | 2.5983 | 3.2566 | 4994 |
| R.200.1000.60.sop | 71556 | 72846 | 74722.2 | 76561 | 1.9925 | 2.5362 | 2.8859 | 1290 |
| R.300.1000.60.sop | 109471 | 110993 | 112747.95 | 114203 | 5.2474 | 6.6597 | 9.1146 | 1522 |
| R.400.1000.15.sop | 38963 | 64354 | 66147.15 | 68407 | 21.304 | 22.863 | 25.351 | 25391 |
| R.500.1000.1.sop | 1316 | 3532 | 3733.14 | 3926 | 631.41 | 738.70 | 858.61 | 2216 |
| R.600.100.60.sop | 23293 | 24300 | 24479.8 | 24711 | 42.649 | 49.824 | 66.338 | 1007 |
| R.600.1000.1.sop | 1337 | 3676 | 3681.5 | 3687 | 1073.2 | 10774.5 | 1081.7 | 2339 |
| R.600.1000.60.sop | 214608 | 224197 | 226373.6 | 228394 | 29.580 | 33.008 | 39.903 | 9589 |
| R.700.1000.15.sop | 65678 | 121526 | 123669.0 | 126399 | 77.331 | 81.251 | 92.250 | 55848 |
| R.700.1000.60.sop | 245589 | 257974 | 259705.3 | 261393 | 80.584 | 92.586 | 100.08 | 12385 |

* Simulated annealing algorithm analysis:

**Strength:**

this algorithm is much faster that algorithm explained in the related origin paper cause instead of searching whole space with time complexity O(n3), perform the search just for ***“problem.dimension”*** times.

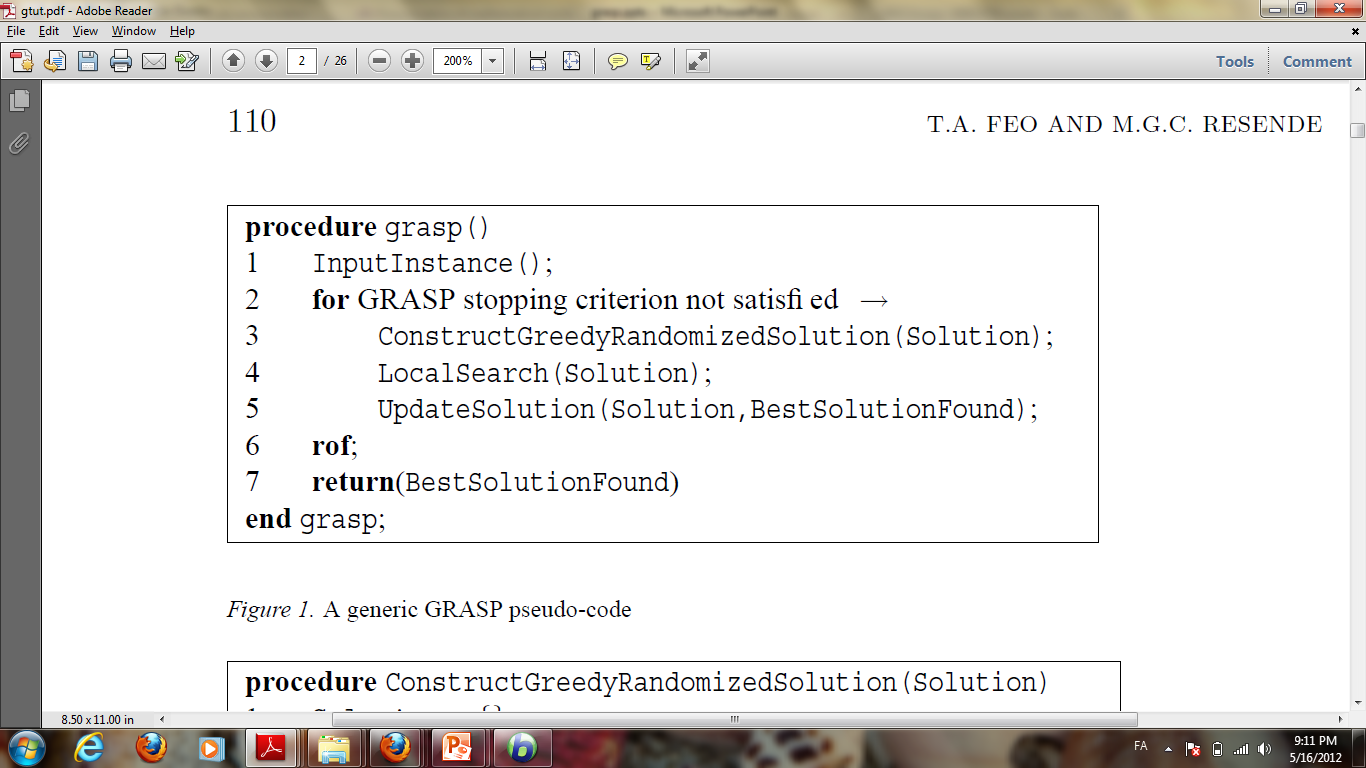
The results are really close to the paper method in most of the cases.

**Weakness:**

because of searching the less problem area that related paper method, in some instances it reaches a little bit worst result.

In overall the algorithm is a less time-consuming version of paper method with good acceptable results.

* GRASP algorithm:



As the pseudo-code shows the algorithm contains of 3 main part.

For SOP problems we implement the methods as bellow:

***“ConstructGreedyRandomizedSolution()”:***

Like what we did in initial heuristic of simulated annealing, step by step we choose next feasible greedy node to add to the path, but instead of using the best node from candidate list we choose from randomly between (0, ALPH) percentages best of candidate list for next node selection (rank-based selection) and by this manner we add random factor beside greedy factor as GRASP behaves.

index = *int*(rnd.uniform(0, ALPHA) \* graph.dimension)

dest = rnd.choice(*list*(candidates[0:1+index]))[0]

solution.append(dest)

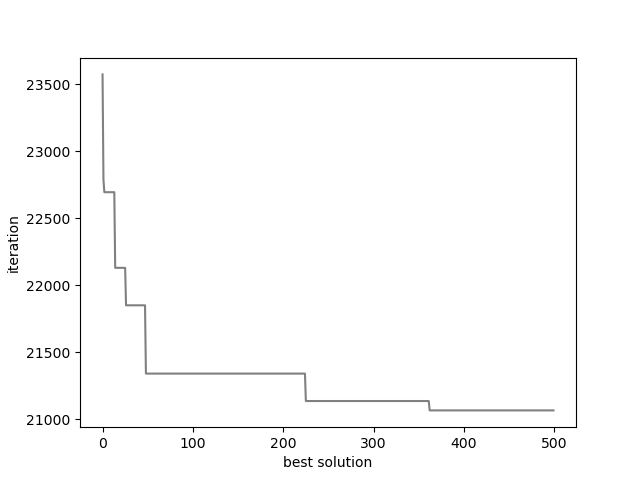
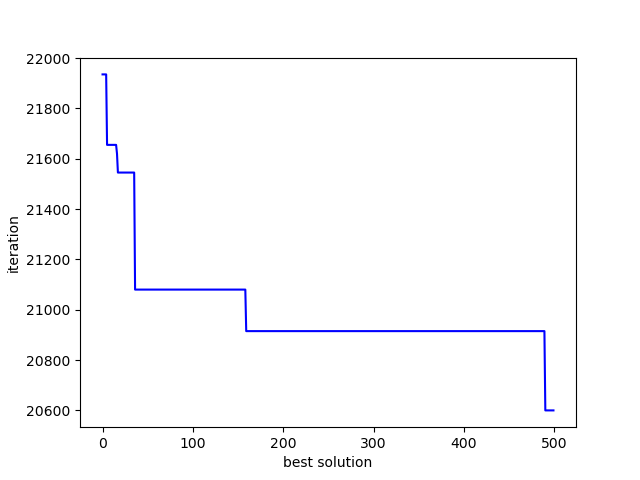
***“LocalSearch()”:***

For searching the local area just like we mentioned in ***“get\_neighbor()”*** previously ,we choose local optimal from best of backward and forward solutions.

***“UpdateSolution()”:***

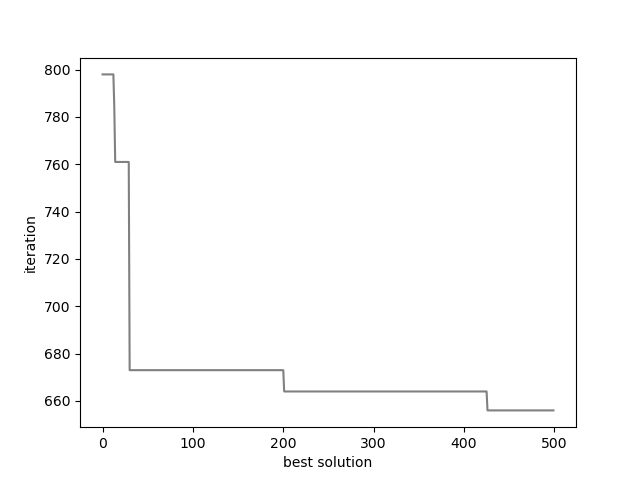
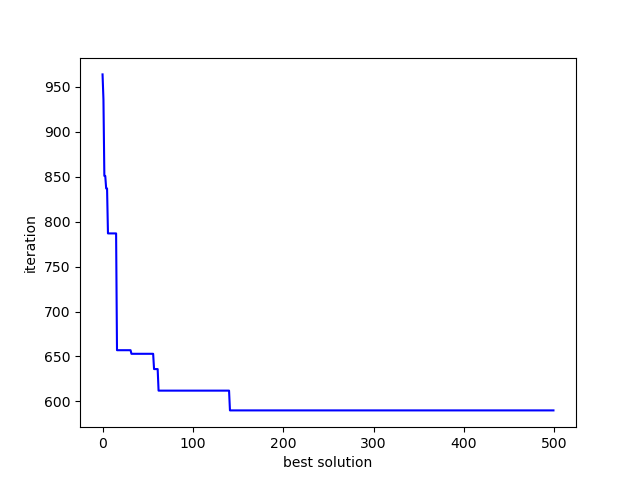
At the end we update global best solution if local optimal was better (smaller cost).

* Local search role analysis:
  + ESC78.sop

*Without local search With local search*

* + R.200.100.1

*Without local search With local search*

As plots shows local search operator cause finding better quality of solution at the end, it may or may not start from better solution. But the benefit of using it is obvious.

ALPHA = 0.02

NUM\_ITERATIONS = 500

* + **Without local search:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instance | best | worst | average | avg\_time |
| ESC78.sop | 20465 | 21370 | 21194.5 | 1.8862 |
| R.200.100.60.sop | 85057 | 87786 | 85892.3 | 24.435 |
| susan.260.158.sop | 1124 | 1143 | 1134.6 | 6.5894 |

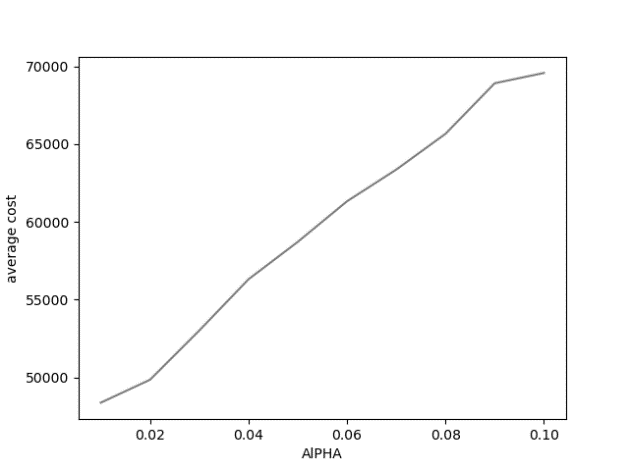
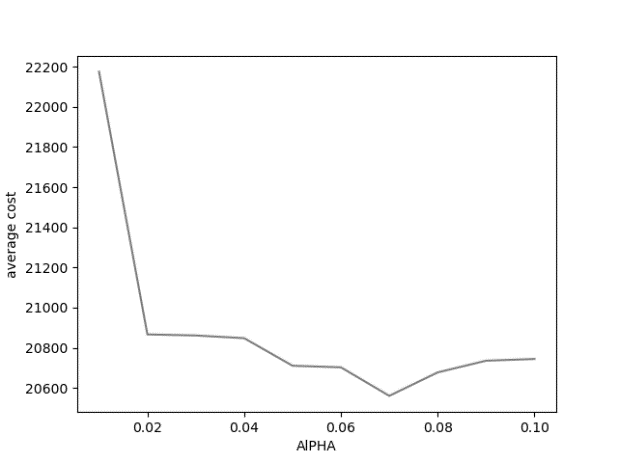
* + **With local search:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instance | best | worst | average | avg\_time |
| ESC78.sop | 20405 | 21220 | 20895.0 | 3.9388 |
| R.200.100.60.sop | 83444 | 87524 | 85955.7 | 26.316 |
| susan.260.158.sop | 1121 | 1139 | 1131.3 | 8.2227 |

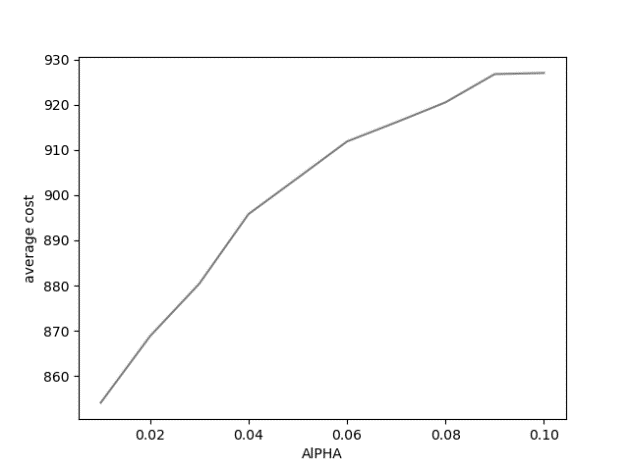
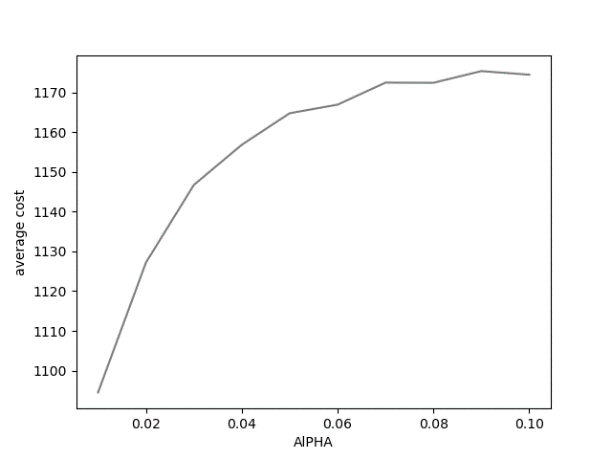
(Results per 10 time of run for each instance)

Local search operation gives better result overall and due to its searching process, it takes more time.

* ALPHA value (maximum size of RCL list) comparison:

Plot of average cost for 20 time run of each instance per AlPHA comes in bellow

*ESC78.sop kro124p.1.sop*



*sasan.260.158.sop jpeg.3184l.107.sop*

based on the result of some instances that plot of some of them came above, ALPHA=0.2 selected for all instances. In some cases, bigger alpha made better average solution.

It seems that bigger alpha cause bigger variance of solutions and due to bigger variance, it causes higher average.

Higher alpha value does more exploration and add more randomness on the other side lower alpha value does more exploitation.

* GRASP solutions variance analysis:
  + Solution variance of 10 times of running algorithm.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instance | best | average | worst | variance |
| ESC78.sop | 20400 | 20776.0 | 20970 | 200.88 |
| kro124p.3.sop | 62362 | 64364.6 | 66162 | 1332.38 |
| p43.4.sop | 84820 | 84820.0 | 84820 | 0.0 |
| prob.7.65.sop | 1689 | 1719.0 | 1774 | 30.78 |
| jpeg.3184.107.sop | 865 | 869.4 | 873 | 3.2 |
| susan.260.158.sop | 1116 | 1124.0 | 1132 | 5.32 |
| R.200.100.1.sop | 340 | 402.3 | 453 | 28.773 |
| R.600.100.60.sop | 28711 | 28765.5 | 28820 | 54.5 |

* + Solution variance of global optimal during running the algorithm.

|  |  |
| --- | --- |
| * Instance | variance |
| ESC78.sop | 301.89 |
| kro124p.3.sop | 1749.65 |
| p43.4.sop | 47.29 |
| prob.7.65.sop | 71.70 |
| jpeg.3184.107.sop | 8.47 |
| susan.260.158.sop | 13.26 |
| R.200.100.1.sop | 24.96 |

It seems that in some cases data variance is high and it’s because of randomness aspect of the algorithm.

* Simulated annealing vs GRASP comparison:
  + **Simulated annealing:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Instance | BKS | best | average | worst | avg\_time | Diff of Best |
| ESC78.sop | 18230 | 18250 | 18400.5 | 18535 | 1.7924 | 20 |
| kro124p.3.sop | 49499 | 59438 | 62762.4 | 66501 | 4.8182 | 9939 |
| prob.7.65.sop | 915 | 1649 | 1930.65 | 2188 | 2.2015 | 734 |
| susan.260.158.sop | 1016 | 1123 | 1128/8 | 1136 | 8.6370 | 6 |
| R.200.100.1.sop | 61 | 340 | 402.3 | 453 | 28.773 | 279 |
| R.400.1000.15.sop | 38963 | 64354 | 66147.15 | 68407 | 22.863 | 279 |

* + **GRASP:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Instance | BKS | best | average | worst | avg\_time | Diff of Best |
| ESC78.sop | 18230 | 20405 | 20895 | 21220 | 3.8190 | 2175 |
| kro124p.3.sop | 49499 | 60161 | 63202.9 | 67032 | 6.2631 | 10662 |
| prob.7.65.sop | 915 | 1560 | 1658/4 | 1812 | 3.2334 | 645 |
| susan.260.158.sop | 1016 | 1123 | 1128/8 | 1136 | 8.6370 | 107 |
| R.200.100.1.sop | 61 | 393 | 448.7 | 483 | 56.704 | 332 |
| R.400.1000.15.sop | 38963 | 144545 | 148052.9 | 150760 | 120.86 | 105582 |

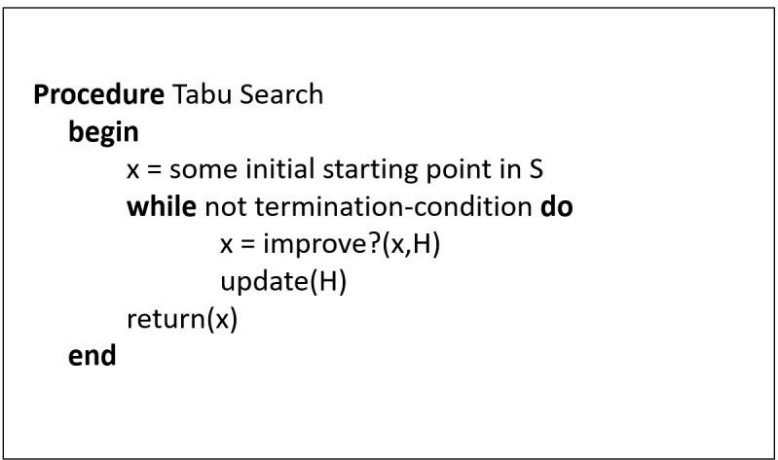
The GRASP algorithm average solution is worse than simulated annealing in compare and its variances of solution are bigger.

It also takes more time in average.in compare with pure greedy, GRASP perform better due to having the random aspect beside greedy.

* GRASP results:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instance | BKS | best | average | worst | min\_time | avg\_time | max\_time | Diff of Best |
| br17.1.sop | 41 | 61 | 61 | 61 | 0.2543 | 0.2572 | 0.2632 | 20 |
| br17.10.sop | 55 | 63 | 63 | 63 | 0.2124 | 0.2153 | 0.2170 | 8 |
| br17.12.sop | 55 | 63 | 63 | 63 | 0.2064 | 0.2315 | 0.2732 | 8 |
| ESC78.sop | 18230 | 20405 | 20895 | 21220 | 3.7371 | 3.8190 | 4.0162 | 2175 |
| ESC98.sop | 2125 | 2125 | 2125 | 2125 | 7.9125 | 8.2295 | 8.5833 | 0 |
| ft53.2.sop | 8026 | 10741 | 11249.2 | 11475 | 1.9532 | 2.1721 | 2.4002 | 2715 |
| ft70.2.sop | 40419 | 44566 | 45606.9 | 46400 | 3.3997 | 3.6471 | 3.8445 | 4147 |
| kro124p.1.sop | 39420 | 48588 | 49968.0 | 51023 | 9.2160 | 9.4826 | 9.6200 | 9168 |
| kro124p.3.sop | 49499 | 60161 | 63202.9 | 67032 | 5.9466 | 6.2631 | 6.6445 | 10662 |
| p43.1.sop | 28140 | 29120 | 29120 | 29120 | 1.3682 | 1.4517 | 1.5105 | 980 |
| p43.4.sop | 83005 | 84820 | 84820 | 84820 | 1.0476 | 1.1261 | 1.2562 | 1815 |
| prob.100.sop | 1123 | 2062 | 2248/6 | 2378 | 7.8009 | 8.0781 | 8.3549 | 939 |
| prob.5.sop | 243 | 417 | 417 | 417 | 1.2846 | 1.3691 | 1.5334 | 174 |
| prob.7.40.sop | 1071 | 2477 | 2477 | 2477 | 1.0850 | 1.1679 | 1.3081 | 1406 |
| prob.7.60.sop | 912 | 1530 | 1563 | 1623 | 2.6015 | 2.7256 | 2.9206 | 618 |
| prob.7.65.sop | 915 | 1560 | 1658/4 | 1812 | 2.9690 | 3.2334 | 3.5530 | 645 |
| prob.7.70.sop | 879 | 1608 | 1665/2 | 1762 | 3.4080 | 3.7098 | 4.1187 | 729 |
| rbg050a.sop | 400 | 474 | 475/2 | 476 | 1.7683 | 1.8415 | 1.9509 | 74 |
| rbg050b.sop | 397 | 501 | 507/8 | 517 | 1.6675 | 1.7497 | 1.9154 | 104 |
| rbg050c.sop | 467 | 543 | 546/8 | 549 | 1.6695 | 1.8106 | 1.9277 | 76 |
| rbg105a.sop | 1023 | 1300 | 1343/2 | 1362 | 6.0060 | 6.0823 | 6.1843 | 277 |
| rbg109a.sop | 198 | 1323 | 1336/2 | 1352 | 6.5961 | 6.8382 | 7.0040 | 1125 |
| rbg117a.sop | 1494 | 1654 | 1668/6 | 1681 | 6.6315 | 6.8512 | 7.0958 | 160 |
| rbg118a.sop | 1423 | 1641 | 1683 | 1699 | 6.7349 | 7.0234 | 7.1377 | 218 |
| rbg124a.sop | 1361 | 1623 | 1635/4 | 1645 | 7.8546 | 8.1561 | 8.5900 | 262 |
| rbg126a.sop | 1381 | 1678 | 1709/2 | 1735 | 8.3840 | 8.5852 | 8.8323 | 297 |
| rbg143a.sop | 1765 | 2077 | 2091/6 | 2111 | 12.001 | 10.228 | 9.8187 | 312 |
| rbg150a.sop | 1750 | 2101 | 2128/4 | 2144 | 13.088 | 13.528 | 14.159 | 351 |
| rbg174a.sop | 2033 | 2461 | 2488/8 | 2501 | 18.461 | 19.102 | 19.628 | 428 |
| rbg190a.sop | 2241 | 2826 | 2857/2 | 2904 | 21.700 | 21.946 | 22.479 | 585 |
| rbg219a.sop | 2544 | 3301 | 3353/8 | 3396 | 30.687 | 31.354 | 31.793 | 757 |
| rbg247a.sop | 3062 | 3930 | 3992/8 | 4043 | 38.060 | 38.995 | 39.407 | 868 |
| rbg285a.sop | 3482 | 4576 | 4617/2 | 4674 | 54.148 | 55.410 | 56.633 | 1094 |
| rbg341a.sop | 2568 | 4583 | 4643/2 | 4716 | 100.00 | 99.833 | 99.956 | 2015 |
| rbg358a.sop | 2545 | 4746 | 4802/6 | 4859 | 118.75 | 120.36 | 123.48 | 2201 |
| ry48p.2.sop | 16666 | 20407 | 20407 | 20407 | 1.5763 | 1.6015 | 1.6224 | 3741 |
| ry48p.3.sop | 19894 | 26118 | 26118 | 26118 | 1.2955 | 1.3214 | 1.3502 | 6224 |
|  |  |  |  |  |  |  |  |  |
| gsm.153.124.sop | 1109 | 1177 | 1185 | 1194 | 4.7513 | 4.8959 | 5.1867 | 68 |
| gsm.462.77.sop | 577 | 585 | 586/4 | 587 | 2.1583 | 2.2547 | 2.4721 | 8 |
| jpeg.3184.107.sop | 791 | 856 | 865/2 | 875 | 3.9890 | 4.0605 | 4.1194 | 65 |
| jpeg.4753.54.sop | 245 | 257 | 259 | 261 | 1.1449 | 1.2211 | 1.3740 | 12 |
| susan.260.158.sop | 1016 | 1123 | 1128/8 | 1136 | 8.5176 | 8.6370 | 8.7726 | 107 |
| typeset.10835.26.sop | 127 | 137 | 137 | 137 | 0.4084 | 0.4323 | 0.4775 | 10 |
| typeset.15577.36.sop | 155 | 175 | 175 | 175 | 0.6161 | 0.6451 | 0.6712 | 20 |
| typeset.16000.68.sop | 84 | 85 | 86/2 | 87 | 1.6542 | 1.7778 | 1.8768 | 1 |
| typeset.1723.25.sop | 64 | 72 | 72 | 72 | 0.3459 | 0.3598 | 0.3951 | 8 |
| typeset.19972.246.sop | 2018 | 2068 | 2073/6 | 2080 | 23.296 | 23.464 | 23.676 | 50 |
| typeset.4724.433.sop | 3466 | 3657 | 3668/8 | 3679 | 107.35 | 107.63 | 107.94 | 191 |
|  |  |  |  |  |  |  |  |  |
| R.200.100.1.sop | 61 | 393 | 448.7 | 483 | 56.464 | 56.704 | 56.955 | 332 |
| R.200.100.60.sop | 71749 | 84814 | 85629/8 | 86253 | 24.674 | 25.005 | 25.163 | 13065 |
| R.200.1000.30.sop | 41196 | 71241 | 73088/8 | 74544 | 19.661 | 19.811 | 20.100 | 30045 |
| R.200.1000.60.sop | 71556 | 83161 | 86006 | 87744 | 24.430 | 24.736 | 24.873 | 11605 |
| R.300.1000.60.sop | 109471 | 134691 | 135201/8 | 136134 | 72.406 | 72.763 | 73.572 | 25220 |
| R.400.1000.15.sop | 38963 | 144545 | 148052.9 | 150760 | 120.11 | 120.86 | 121.68 | 105582 |
| R.500.1000.1.sop | 1316 | 11371 | 12200.9 | 12815 | 1025.4 | 1084.3 | 998.53 |  |
| R.600.100.60.sop | 23293 | 28711 | 28765.5 | 28820 | 522.35 | 543.22 | 564.09 |  |

* Tabu search algorithm:



***“Initial starting point( )”:***

just like simulated annealing approach heuristic initial manner use to create initial solution.

***“Improve(x,H)”:***

Create candidate solution list consist of n/2 backward, n/2 forward exchange result and.

Then sort the candidate list and choose first best feasible solution based on actions in Tabu list and then add new action to Tabu list.

***“update(H)”:***

Decreasing the actions lifetime and delete if expiration time comes.

* Tabu list size comparison:

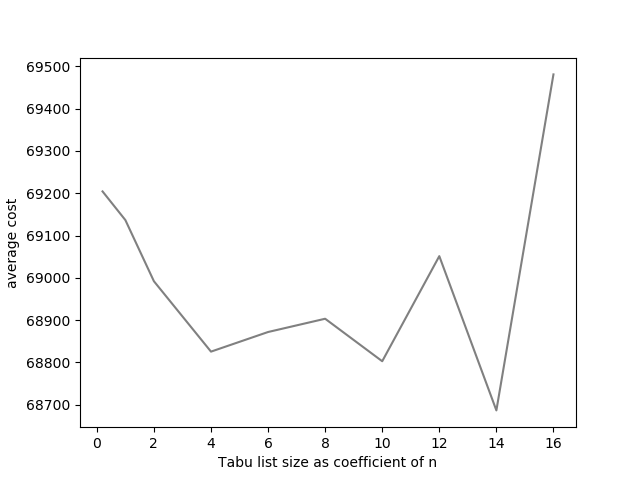
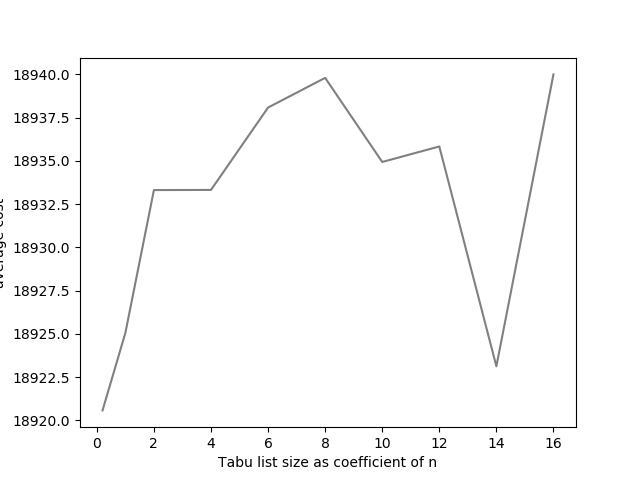
NUM\_ITERATIONS = 500

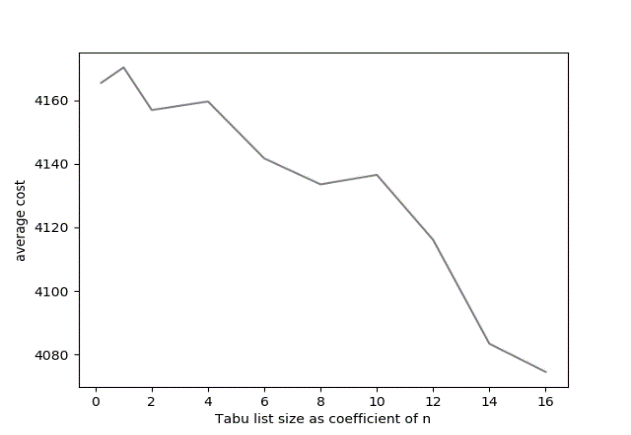
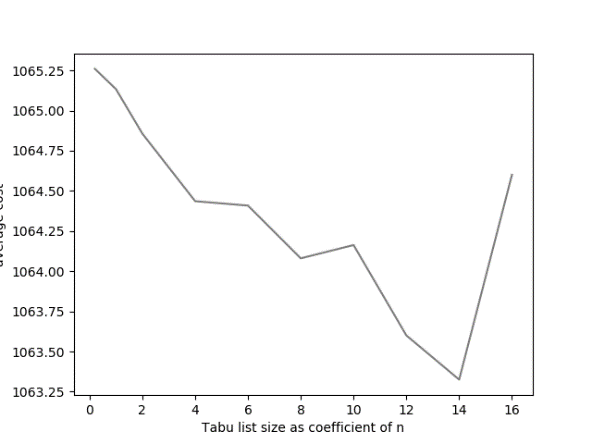
TABU\_LIST\_MODE = SHORT\_TERM

INIT\_HEURISTIC = True

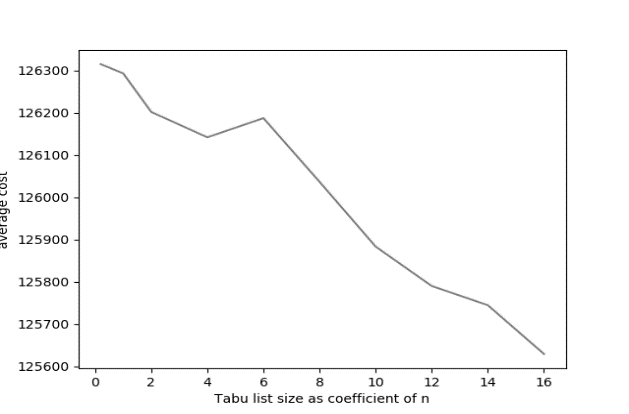
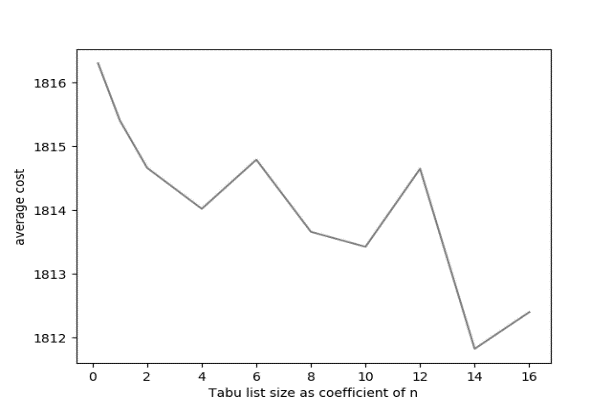
ACTIONS\_EXPIRATION\_DATE = 4

20 times run per each tabu list size configuration



*ESC78.sop kro124p.3.*

*prob.7.65.sop susan.260.158.sop*

*R.300.1000.60.sop rbg150a.sop*

As plot shows the best coefficient in the range was 14 and Tabu list size would be:

TABU\_LIST\_SIZE = *int*(problem.dimension/14)

* Tabu list elements expiration time(lifetime) comparison:

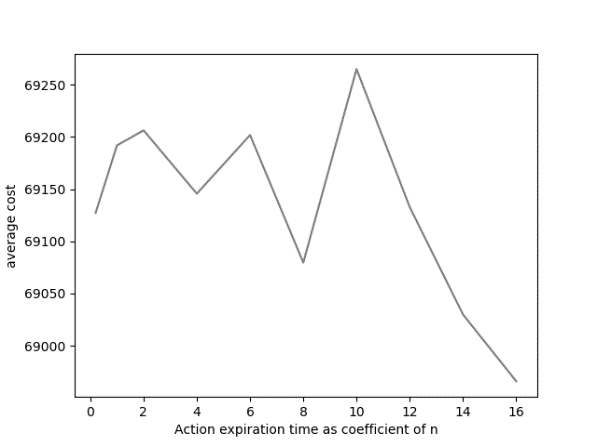
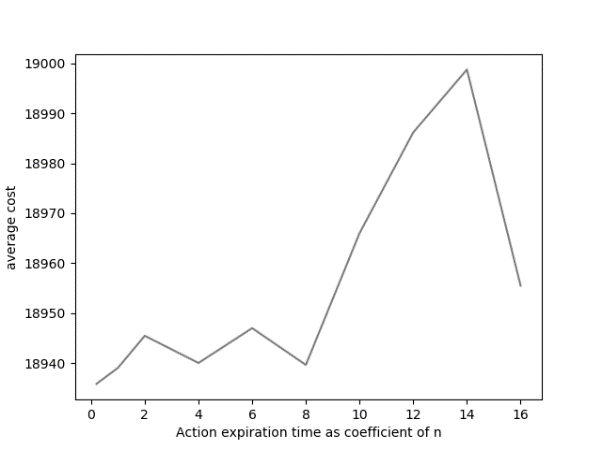
NUM\_ITERATIONS = 500

TABU\_LIST\_MODE = SHORT\_TERM

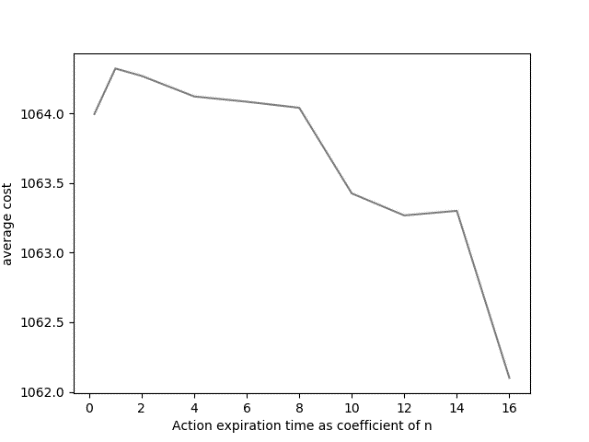
INIT\_HEURISTIC = True

TABU\_LIST\_SIZE = 14

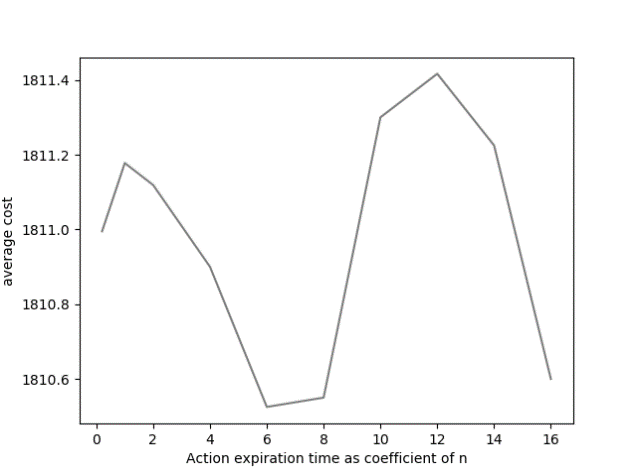
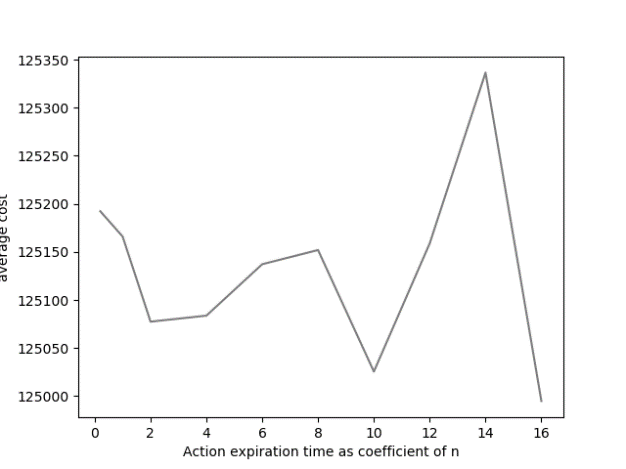
20 times run per each element expiration time configuration



*ESC78.sop kro124p.3.sop*



*prob.7.65.sop susan.260.158.sop*



*R.300.1000.60.sop\_exp rbg150a.sop*

So we keep actions for time as: MAX\_MEM\_DEPTH = *int*(problem.dimension/16)

* Variance comparison with GRASP:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instance | GRASP average | Tabu  average | GRASP variance | Tabu  variance |
| kro124p.3.sop | 64364.6 | 67817.5 | 1332.38 | 2589.92 |
| p43.4.sop | 84820.0 | 83589.0 | 0.0 | 345.722 |
| prob.7.65.sop | 1719.0 | 4191.6 | 30.78 | 255.51 |
| jpeg.3184.107.sop | 869.4 | 832.7 | 3.2 | 8.379 |
| susan.260.158.sop | 1124.0 | 1061.7 | 5.32 | 11.3846 |
| R.200.100.1.sop | 402.3 | 595.5 | 28.773 | 50.66606 |

Tabu search has nearly the same average solution in compare with GRASP, but it always produces solutions with higher variances.it seems that selecting solution based on action feasibility causes more exploration and higher variance.

* solution comparison with simulated annealing:
  + **Simulated annealing:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Instance | BKS | best | average | worst | variance | avg\_time | Diff of Best |
| kro124p.3.sop | 49499 | 60915 | 63961.7 | 67737 | 2267.118 | 4.8182 | 11416 |
| prob.7.65.sop | 915 | 1704 | 1888.8 | 2124 | 138.627 | 1.7221 | 734 |
| ry48p.3.sop | 19894 | 22224 | 23397.8 | 24215 | 614.056 | 0.7341 | 2330 |
| susan.260.158.sop | 1016 | 1026 | 1036.4 | 1056 | 8.237 | 1.4402 | 10 |
| R.200.100.1.sop | 61 | 152 | 176.4 | 183 | 8.754 | 28.593 | 91 |
| R.400.1000.15.sop | 38963 | 63673 | 65818.5 | 68069 | 1457.441 | 25.197 | 24710 |

* + **Tabu search:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Instance | BKS | best | average | worst | variance | avg\_time | Diff of Best |
| kro124p.3.sop | 49499 | 65288 | 67817.5 | 73824 | 2589.92 | 3.5098 | 15789 |
| prob.7.65.sop | 915 | 3665 | 4191.6 | 4615 | 255.51 | 2.2015 | 2750 |
| ry48p.3.sop | 19894 | 23210 | 24488 | 25766 | 0.6731 | 0.7420 | 3316 |
| susan.260.158.sop | 1016 | 1043 | 1061.7 | 1078 | 11.3846 | 1.4507 | 27 |
| R.200.100.1.sop | 61 | 489 | 595.5 | 651 | 50.66606 | 28.366 | 428 |
| R.400.1000.15.sop | 38963 | 71080 | 72614.8 | 75225 | 1410.801 | 22.686 | 32117 |

It’s obvious that simulated annealing it better by aspects of solution quality and variance.

Maybe it’s because of that simulated annealing has good exploitation due to decreasing temperature over time, but Tabu search has higher exploration manner.

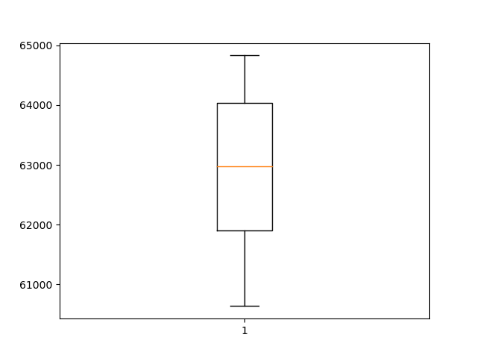
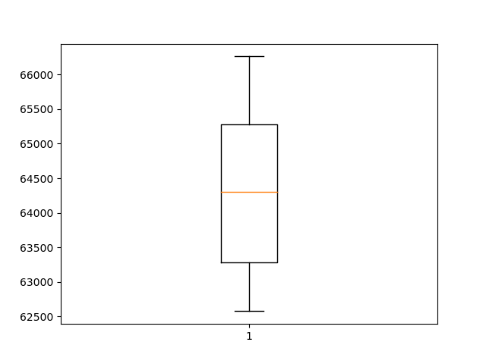
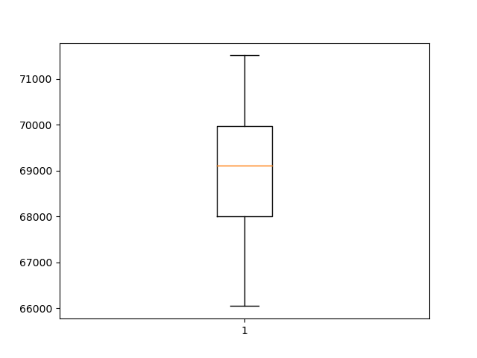
Maybe because of setting the greedy initial solutions for simulated annealing algorithm, its performance improves really better from its original version.

* Borda count voting based on average solutions:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instance | kro124p.3 | prob.7.65 | ry48p.3 | susan.260.158 | rbg109a | R.200.100.1 | R.400.1000.15 | Total score |
| Simulated annealing | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 8 |
| GRASP | 2 | 1 | 3 | 3 | 3 | 2 | 3 | 17 |
| Tabu search | 3 | 3 | 2 | 2 | 2 | 3 | 2 | 17 |

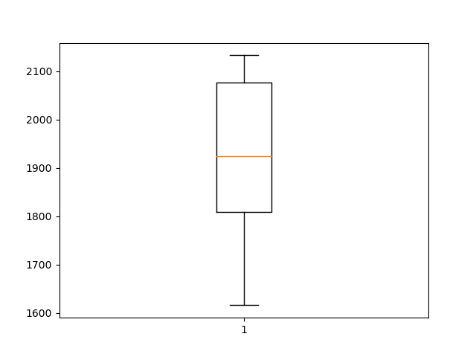
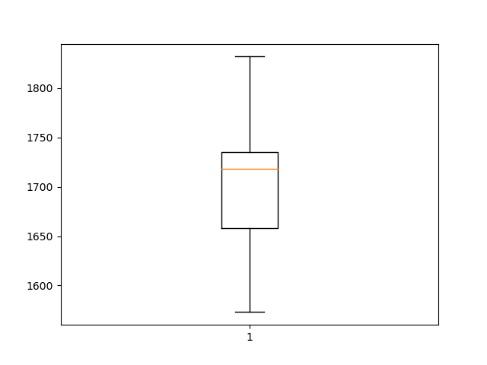
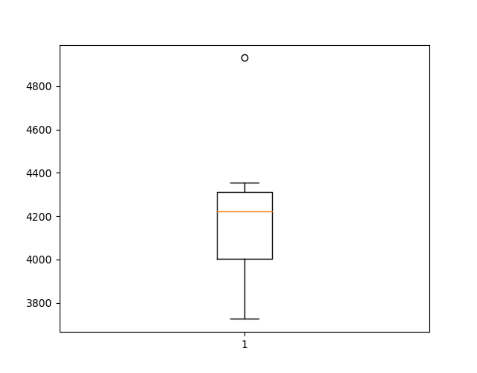
As the comparison shows the rank of GRASP and Tabu search is really close for selected instances and it seems that they both have same degree of exploration and exploitation but simulated annealing seems to have better exploitations and this factor cause its good performance.

* Algorithms box plot (for 10 time running) comparing:
  + kro124p.3

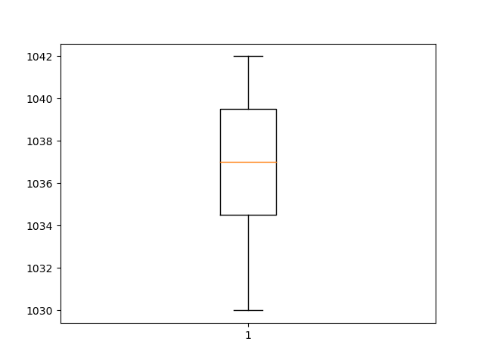
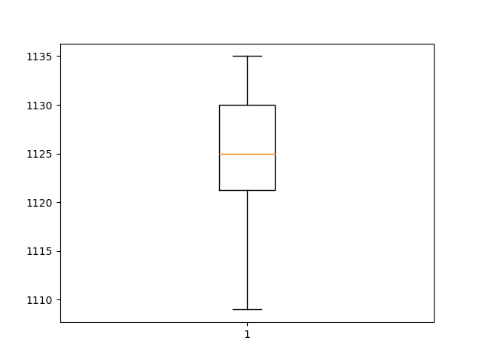
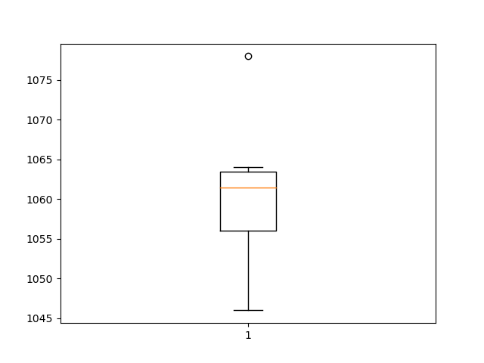
*Simulated annealing GRASP Tabu search*

* + prob.7.65

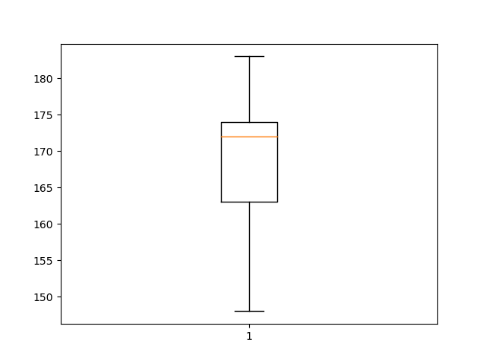
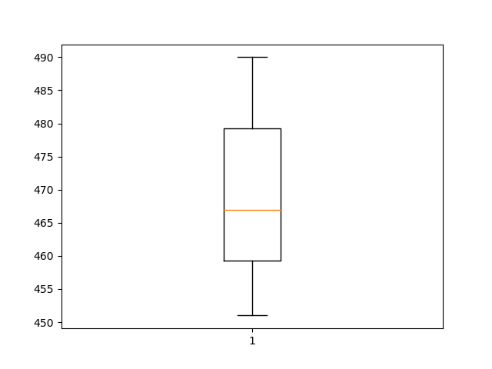
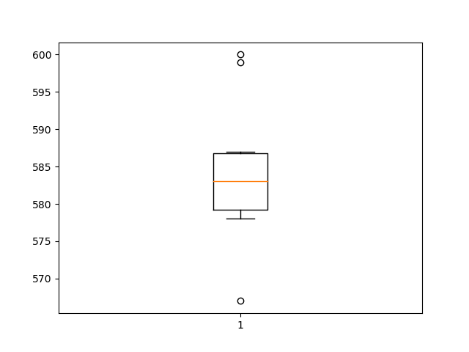
*Simulated annealing GRASP Tabu search*

* + susan.260.158

*Simulated annealing GRASP Tabu search*

* + R.200.100.1

*Simulated annealing GRASP Tabu search*

* Tabu search results:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instance | BKS | best | average | worst | min\_time | avg\_time | max\_time | Diff of Best |
| br17.1.sop | 41 | 41 | 46 | 54 | 4/171331 | 0.1406 | 0.1570 | 0 |
| br17.10.sop | 55 | 57 | 63/2 | 81 | 6/415606 | 0.1077 | 0.1154 | 2 |
| br17.12.sop | 55 | 56 | 65/6 | 80 | 6/421838 | 0.1037 | 0.1182 | 1 |
| ESC78.sop | 18230 | 18515 | 18967.5 | 19325 | 216.301 | 1.8291 | 1.9568 | 285 |
| ESC98.sop | 2125 | 2125 | 2125 | 2125 | 0 | 4.3921 | 4.6999 | 0 |
| ft53.2.sop | 8026 | 9365 | 10362.5 | 11655 | 599.48 | 1.1829 | 1.2211 | 1339 |
| ft70.2.sop | 40419 | 45451 | 46936.3 | 48118 | 886.274 | 2.0437 | 2.2631 | 5032 |
| kro124p.1.sop | 39420 | 52980 | 59270/2 | 63593 | 3253/095 | 5.5966 | 5.8464 | 13560 |
| kro124p.3.sop | 49499 | 65288 | 67817.5 | 73824 | 2589.92 | 3.0647 | 3.5098 | 15789 |
| p43.1.sop | 28140 | 28535 | 28839.5 | 29055 | 171.66 | 0.8965 | 0.9395 | 395 |
| p43.4.sop | 83005 | 83105 | 83589.0 | 84230 | 345.722 | 0.3610 | 0.3914 | 100 |
| prob.100.sop | 1123 | 4667 | 5355/8 | 6509 | 476.7728 | 4.4701 | 4.6030 | 3544 |
| prob.5.sop | 243 | 488 | 642/4 | 782 | 74.0907 | 0.8123 | 0.8636 | 245 |
| prob.7.40.sop | 1071 | 2376 | 3096/7 | 3511 | 348/2941 | 0.6883 | 0.7319 | 1305 |
| prob.7.60.sop | 912 | 3033 | 3547/7 | 3861 | 308/1435 | 1.5020 | 1.5881 | 2121 |
| prob.7.65.sop | 915 | 3665 | 4191.6 | 4615 | 255.51 | 2.0096 | 2.2015 | 2750 |
| prob.7.70.sop | 879 | 3113 | 4147/7 | 4805 | 518/5316 | 2.0012 | 2.1218 | 2234 |
| rbg050a.sop | 400 | 443 | 465/6 | 489 | 14/96128 | 0.8771 | 0.9455 | 43 |
| rbg050b.sop | 397 | 423 | 438/4 | 451 | 9/057593 | 0.8448 | 0.9022 | 26 |
| rbg050c.sop | 467 | 492 | 508 | 522 | 8/330666 | 0.8440 | 0.8855 | 25 |
| rbg105a.sop | 1023 | 1099 | 1140/4 | 1187 | 26/07374 | 1.7933 | 1.9109 | 76 |
| rbg109a.sop | 198 | 1085 | 1144/9 | 1186 | 25/56736 | 1.7671 | 1.8380 | 887 |
| rbg117a.sop | 1494 | 1552 | 1591/6 | 1644 | 31/985 | 1.4164 | 1.4568 | 58 |
| rbg118a.sop | 1423 | 1500 | 1525/4 | 1568 | 20/8 | 1.7533 | 1.7835 | 77 |
| rbg124a.sop | 1361 | 1440 | 1486/8 | 1542 | 34/87922 | 1.6948 | 1.7380 | 79 |
| rbg126a.sop | 1381 | 1493 | 1517/9 | 1578 | 25/0697 | 1.9843 | 2.0397 | 112 |
| rbg143a.sop | 1765 | 1851 | 1895/8 | 1964 | 33/90221 | 2.0928 | 2.1434 | 86 |
| rbg150a.sop | 1750 | 1797 | 1814/5 | 1834 | 11/43897 | 3.5238 | 3.6091 | 47 |
| rbg174a.sop | 2033 | 2090 | 2109/2 | 2128 | 12/05653 | 4.7181 | 4.8193 | 57 |
| rbg190a.sop | 2241 | 2345 | 2391/4 | 2424 | 23/76215 | 4.9893 | 5.1165 | 104 |
| rbg219a.sop | 2544 | 2686 | 2736/5 | 2808 | 39/28931 | 6.6339 | 6.7116 | 142 |
| rbg247a.sop | 3062 | 3196 | 3267.3 | 3304 | 28.88 | 7.8361 | 7.9842 | 134 |
| rbg285a.sop | 3482 | 3702 | 3754/6 | 3825 | 38/01105 | 11.479 | 11.686 | 220 |
| rbg341a.sop | 2568 | 2974 | 3080.1 | 3140 | 45.058 | 28.002 | 30.609 | 406 |
| rbg358a.sop | 2545 | 3015 | 3050/5 | 3121 | 36/65038 | 33.822 | 34.290 | 470 |
| ry48p.2.sop | 16666 | 19834 | 21305.2 | 22618 | 873.76 | 0.9015 | 1.0019 | 3168 |
| ry48p.3.sop | 19894 | 23210 | 24488 | 25766 | 0.6731 | 0.7106 | 0.7420 | 3316 |
| gsm.153.124.sop | 1109 | 1132 | 1150/6 | 1169 | 11/08332 | 0.7227 | 0.7637 | 23 |
| gsm.462.77.sop | 577 | 586 | 593/7 | 603 | 4/754997 | 0.5922 | 0.6224 | 9 |
| jpeg.3184.107.sop | 791 | 815 | 832.7 | 846 | 8.379 | 0.8592 | 0.8846 | 24 |
| jpeg.4753.54.sop | 245 | 259 | 267/4 | 277 | 4/882622 | 0.3730 | 0.4134 | 14 |
| susan.260.158.sop | 1016 | 1043 | 1061.7 | 1078 | 11.3846 | 1.3065 | 1.4507 | 27 |
| typeset.10835.26.sop | 127 | 133 | 139/3 | 151 | 5/814637 | 0.1901 | 0.2266 | 6 |
| typeset.15577.36.sop | 155 | 167 | 172/6 | 183 | 4/340507 | 0.2369 | 0.2657 | 12 |
| typeset.16000.68.sop | 84 | 84 | 92.8 | 105 | 6.4156 | 0.5754 | 0.6166 | 0 |
| typeset.1723.25.sop | 64 | 70 | 76/6 | 83 | 3/746999 | 0.1627 | 0.1886 | 6 |
| typeset.19972.246.sop | 2018 | 2054 | 2078/4 | 2100 | 14/988 | 1.4695 | 1.5317 | 36 |
| typeset.4724.433.sop | 3466 | 3610 | 3653/2 | 3704 | 26/65633 | 5.2469 | 5.3356 | 144 |
| R.200.100.1.sop | 61 | 489 | 595/5 | 651 | 50/66606 | 27.902 | 28.366 | 428 |
| R.200.100.60.sop | 71749 | 82739 | 84762/7 | 86314 | 1122/801 | 1.6907 | 743.63 | 10990 |
| R.200.1000.30.sop | 41196 | 52713 | 55522/2 | 59421 | 1819/576 | 2.3562 | 2.8049 | 11517 |
| R.200.1000.60.sop | 71556 | 82422 | 85619/2 | 88806 | 1801/524 | 1.7393 | 1.8218 | 10866 |
| R.300.1000.60.sop | 109471 | 122242 | 125247/3 | 128214 | 1762/341 | 4.5144 | 4.8479 | 12771 |
| R.400.1000.15.sop | 38963 | 71080 | 72614/8 | 75225 | 1410/801 | 21.660 | 22.686 | 32117 |
| R.500.1000.1.sop | 1316 | 9019 | 9460/1 | 9822 | 232/0528 | 636.67 | 660.26 | 7703 |
| R.600.100.60.sop | 23293 | 26083 | 26457.8 | 26764 | 247/129 | 27.174 | 27.893 | 2790 |

CCVRP optimization with Genetic and ACS

* Problem Description:

The Clustered capacitated vehicle routing problem (CCVRP) consist of n-1 costumers with certain need and one depot with some vehicles with specific amount of capacity.

Each customer vi (i ∈ {1,…,n}) has a known nonnegative demand di to be delivered or collected and the depot has a fictitious demand d0 = 0. There exist m identical vehicles, each with a capacity Q and in order to ensure feasibility we assume that di ⩽ Q for each i ∈ {1,…,n}.

Problem assumption:

* + each route starts and ends at the depot vertex;
  + once a vehicle enters a cluster, it visits all the vertices within the cluster before leaving it;
  + the sum of the demands of the visited vertices by a route does not exceed the capacity of the vehicle, Q.
* Instances Description:

Instances are created based on CVRP instances form TSPLIB library with difference that we created new problem that is a clustered version of CVRP.

Each CVRP instance file consists of two part as **specification part** that contains information about the instance data and **data part**.

* Algorithm Description:

The algorithm designed base on the related paper as *(A novel two-level optimization approach for clustered vehicle routing problem).*

Our approach is obtained by decomposing the problem into two logical and natural subproblems:

an upper-level (global) subproblem and a lower-level (local) subproblem. The first subproblem aims at determining the routes visiting the clusters, called global routes, using a genetic algorithm applied

to the corresponding global graph (see details in Section 3) while the aim of the second subproblem is to determine the visiting order within the clusters for the above-mentioned routes. The second subproblem is solved by transforming each global route into a TSP which then is

computed optimally using the Concorde TSP solver.

* Global subproblem with GA:

As mentioned, we solve this subproblem using GA, so we describe out GA as follow parts:

* + Representation:

Global subproblem search space is consist of number of clusters Vi (i ∈ {0…,n}) witch depot node placed in V0 cluster .

So out representation would be a sequence of cluster numbers that shows out global routes toward clusters. Depot cluster would be seen repeatedly as finish each route from depot and back to it.

As images shows one feasible solution can be: (7 1 2 0 3 4 0 5 6)

For creating the chromosome, we list cluster needs in descending and trying to satisfy needs by minimum vehicle number.

* + Fitness function:

The fitness function of each individual chromosome in the population is given by the total length of the best corresponding clustered routes associated to the collection of global routes specified by the chromosome. This distance also takes into account the order in which the vertices within the clusters are visited. Our aim is to minimize this total distance.

* + CrossOver:

Our GA uses a custom version of the two-point crossover. The crossover function takes two parent candidate solutions as input and outputs two solutions.

Cross over acts like two-point crossover but it allows repetition of depot cluster (because at the end of each route we came back to depot) as number of it was repeated in the parent chromosome.

P1 = (6 8 0 | 1 3 7 | 0 5 4 2)

P2 = (7 2 1 | 6 0 4 | 3 0 5 8)

O1 = (2 6 0 1 3 7 4 0 5 8)

O2 = (8 0 1 3 6 0 4 7 5 2)

* + Mutation:

we use a swapping inter-cluster mutation operator which acts as follows: we randomly select genes (i.e. clusters) and if the genes are from different global routes, their position is exchanged.

(5 8 1 0 3 7 0 6 4 2) -> (5 6 1 0 3 7 0 8 4 2)

* + Parent Selection:

Two parents with same chromosome size randomly selected from 30% of best population.

* + Survivor Selection:

Elitism manner used in the way that only best individual of every generation moving into the next generation.

* Local subproblem:

The basic idea used in our transformation is to add an artificial cost M to all the inter-cluster edges in this way forcing the vehicle to visit all the vertices within the cluster before leaving it.

1. The set of nodes of Gp and Gp′ are the same.

2. The entries of the cost matrix Gp′ are defined as follows:

(a) if vi,vj ∈ Vk then c′ (vi,vj) = c (vi,vj);

(b) if vi ∈ Vk,vj ∈ Vl with k ≠ l then c′ (vi,vj) = c (vi,vj) + M;

where M > Σ ∈ c (vi,vj).

After that we passing the edge matrix to TSP solver with bellow configuration.

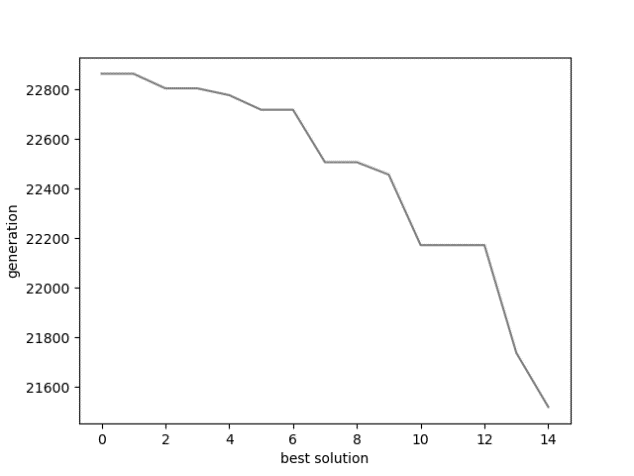
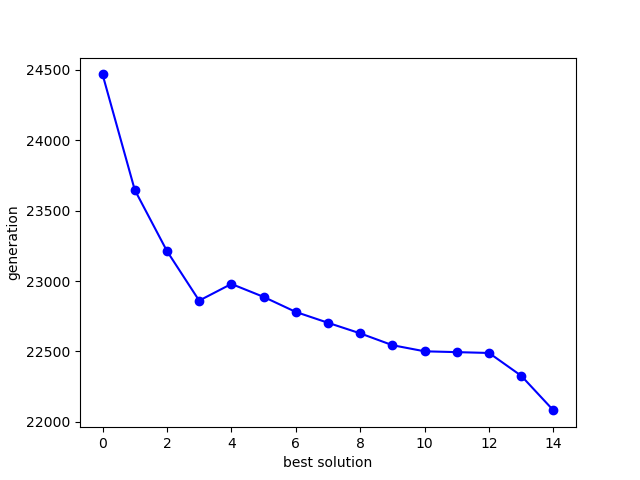
For solving local subproblem open source TSP solver used as bellow.

solver.read\_mat(mat)

config = TwoOpt\_solver(*initial\_tour*='NN', *iter\_num*=100)

            answer = solver.get\_approx\_solution(config )

* GA learning process:

Best solution per generation average solution per generation

* GA Results:

Because of HW limitation of execution time (1 minute per instance) bellow

configuration selected.

MUTATION\_RATE = 0.2

POPULATION\_SIZE = 40

MAX\_GENERATION = 10

XOVER\_METHOD = ORDER\_2POINT

SELECTION = RANDOM

SURVIVOR\_SEL\_TYPE = ELITISM (best will be kept)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ρ = 10% |  |  |  |  |  |  |  |  |  |  |  |
| index | **clusters** | **vehicles** | **vertices** | **Q** | **BKS** | **Best** | **average** | **worst** | **variance** | **avg\_time** | **avg vehicles** |
| 1 | 120 | 9 | 241 | 550 | 5759/25 | 16532.41 | 17153.33 | 17830.24 | 406.655 | 22.984 | 9 |
| 2 | 101 | 10 | 321 | 700 | 9247/92 | 22933.83 | 23503.14 | 24101.87 | 287.613 | 32.732 | 10 |
| 3 | 96 | 10 | 401 | 900 | 12904/6 | 29986.90 | 30851.00 | 31644.01 | 524.867 | 73.037 | 10 |
| 4 | 104 | 10 | 481 | 1000 | 17810/4 | 41385.85 | 42151.04 | 43103.16 | 503.638 | 115.14 | 10 |
| 5 | 49 | 5 | 201 | 900 | 8960/31 | 16526.34 | 17232.05 | 17702.95 | 334.257 | 56.114 | 5 |
| 6 | 67 | 7 | 281 | 900 | 10976/5 | 22204.49 | 22861.75 | 23504.54 | 372.046 | 64.244 | 7 |
| 7 | 88 | 9 | 361 | 900 | 12485/8 | 28235.49 | 28795.24 | 29269.60 | 378.717 | 71.651 | 9 |
| 8 | 108 | 11 | 441 | 900 | 13331/2 | 33223.84 | 34325.43 | 35153.69 | 519.279 | 84.987 | 11 |
| 9 | 51 | 15 | 256 | 1000 | 710/64 | 974.90 | 1022.87 | 1045.15 | 19.996 | 32.074 | 16.0 |
| 10 | 56 | 18 | 324 | 1000 | 908/89 | 1243.49 | 1267.09 | 1300.11 | 19.571 | 45.193 | 18.0 |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
| ρ = 25% |  |  |  |  |  |  |  |  |  |  |  |
| index | **clusters** | **Vehicles** | **vertices** | **Q** | **BKS** | **best** | **average** | **Worst** | **variance** | **avg\_time** | **avg vehicles** |
| 1 | 40 | 10 | 241 | 550 | 6051/04 | 9188.307 | 9592.559 | 9899.38 | 166.623 | 52.46 | 10 |
| 3 | 38 | 10 | 401 | 900 | 13692/6 | 17823.20 | 18334.96 | 18584.4 | 210.22 | 50.949 | 10 |
| 5 | 19 | 5 | 201 | 900 | 9340/7 | 11125.28 | 11383.95 | 11579.1 | 140.223 | 41.615 | 5 |
| 7 | 34 | 9 | 361 | 900 | 12348/1 | 16028.19 | 16513.08 | 17077.0 | 301.396 | 45.701 | 9 |
| 9 | 51 | 16 | 256 | 1000 | 717/63 | 1032.50 | 1080.82 | 1128.57 | 31.236 | 34.746 | 16 |
| 11 | 63 | 20 | 400 | 1000 | 1131/84 | 1621.83 | 1688.73 | 1757.85 | 34.881 | 55.476 | 19.4 |
| 13 | 98 | 27 | 253 | 1000 | 1034/3 | 1786.37 | 1818.89 | 1872.01 | 26.135 | 20.263 | 28.0 |
| 15 | 124 | 36 | 397 | 1000 | 1667/08 | 2892.85 | 2938.43 | 2987.39 | 32.63 | 41.011 | 36.4 |
| 17 | 98 | 23 | 241 | 200 | 795/33 | 1738.09 | 1780.32 | 1823.69 | 26.223 | 20.235 | 23.4 |
| 19 | 153 | 33 | 361 | 200 | 1538/2 | 3619.93 | 3694.05 | 3776.65 | 47.559 | 26.072 | 34.7 |
|  |  |  |  |  |  |  |  |  |  |  |  |
| ρ = 50% |  |  |  |  |  |  |  |  |  |  |  |
| index | **clusters** | **vehicles** | **vertices** | **Q** | **BKS** | **best** | **average** | **worst** | **variance** | **avg\_time** | **avg vehicles** |
| 11 | 37 | 18 | 400 | 1000 | 1101/51 | 1220.54 | 1285.73 | 1323.74 | 26.892 | 140.71 | 19.0 |
| 12 | 40 | 20 | 484 | 1000 | 1311/91 | 1468.56 | 1512.06 | 1538.39 | 20.35 | 160.09 | 20.0 |
| 13 | 58 | 28 | 253 | 1000 | 1053/47 | 1283.13 | 1313.73 | 1334.24 | 18.24 | 27.601 | 29.0 |
| 14 | 66 | 32 | 321 | 1000 | 1342/7 | 1690.19 | 1714.35 | 1738.92 | 17.303 | 34.624 | 33.0 |
| 15 | 73 | 36 | 397 | 1000 | 1657/22 | 2056.48 | 2088.56 | 2146.57 | 28.903 | 37.968 | 37.0 |
| 16 | 80 | 39 | 481 | 1000 | 2003/1 | 2483.53 | 2536.66 | 2574.27 | 31.63 | 41.825 | 40.0 |
| 17 | 47 | 24 | 241 | 200 | 881/66 | 1081.14 | 1101.15 | 1117.64 | 11.417 | 25.301 | 24.0 |
| 18 | 59 | 30 | 301 | 200 | 1199/12 | 1535.39 | 1562.26 | 1586.88 | 15.86 | 30.215 | 30.0 |
| 19 | 69 | 35 | 361 | 200 | 1612/33 | 2049.17 | 2102.60 | 2132.47 | 25.443 | 34.669 | 35.0 |
| 20 | 81 | 41 | 421 | 200 | 2278/64 | 2856.31 | 2941.40 | 2993.29 | 37.502 | 39.778 | 41.0 |
|  |  |  |  |  |  |  |  |  |  |  |  |
| ρ = 75% |  |  |  |  |  |  |  |  |  |  |  |
| index | **clusters** | **vehicles** | **vertices** | **Q** | **BKS** | **best** | **average** | **worst** | **variance** | **avg\_time** | **avg vehicles** |
| 2 | 13 | 13 | 321 | 700 | 10204/3 | 10520.98 | 10520.98 | 10520.98 | 0.0 | 31.146 | 13.0 |
| 4 | 13 | 13 | 481 | 1000 | 17077/5 | 17626.74 | 17626.74 | 17626.74 | 0.0 | 48.007 | 13.0 |
| 6 | 9 | 9 | 281 | 900 | 11452/0 | 11847.54 | 11847.54 | 11847.54 | 0.0 | 21.714 | 9.0 |
| 8 | 14 | 13 | 441 | 900 | 13882/23 | 14485.70 | 14516.30 | 14644.08 | 57.622 | 42.148 | 13.0 |
| 10 | 22 | 21 | 324 | 1000 | 1000/507 | 1028.73 | 1028.73 | 1028.73 | 0.0 | 31.6839 | 22.0 |
| 12 | 27 | 26 | 484 | 1000 | 1475/679 | 1502.48 | 1504.87 | 1518.18 | 4.54 | 20.698 | 26.0 |
| 14 | 42 | 41 | 321 | 1000 | 1520/546 | 1540.77 | 1552.04 | 1562.49 | 8.058 | 19.034 | 41.0 |
| 16 | 51 | 51 | 481 | 1000 | 2265/537 | 2308.49 | 2308.49 | 2308.49 | 0.0 | 22.7072 | 51.0 |
| 18 | 38 | 37 | 301 | 200 | 1392/153 | 1422.93 | 1422.93 | 1422.93 | 0.0 | 22.992 | 39.0 |
| 20 | 53 | 52 | 421 | 200 | 2502/34 | 2599.88 | 2599.88 | 2599.88 | 0.0 | 35.1147 | 53.0 |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
| ρ = 100% | |  |  |  |  |  |  |  |  |  |  |
| index | **clusters** | **vehicles** | **vertices** | **Q** | **BKS** | **best** | **average** | **worst** | **variance** | **avg\_time** | **avg vehicles** |
| 1 | 9 | 9 | 241 | 550 | 6293/036 | 6401.09 | 6401.09 | 6401.09 | 0.0 | 28.533 | 9.0 |
| 2 | 10 | 10 | 321 | 700 | 9879/586 | 10187.39 | 10187.39 | 10187.39 | 0.0 | 46.259 | 10.0 |
| 4 | 10 | 10 | 481 | 1000 | 16130/39 | 16664.14 | 16664.14 | 16664.14 | 0.0 | 75.881 | 10.0 |
| 5 | 5 | 5 | 201 | 900 | 8394/111 | 8679.34 | 8679.34 | 8679.34 | 0.0 | 49.568 | 5.0 |
| 7 | 8 | 8 | 361 | 900 | 11346/11 | 11705.95 | 11705.95 | 11705.95 | 0.0 | 49.408 | 8.0 |
| 8 | 10 | 10 | 441 | 900 | 13188/94 | 13572.42 | 13572.42 | 13572.42 | 0.0 | 48.076 | 10.0 |
| 10 | 16 | 16 | 324 | 1000 | 837/516 | 860.64 | 860.64 | 860.64 | 0.0 | 25.972 | 16.0 |
| 11 | 18 | 18 | 400 | 1000 | 1054/133 | 1091.88 | 1091.88 | 1091.88 | 0.0 | 42.810 | 18.0 |
| 19 | 34 | 34 | 361 | 200 | 1667/454 | 1696.12 | 1696.12 | 1696.12 | 0.0 | 25.765 | 34.0 |
| 20 | 39 | 39 | 421 | 200 | 2128/597 | 2158.55 | 2158.55 | 2158.55 | 0.0 | 28.537 | 39.0 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instance name | BKS |  | best | average | worst | variance | avg\_time | avg vehicles |
| e-n10-c2.map | 2016/57 |  | 2016.57 | 2016.57 | 2016.57 | 0.0 | 0.4629 | 2.0 |
| a-n15-c4.map | 1947/3 |  | 1961.98 | 1961.98 | 1961.98 | 0.0 | 0.3818 | 3.0 |
| b-n15-c4.map | 2602/56 |  | 2684.84 | 2933.94 | 3325.27 | 305.288 | 0.0462 | 2.4 |
| a-n20-c5.map | 2759/13 |  | 2788.79 | 2788.79 | 2788.79 | 0.0 | 0.4808 | 3.0 |
| c-n20-c5.map | 3028/83 |  | 3038.93 | 3118.86 | 3438.59 | 159.862 | 0.4508 | 4.0 |
| d-n20-c5.map | 2239/09 |  | 2239.09 | 2244.23 | 2290.48 | 15.419 | 0.5597 | 3.0 |
| e-n20-c5.map | 3343/34 |  | 3343.34 | 3343.34 | 3343.34 | 0.0 | 0.4799 | 4.0 |
| b-n30-c6.map | 3116/84 |  | 3285.15 | 3361.59 | 3514.92 | 83.185 | 0.8846 | 3.0 |

* GA One-minute run Results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ρ = 10% |  |  |  |  |
| file name | **BKS** | **vehicles** | **best** | **vehicles** |
| kelly01.ccvrp | 5759/25 | 9 | 16751/19 | 9 |
| kelly02.ccvrp | 9247/92 | 10 | 22744/33 | 10 |
| kelly03.ccvrp | 12904/6 | 10 | 28874/64 | 10 |
| kelly04.ccvrp | 17810/4 | 10 | 41207/75 | 11 |
| kelly05.ccvrp | 8960/31 | 5 | 16768/86 | 5 |
| kelly06.ccvrp | 10976/5 | 7 | 22215/53 | 7 |
| kelly07.ccvrp | 12485/8 | 9 | 28314/09 | 9 |
| kelly08.ccvrp | 13331/2 | 11 | 33422/68 | 11 |
| kelly09.ccvrp | 710/64 | 15 | 1065/26 | 16 |
| kelly10.ccvrp | 908/89 | 18 | 1287/338 | 19 |
|  |  |  |  |  |
|  |  |  |  |  |
| ρ = 25% |  |  |  |  |
| file name | **BKS** | **vehicles** | **best** | **vehicles** |
| kelly01.ccvrp | 6051/04 | 10 | 9409/454 | 10 |
| kelly03.ccvrp | 13692/6 | 10 | 18045/11 | 10 |
| kelly05.ccvrp | 9340/7 | 5 | 11209/1 | 5 |
| kelly07.ccvrp | 12348/1 | 9 | 16518/23 | 9 |
| kelly09.ccvrp | 717/63 | 16 | 1054/309 | 16 |
| kelly11.ccvrp | 1131/84 | 20 | 1664/447 | 20 |
| kelly13.ccvrp | 1034/3 | 27 | 1772/769 | 30 |
| kelly15.ccvrp | 1667/08 | 36 | 2851/527 | 38 |
| kelly17.ccvrp | 795/33 | 23 | 1635/427 | 23 |
| kelly19.ccvrp | 1538/2 | 33 | 3678/73 | 35 |
|  |  |  |  |  |
|  |  |  |  |  |
| ρ = 50% |  |  |  |  |
| file name | **BKS** | **vehicles** | **best** | **vehicles** |
| kelly11.ccvrp | 1101/51 | 18 | 1263/157 | 19 |
| kelly12.ccvrp | 1311/92 | 20 | 1498/865 | 20 |
| kelly13.ccvrp | 1053/47 | 28 | 1273/728 | 29 |
| kelly14.ccvrp | 1342/7 | 32 | 1673/094 | 33 |
| kelly15.ccvrp | 1657/22 | 36 | 2041/113 | 37 |
| kelly16.ccvrp | 2003/1 | 39 | 2516/534 | 40 |
| kelly17.ccvrp | 881/66 | 24 | 1088/141 | 24 |
| kelly18.ccvrp | 1199/12 | 30 | 1533/102 | 30 |
| kelly19.ccvrp | 1612/33 | 35 | 2030/046 | 35 |
| kelly20.ccvrp | 2278/64 | 41 | 2939/032 | 41 |
|  |  |  |  |  |
|  |  |  |  |  |
| ρ = 75% |  |  |  |  |
| file name | **BKS** | **vehicles** | **best** | **vehicles** |
| kelly02.ccvrp | 10204/3 | 13 | 10520/98 | 13 |
| kelly04.ccvrp | 17077/6 | 13 | 17626/74 | 13 |
| kelly06.ccvrp | 11452 | 9 | 11847/54 | 9 |
| kelly08.ccvrp | 13882/2 | 13 | 14492/69 | 13 |
| kelly10.ccvrp | 1000/51 | 21 | 1023/978 | 21 |
| kelly12.ccvrp | 1475/68 | 26 | 1502/488 | 26 |
| kelly14.ccvrp | 1520/55 | 41 | 1540/777 | 41 |
| kelly16.ccvrp | 2265/54 | 51 | 2308/495 | 51 |
| kelly18.ccvrp | 1392/15 | 37 | 1401/552 | 37 |
| kelly20.ccvrp | 2502/34 | 52 | 2572/521 | 52 |
|  |  |  |  |  |
|  |  |  |  |  |
| ρ = 100% |  |  |  |  |
| file name | **BKS** | **vehicles** | **best** | **vehicles** |
| kelly01.ccvrp | 6293/04 | 9 | 6401/095 | 9 |
| kelly02.ccvrp | 9879/59 | 10 | 10187/4 | 10 |
| kelly04.ccvrp | 16130/4 | 10 | 16664/14 | 10 |
| kelly05.ccvrp | 8394/11 | 5 | 8679/348 | 5 |
| kelly07.ccvrp | 11346/1 | 8 | 11705/95 | 8 |
| kelly08.ccvrp | 13188/9 | 10 | 13572/43 | 10 |
| kelly10.ccvrp | 837/516 | 16 | 860/6491 | 16 |
| kelly11.ccvrp | 1054/13 | 18 | 1091/883 | 18 |
| kelly19.ccvrp | 1667/45 | 34 | 1696/128 | 34 |
| kelly20.ccvrp | 2128/6 | 39 | 2158/556 | 39 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instance name | BKS |  | best | avg vehicles |
| e-n10-c2.map | 2016/57 |  | 2016.57 | 2.0 |
| a-n15-c4.map | 1947/3 |  | 1961.98 | 3.0 |
| b-n15-c4.map | 2602/56 |  | 2684.84 | 2.4 |
| a-n20-c5.map | 2759/13 |  | 2788.79 | 3.0 |
| c-n20-c5.map | 3028/83 |  | 3038.93 | 4.0 |
| d-n20-c5.map | 2239/09 |  | 2239.09 | 3.0 |
| e-n20-c5.map | 3343/34 |  | 3343.34 | 4.0 |
| b-n30-c6.map | 3116/84 |  | 3285.15 | 3.0 |

* GA Algorithm analysis:

Algorithm diversity seems to be low per some instances because of deleting non feasible solutions that could be create after Xover or mutation. Replacing parent instead of child when child isn’t feasible will also decrease population diversity.

After analysis for example in execution of ***a-n15-c4.map*** instance 30% of population was not feasible after Xover or mutation. As dimension of problem decrease diversity of population will also decrease.

Algorithm representation is another important factor of diversity.

The representation that described in the related paper cause many-to-one mapping between phenotype and genotype in the way that for example Ch1 = (6 7 0 1 2 3 0 4 5) and Ch2 = (1 2 3 0 6 7 0 4 5) have same fitness value cause they both have same global routes but the order of them is different.

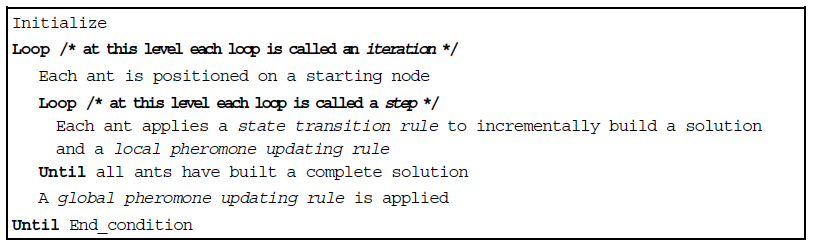
Even one chromosome itself could be mapped to many members of phenotype for because representation shows only sequence of clusters to be seen and doesn’t specifies node orders to be visiting in a cluster.

In compare results were reasonably close to best known solution.

But algorithm configuration that used in this report has significantly low number of generation (10 generation) in compare with paper configuration (200 generation) and it causes less chance of getting out of local optima.

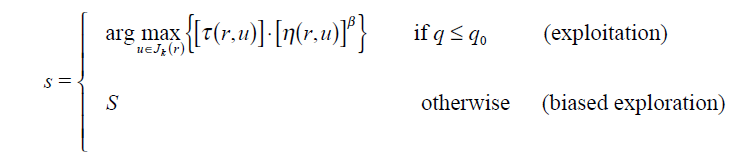
Mutation rate (20%) will shows its role of preserving diversity could not be achieved by small number of generations.

* Global and local subproblem with ACS:
  + Each ant solution form would be same as GA representation.
  + M ants would place randomly on graph clusters.
  + Local subproblem is the same as describe in GA and use in determining the global best solution in the colony.

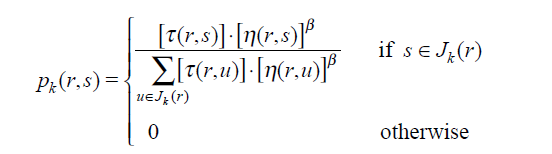


* **ACS state transition rule**

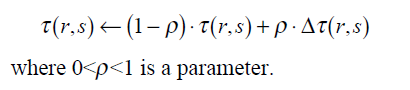
an ant positioned on node *r* chooses the city *s* to move to by applying the rule given:



we S set as follow:

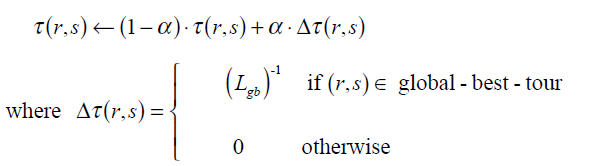


* **ACS local updating rule**



1. we set Dt (*r*,*s*) = t0 , where t0 is the initial pheromone level
2. (ii) we set Dt(*r*,*s*) = 0.

* **ACS global updating rule**



0<a<1 is the pheromone decay parameter, and *Lgb* is the length of the globally best tour from the beginning of the trial

* **ACS parameter settings**

b=2, *q*0=0.9, a=r=0.1, t0=(*n*·L*nn*)-1, where L*nn* is

the tour length produced by the nearest neighbor heuristic

values of the parameters were largely independent of the problem,

except for t0 for which, as we said, t0 =(*n*·L*nn*)-1. The number of ants used is *m*=10 (this

choice is explained in Section IV.B). Regarding their initial positioning, ants are placed

randomly, with at most one ant in each city.

* ACS Implementation
* State transition:

# exploitation

if q < Q0:

             s = candidates[args.index(max(args))]

         # exploration

         else:

             p = [arg/sum(args) for arg in args]

             # roulette wheel

             s = candidates[rouletteWheel(p)]

* Local update:

Update the pheromone path in both ways.

self.T[r][s] = (1-RHO) \* self.T[r][s] + RHO \* self.T0

self.T[s][r] = (1-RHO) \* self.T[s][r] + RHO \* self.T0

* Global update:

If edge(i,j) is in the best solution.set the delta as 1/ *Lgb*.

And after that update edge path in both ways.

 if i in bestSol and j in bestSol and + \

                        abs(bestSol.index(i) - bestSol.index(j)) == 1:

                    deltaT = 1/self.best[1]

                self.T[i][j] = (1-ALPHA) \* self.T[i][j] + \

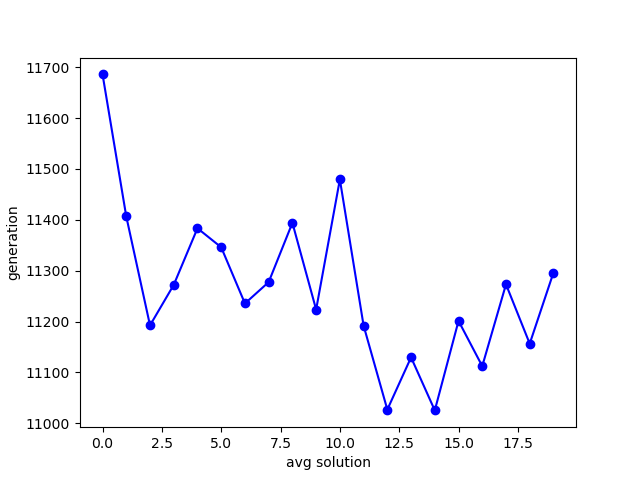
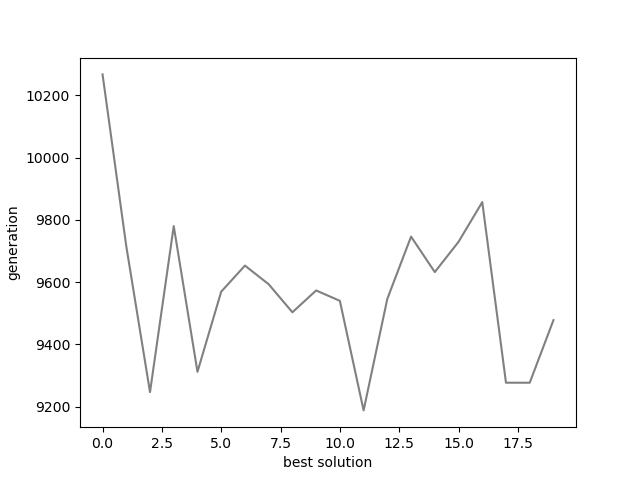
                    (ALPHA \* deltaT)

                self.T[j][j] = (1-ALPHA) \* self.T[j][i] + \

                    (ALPHA \* deltaT)

* ACS convergence process:

Best and average solution per iteration for instance 0.1%\_ kelly01



It shows that overall, the pheromone trail leads the search to better solution during algorithm.

Although exploration manner search could happen to.

* ACS Results:

iterations = 10

ALPHA = RHO = 0.2

Beta = 2

Q0 = 0.9

T0 = 0.00005 (value is based on some search results)

Ants\_num = max(Clusters\_num/10 , 2)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ρ = 10% |  |  |  |  |  |  |  |  |  |  |
| index | **clusters** | **vehicles** | **vertices** | **Q** | **BKS** | **Best** | **average** | **worst** | **variance** | **avg\_time** |
| 1 | 120 | 9 | 241 | 550 | 5759/25 | 9311/123 | 9524/57 | 10076/36 | 230/952 | 18.097 |
| 2 | 101 | 10 | 321 | 700 | 9247/92 | 14608/26 | 15142/85 | 15540/95 | 279/1 | 26.299 |
| 3 | 96 | 10 | 401 | 900 | 12904/6 | 19725/90 | 20017/18 | 20326/97 | 190/524 | 41.711 |
| 4 | 104 | 10 | 481 | 1000 | 17810/4 | 25823/08 | 26967/54 | 27998/27 | 678/783 | 64.355 |
| 5 | 49 | 5 | 201 | 900 | 8960/31 | 13030/843 | 13436/76 | 13761/94 | 262/992 | 9.5907 |
| 6 | 67 | 7 | 281 | 900 | 10976/5 | 15562/00 | 16137/14 | 16708/44 | 361/586 | 18.995 |
| 7 | 88 | 9 | 361 | 900 | 12485/8 | 19062/93 | 19561/38 | 20021/09 | 332/632 | 35.513 |
| 8 | 108 | 11 | 441 | 900 | 13331/2 | 20047/18 | 21490/08 | 22168/08 | 620/596 | 50.363 |
| 9 | 51 | 15 | 256 | 1000 | 710/64 | 787/7316 | 799/3658 | 809/9543 | 6/832 | 4.0530 |
| 10 | 56 | 18 | 324 | 1000 | 908/89 | 989/7373 | 996/7994 | 1005/142 | 5/058 | 6.0530 |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| ρ = 25% |  |  |  |  |  |  |  |  |  |  |
| index | **clusters** | **Vehicles** | **vertices** | **Q** | **BKS** | **best** | **average** | **Worst** | **variance** | **avg\_time** |
| 1 | 40 | 10 | 241 | 550 | 6051/04 | 7152/93 | 7324/41 | 7437/47 | 84/994 | 3.6489 |
| 3 | 38 | 10 | 401 | 900 | 13692/6 | 15311/51 | 15637/0 | 16007/93 | 199/748 | 9.7655 |
| 5 | 19 | 5 | 201 | 900 | 9340/7 | 10379/19 | 10486/2 | 10548/73 | 47/584 | 4.6584 |
| 7 | 34 | 9 | 361 | 900 | 12348/1 | 13918/10 | 14014/0 | 14080/33 | 58/941 | 9.4245 |
| 9 | 51 | 16 | 256 | 1000 | 717/63 | 767/8084 | 786/112 | 804/6502 | 10/616 | 3.5788 |
| 11 | 63 | 20 | 400 | 1000 | 1131/84 | 1221/950 | 1233/58 | 1246/456 | 9/698 | 8.7118 |
| 13 | 98 | 27 | 253 | 1000 | 1034/3 | 1165/701 | 1209/72 | 1233/166 | 20/108 | 5.5740 |
| 15 | 124 | 36 | 397 | 1000 | 1667/08 | 1863/173 | 1897/73 | 1923/843 | 15/59 | 13.096 |
| 17 | 98 | 23 | 241 | 200 | 795/33 | 1040/013 | 1070/78 | 1100/964 | 19/253 | 5.7340 |
| 19 | 153 | 33 | 361 | 200 | 1538/2 | 2026/525 | 2091/53 | 2131/029 | 25/555 | 19.384 |
|  |  |  |  |  |  |  |  |  |  |  |
| ρ = 50% |  |  |  |  |  |  |  |  |  |  |
| index | **clusters** | **vehicles** | **vertices** | **Q** | **BKS** | **best** | **average** | **worst** | **variance** | **avg\_time** |
| 11 | 37 | 18 | 400 | 1000 | 1101/51 | 1132/953 | 1152/95 | 1166/500 | 10/472 | 4.5660 |
| 12 | 40 | 20 | 484 | 1000 | 1311/91 | 1352/895 | 1355/98 | 1359/790 | 1/995 | 7.4104 |
| 13 | 58 | 28 | 253 | 1000 | 1053/47 | 1069/599 | 1080/11 | 1093/557 | 7/775 | 1.9484 |
| 14 | 66 | 32 | 321 | 1000 | 1342/7 | 1370/867 | 1390/00 | 1404/544 | 8/793 | 2.8605 |
| 15 | 73 | 36 | 397 | 1000 | 1657/22 | 1692/704 | 1702/33 | 1710/727 | 6/215 | 4.6001 |
| 16 | 80 | 39 | 481 | 1000 | 2003/1 | 2048/210 | 2067/94 | 2088/451 | 11/053 | 7.1417 |
| 17 | 47 | 24 | 241 | 200 | 881/66 | 899/0229 | 909/308 | 921/2504 | 5/69 | 1.2698 |
| 18 | 59 | 30 | 301 | 200 | 1199/12 | 1245/925 | 1255/40 | 1264/584 | 6/151 | 2.1820 |
| 19 | 69 | 35 | 361 | 200 | 1612/33 | 1666/086 | 1673/89 | 1679/125 | 4/428 | 3.0638 |
| 20 | 81 | 41 | 421 | 200 | 2278/64 | 2345/878 | 2362/97 | 2378/723 | 10/285 | 4.6380 |
|  |  |  |  |  |  |  |  |  |  |  |
| ρ = 75% |  |  |  |  |  |  |  |  |  |  |
| index | **clusters** | **vehicles** | **vertices** | **Q** | **BKS** | **best** | **average** | **worst** | **variance** | **avg\_time** |
| 2 | 13 | 13 | 321 | 700 | 10204/3 | 10204/3 | 10520/983 | 10520/983 | 10520/983 | 0 |
| 4 | 13 | 13 | 481 | 1000 | 17077/5 | 17077/59 | 17626/743 | 17626/743 | 17626/743 | 0 |
| 6 | 9 | 9 | 281 | 900 | 11452/0 | 11452/01 | 11847/545 | 11847/545 | 11847/545 | 0 |
| 8 | 14 | 13 | 441 | 900 | 13882/23 | 13882/23 | 14485/703 | 14491/99 | 14492/689 | 2/096 |
| 10 | 22 | 21 | 324 | 1000 | 1000/507 | 1000/507 | 1023/9781 | 1027/308 | 1028/7352 | 2/18 |
| 12 | 27 | 26 | 484 | 1000 | 1475/679 | 1475/679 | 1502/488 | 1517/308 | 1542/4236 | 16/796 |
| 14 | 42 | 41 | 321 | 1000 | 1520/546 | 1520/546 | 1540/7771 | 1542/4697 | 1546/4192 | 2/586 |
| 16 | 51 | 51 | 481 | 1000 | 2265/537 | 2265/537 | 2308/4947 | 2308/4947 | 2308/4947 | 0 |
| 18 | 38 | 37 | 301 | 200 | 1392/153 | 1392/153 | 1401/5521 | 1403/69 | 1422/931 | 6/414 |
| 20 | 53 | 52 | 421 | 200 | 2502/34 | 2502/34 | 2586/0475 | 2594/3502 | 2599/8854 | 6/779 |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| ρ = 100% | |  |  |  |  |  |  |  |  |  |
| index | **clusters** | **vehicles** | **vertices** | **Q** | **BKS** | **best** | **average** | **worst** | **variance** | **avg\_time** |
| 1 | 9 | 9 | 241 | 550 | 6293/036 | 6401/0949 | 6401/0949 | 6401/0949 | 0 | 1.9407 |
| 2 | 10 | 10 | 321 | 700 | 9879/586 | 10187/397 | 10187/397 | 10187/397 | 0 | 4.2388 |
| 4 | 10 | 10 | 481 | 1000 | 16130/39 | 16664/141 | 16664/141 | 16664/141 | 0 | 9.1544 |
| 5 | 5 | 5 | 201 | 900 | 8394/111 | 8679/3478 | 8679/3478 | 8679/3478 | 0 | 3.2554 |
| 7 | 8 | 8 | 361 | 900 | 11346/11 | 11705/953 | 11705/953 | 11705/953 | 0 | 5.3053 |
| 8 | 10 | 10 | 441 | 900 | 13188/94 | 13572/426 | 13572/426 | 13572/426 | 0 | 7.3177 |
| 10 | 16 | 16 | 324 | 1000 | 837/516 | 860/64914 | 860/64914 | 860/64914 | 0 | 2.9047 |
| 11 | 18 | 18 | 400 | 1000 | 1054/133 | 1091/8828 | 1091/8828 | 1091/8828 | 0 | 3.9081 |
| 19 | 34 | 34 | 361 | 200 | 1667/454 | 1696/1283 | 1696/1283 | 1696/1283 | 0 | 1.4732 |
| 20 | 39 | 39 | 421 | 200 | 2128/597 | 2158/5557 | 2158/5557 | 2158/5557 | 0 | 1.9021 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Instance name | BKS |  | best | average | worst | variance | avg\_time |
| e-n10-c2.map | 2016/57 |  | 2016.571 | 2016.571 | 2016.571 | 0 | 0.0123 |
| a-n15-c4.map | 1947/3 |  | 1961/9887 | 1961/9887 | 1961/9887 | 0 | 0.0153 |
| b-n15-c4.map | 2602/56 |  | 2684.840 | 2684.840 | 2684.840 | 0 | 0.0206 |
| a-n20-c5.map | 2759/13 |  | 2788.78 | 2788.78 | 2788.78 | 0 | 0.0283 |
| c-n20-c5.map | 3028/83 |  | 3038.936 | 3038.936 | 3038.936 | 0 | 0.0224 |
| d-n20-c5.map | 2239/09 |  | 2239.09 | 2239.09 | 2239.09 | 0 | 0.0288 |
| e-n20-c5.map | 3343/34 |  | 3343.3399 | 3343.3399 | 3343.3399 | 0 | 0.0263 |
| b-n30-c6.map | 3116/84 |  | 3125.264 | 3125.264 | 3125.264 | 0 | 0.0550 |

* ACS One-minute run Results:

|  |  |  |  |
| --- | --- | --- | --- |
| ρ = 10% |  |  |  |
| file name | **BKS** |  | **best** |
| kelly01.ccvrp | 5759/25 |  | 9580/8395 |
| kelly02.ccvrp | 9247/92 |  | 15034/384 |
| kelly03.ccvrp | 12904/6 |  | 19805/949 |
| kelly04.ccvrp | 17810/4 |  | 27206/636 |
| kelly05.ccvrp | 8960/31 |  | 12710/506 |
| kelly06.ccvrp | 10976/5 |  | 16211/253 |
| kelly07.ccvrp | 12485/8 |  | 19635/97 |
| kelly08.ccvrp | 13331/2 |  | 20358/169 |
| kelly09.ccvrp | 710/64 |  | 796/07424 |
| kelly10.ccvrp | 908/89 |  | 978/17974 |
|  |  |  |  |
|  |  |  |  |
| ρ = 25% |  |  |  |
| file name | **BKS** |  | **best** |
| kelly01.ccvrp | 6051/04 |  | 7162/3492 |
| kelly03.ccvrp | 13692/6 |  | 15173/437 |
| kelly05.ccvrp | 9340/7 |  | 10379/19 |
| kelly07.ccvrp | 12348/1 |  | 13591/408 |
| kelly09.ccvrp | 717/63 |  | 776/39522 |
| kelly11.ccvrp | 1131/84 |  | 1229/9866 |
| kelly13.ccvrp | 1034/3 |  | 1207/9695 |
| kelly15.ccvrp | 1667/08 |  | 1872/9098 |
| kelly17.ccvrp | 795/33 |  | 1027/5664 |
| kelly19.ccvrp | 1538/2 |  | 2054/3725 |
|  |  |  |  |
|  |  |  |  |
| ρ = 50% |  |  |  |
| file name | **BKS** |  | **best** |
| kelly11.ccvrp | 1101/51 |  | 1136/72 |
| kelly12.ccvrp | 1311/92 |  | 1354/43 |
| kelly13.ccvrp | 1053/47 |  | 1067/8632 |
| kelly14.ccvrp | 1342/7 |  | 1374/37 |
| kelly15.ccvrp | 1657/22 |  | 1692/8696 |
| kelly16.ccvrp | 2003/1 |  | 2044/8591 |
| kelly17.ccvrp | 881/66 |  | 901/4698 |
| kelly18.ccvrp | 1199/12 |  | 1233/3751 |
| kelly19.ccvrp | 1612/33 |  | 1654/4038 |
| kelly20.ccvrp | 2278/64 |  | 2353/2575 |
|  |  |  |  |
|  |  |  |  |
| ρ = 75% |  |  |  |
| file name | **BKS** |  | **best** |
| kelly02.ccvrp | 10204/3 |  | 10520/983 |
| kelly04.ccvrp | 17077/6 |  | 17626/743 |
| kelly06.ccvrp | 11452 |  | 11847/545 |
| kelly08.ccvrp | 13882/2 |  | 14492/689 |
| kelly10.ccvrp | 1000/51 |  | 1023/9781 |
| kelly12.ccvrp | 1475/68 |  | 1502/488 |
| kelly14.ccvrp | 1520/55 |  | 1540/7771 |
| kelly16.ccvrp | 2265/54 |  | 2308/4947 |
| kelly18.ccvrp | 1392/15 |  | 1401/5521 |
| kelly20.ccvrp | 2502/34 |  | 2572/5211 |
|  |  |  |  |
|  |  |  |  |
| ρ = 100% |  |  |  |
| file name | **BKS** |  | **best** |
| kelly01.ccvrp | 6293/04 |  | 6401/0949 |
| kelly02.ccvrp | 9879/59 |  | 10187/397 |
| kelly04.ccvrp | 16130/4 |  | 16664/141 |
| kelly05.ccvrp | 8394/11 |  | 8679/3478 |
| kelly07.ccvrp | 11346/1 |  | 11705/953 |
| kelly08.ccvrp | 13188/9 |  | 13572/426 |
| kelly10.ccvrp | 837/516 |  | 860/64914 |
| kelly11.ccvrp | 1054/13 |  | 1091/8828 |
| kelly19.ccvrp | 1667/45 |  | 1696/1283 |
| kelly20.ccvrp | 2128/6 |  | 2158/5557 |

|  |  |  |  |
| --- | --- | --- | --- |
| Instance name | BKS |  | best |
| e-n10-c2.map | 2016/57 |  | 2016.571 |
| a-n15-c4.map | 1947/3 |  | 1961/9887 |
| b-n15-c4.map | 2602/56 |  | 2684.840 |
| a-n20-c5.map | 2759/13 |  | 2788.78 |
| c-n20-c5.map | 3028/83 |  | 3038.936 |
| d-n20-c5.map | 2239/09 |  | 2239.09 |
| e-n20-c5.map | 3343/34 |  | 3343.3399 |
| b-n30-c6.map | 3116/84 |  | 3125.264 |

* Comparing GA and ACS Results:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ρ = 10% |  |  |  |  |  |  |
| file name | **BKS** |  | **GA best** | **GA avg time** | **ACS best** | **ACS avg time** |
| kelly08.ccvrp | 13331/2 |  | 33422/68 | 84.987 | 20047/18 | 50.363 |
| kelly10.ccvrp | 908/89 |  | 1287/338 | 45.193 | 989/7373 | 6.0530 |
|  |  |  |  |  |  |  |
| ρ = 25% |  |  |  |  |  |  |
| file name | **BKS** |  | **GA best** | **GA avg time** | **ACS best** | **ACS avg time** |
| kelly11.ccvrp | 1131/84 |  | 1664/447 | 55.476 | 1221/950 | 8.7118 |
| kelly19.ccvrp | 1538/2 |  | 3678/73 | 26.072 | 2026/525 | 19.384 |
|  |  |  |  |  |  |  |
| ρ = 50% |  |  |  |  |  |  |
| file name | **BKS** |  | **GA best** | **GA avg time** | **ACS best** | **ACS avg time** |
| kelly19.ccvrp | 1612/33 |  | 2030/046 | 34.669 | 1666/086 | 4/428 |
| kelly20.ccvrp | 2278/64 |  | 2939/032 | 39.778 | 2345/878 | 10/285 |
|  |  |  |  |  |  |  |
| ρ = 75% |  |  |  |  |  |  |
| file name | **BKS** |  | **GA best** | **GA avg time** | **ACS best** | **ACS avg time** |
| kelly10.ccvrp | 1000/51 |  | 1023/978 | 31.6839 | 1000/507 | 2/18 |
| kelly20.ccvrp | 2502/34 |  | 2572/521 | 35.1147 | 2502/34 | 6/779 |
|  |  |  |  |  |  |  |
| ρ = 100% |  |  |  |  |  |  |
| file name | **BKS** |  | **GA best** | **GA avg time** | **ACS best** | **ACS avg time** |
| kelly08.ccvrp | 13188/9 |  | 13572/43 | 48.076 | 13572/426 | 7.3177 |
| kelly19.ccvrp | 1667/45 |  | 1696/128 | 25.765 | 1696/1283 | 1.4732 |

As result shows ACS perform better in solution quality and algorithm time but the initial heuristic version of computing t0 =(*n*·L*nn*)-1 take much more time than GA (which I skipped that because of lacking of time), so time comparing of algorithms would not be much fair in this case.

In compare with GA we have both exploitation and exploration mechanism in ACS but instead of creating new solutions from parents in GA and select the better solutions based on pressure degree of the xOver and mutation result ,we choose constructive manner that creating the solution step by step based on some heuristic and collective information of colony(pheromone).

So, it seems that in this way search would be more targeted in compare with GA.

* Ordinal Data Analysis with Copeland’s method:

We chose 6 instance (last instance of each ρ = 10%, ρ = 25%, ρ = 50%, ρ = 75%, ρ = 100%) and b-n30-c6.map instance for comparing two algorithms against each other.

|  |  |  |
| --- | --- | --- |
|  | GA | ACS |
| GA | ---- | 0 |
| ACS | 4 | ---- |

2 times both was equal. GA = (0-4) = -4, ACS = (4-0) = 4

So its obvious that for this instances ACS works better and beats GA in performance.

Function value optimization with PSO

* Problem Description:

The problem purpose is to trying minimize function value by finding global optimum point in the search space.

benchmark functions are described as bellow:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Name | Test Function | S | Global opt. |  |
| **E** | Rosenbrock | C:\Users\somayeh\Desktop\Untitled.png |  |  | 0 |
| Step | C:\Users\somayeh\Desktop\Untitled.png |  |  | 0 |
| **M** | Ackley | C:\Users\somayeh\Desktop\Untitled.png |  |  | 0 |
| Griewank | C:\Users\somayeh\Desktop\Untitled.png |  |  | 0 |
| **H** | Rastrigin | C:\Users\somayeh\Desktop\Untitled.png |  |  | 0 |
| Generalized Penalized | C:\Users\somayeh\Desktop\Untitled.png |  |  | 0 |

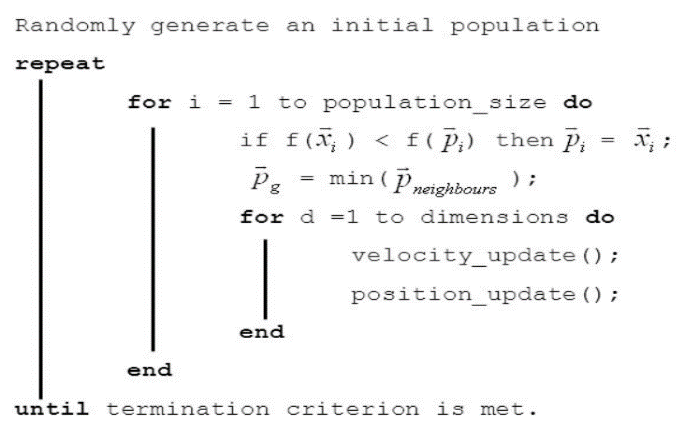
**Description:**

denotes the solution space dimension, denotes a subset of , and the global optimal solution and the global optimal value of classical benchmark functions are given in column 5 and column 6, respectively. Ten independent experiments must be completed for each optimization function considering .

* Algorithm Description:

PSO algorithm is a decentralized Swarm Intelligence search process. The swarm consist of particles with position and velocity related to them. Each particle remembers its best point ever seen as parameter calls “**pbest**”. The whole swarm best reached point remembers as parameter calls “**gbest**”.

The basic concept of PSO lies in accelerating each particle toward its **pbest** and the **gbest** locations, with a random weighted acceleration at each time.



Our PSO algorithm properties come in below:

* + **Initializing:**

Initial particles position set randomly base on problem domain.

initial particles velocity takes positive and negative 10% of particles position as velocity.

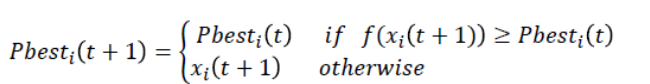
* + **Position & velocity update:**

**

*d* is the dimension, *c1* and *c2* are positive constants, *rand1* and *rand2* are random numbers, and *w* is the inertia weight.

usually *c1+ c2 = 4*. No good reason other than empiricism.

* + **Pbest & Gbest update:**





* + **Inertia weight update:**
    - Large inertia weight facilitates global exploration
    - small on facilitates local exploitation

By decreasing the inertia weight best performance archives.

many research works are conducted where the value is chosen as: w(initial) = 0.9 and decrease to w(final) = 0.2.

We use exponential manner for this purpose as below:

For initial iteration we set w= 0.9 and for the rest we keep it as 0.2.

* PSO learning process:

N = 30

ITERATIONS = 500

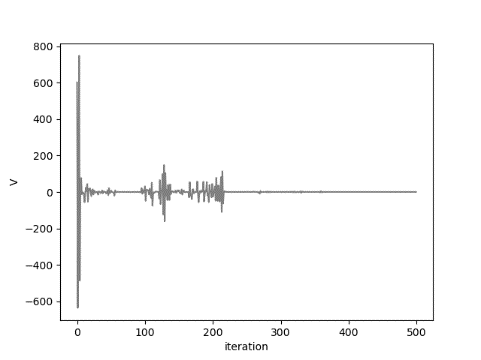
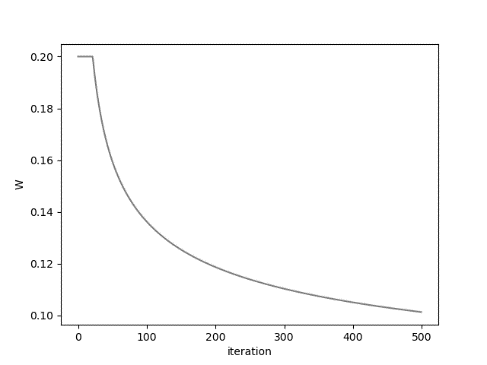
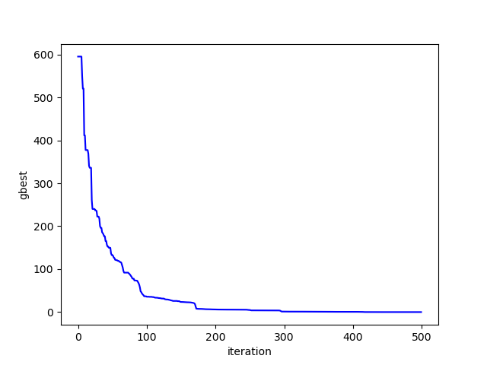
SWARM\_SIZE = 80

wMax = 0.9

wMin = 0.2

inertia mode = logarithmic

* + **Griewank** **instance**



Gbest /iteration inertia weight /iteration velocity/iteration

* PSO Results without bounded velocity:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Rosenbrock |  |  |  |  |  |  |  |
| n | **best** | **average** | **worst** | **variance** | **min\_time** | **avg\_time** | **max\_time** |
| *10* | 0/183 | 36/113 | 246/425 | 73/68 | 3/534 | 3/724 | 3/925 |
| *30* | 36/27 | 91264283 | 2/42E+08 | 1/12E+08 | 30/413 | 31/808 | 34/178 |
| *50* | 372345193 | 427387669 | 5/65E+08 | 52384448 | 81/514 | 85/15 | 88/959 |
|  |  |  |  |  |  |  |  |
| Step |  |  |  |  |  |  |  |
| n | **best** | **average** | **worst** | **variance** | **min\_time** | **avg\_time** | **max\_time** |
| *10* | 0 | 0 | 0 | 0 | 1/532 | 1/764 | 2/091 |
| *30* | 0 | 18830/998 | 73022/25 | 29188/32 | 10/902 | 11/979 | 13/843 |
| *50* | 3/867 | 106877/16 | 129563/6 | 36214/94 | 29/053 | 31/056 | 32/958 |
|  |  |  |  |  |  |  |  |
| Ackley |  |  |  |  |  |  |  |
| n | **best** | **average** | **worst** | **variance** | **min\_time** | **avg\_time** | **max\_time** |
| *10* | 0 | 0/116 | 1/155 | 0/347 | 3/071 | 3/155 | 3/31 |
| *30* | 1/34 | 9/527 | 20/417 | 7/166 | 22/673 | 25/178 | 29/165 |
| *50* | 18/311 | 20/243 | 20/811 | 0/956 | 60/583 | 64/546 | 68/012 |
|  |  |  |  |  |  |  |  |
| Griewank |  |  |  |  |  |  |  |
| n | **best** | **average** | **worst** | **variance** | **min\_time** | **avg\_time** | **max\_time** |
| 10 | 0/081 | 0/169 | 0/28 | 0/061 | 3/3 | 4/042 | 4/51 |
| 30 | 0/007 | 228/708 | 630/432 | 282/108 | 25/401 | 26/921 | 30/285 |
| 50 | 948/244 | 1053/5 | 1180/919 | 64/91 | 64/006 | 66/931 | 74/236 |
|  |  |  |  |  |  |  |  |
| Rastrigin |  |  |  |  |  |  |  |
| n | **best** | **average** | **worst** | **variance** | **min\_time** | **avg\_time** | **max\_time** |
| *10* | 2/985 | 13/432 | 35/818 | 8/304 | 2/977 | 3/093 | 3/652 |
| *30* | 41/87 | 366393/93 | 1840626 | 732424 | 22/784 | 23/651 | 24/491 |
| *50* | 468/99 | 2453097/4 | 3394777 | 1242255 | 62/361 | 65/773 | 70/785 |
|  |  |  |  |  |  |  |  |
| Generalized Penalized | |  |  |  |  |  |  |
| n | **best** | **average** | **worst** | **variance** | **min\_time** | **avg\_time** | **max\_time** |
| *10* | -3/065 | 2273442/9 | 15389408 | 4889766 | 5/798 | 6/182 | 7/254 |
| *30* | -1/022 | 251187239 | 6/7E+08 | 2/62E+08 | 49/613 | 56/328 | 61/644 |
| *50* | 915027566 | 1/044E+09 | 1/25E+09 | 1/02E+08 | 144/241 | 154/673 | 173/673 |

* PSO Results with bounded velocity:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Rosenbrock |  |  |  |  |  |  |  |
| n | **best** | **average** | **worst** | **variance** | **min\_time** | **avg\_time** | **max\_time** |
| *10* | 0.371 | 2.897 | 7.328 | 2.22 | 3/534 | 3/724 | 3/925 |
| *30* | 46.455 | 109172085.95 | 27455607 | 11508433 | 30/413 | 31/808 | 34/178 |
| *50* | 423.671 | 300610421.6 | 48870024 | 20181143 | 81/514 | 85/15 | 88/959 |
|  |  |  |  |  |  |  |  |
| Step |  |  |  |  |  |  |  |
| n | **best** | **average** | **worst** | **variance** | **min\_time** | **avg\_time** | **max\_time** |
| *10* | 0 | 1020.734 | 6461.495 | 2129.866 | 1/532 | 1/764 | 2/091 |
| *30* | 0 | 30204.419 | 69741.778 | 30895.7 | 10/902 | 11/979 | 13/843 |
| *50* | 108094.89 | 116438.34 | 123925 | 5939.35 | 29/053 | 31/056 | 32/958 |
|  |  |  |  |  |  |  |  |
| Ackley |  |  |  |  |  |  |  |
| n | **best** | **average** | **worst** | **variance** | **min\_time** | **avg\_time** | **max\_time** |
| *10* | 0 | 3.738 | 19.317 | 7.481 | 3/071 | 3/155 | 3/31 |
| *30* | 16.644 | 19.144 | 20.699 | 1.339 | 22/673 | 25/178 | 29/165 |
| *50* | 19.094 | 20.136 | 20.863 | 0.75 | 60/583 | 64/546 | 68/012 |
|  |  |  |  |  |  |  |  |
| Griewank |  |  |  |  |  |  |  |
| n | **best** | **average** | **worst** | **variance** | **min\_time** | **avg\_time** | **max\_time** |
| 10 | 0.034 | 37.907 | 138.325 | 49.55 | 3/3 | 4/042 | 4/51 |
| 30 | 0.038 | 98.677 | 546.92 | 197.694 | 25/401 | 26/921 | 30/285 |
| 50 | 0.511 | 719.646 | 1123 | 476.1 | 64/006 | 66/931 | 74/236 |
|  |  |  |  |  |  |  |  |
| Rastrigin |  |  |  |  |  |  |  |
| n | **best** | **average** | **worst** | **variance** | **min\_time** | **avg\_time** | **max\_time** |
| *10* | 10.945 | 21074.108 | 210531.1 | 63152.34 | 2/977 | 3/093 | 3/652 |
| *30* | 130.234 | 485757.737 | 1663701.3 | 741979.4 | 22/784 | 23/651 | 24/491 |
| *50* |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| Generalized Penalized | |  |  |  |  |  |  |
| n | **best** | **average** | **worst** | **variance** | **min\_time** | **avg\_time** | **max\_time** |
| *10* |  |  |  |  |  |  |  |
| *30* |  |  |  |  |  |  |  |
| *50* |  |  |  |  |  |  |  |

* PSO One-minute run Results:

|  |  |  |
| --- | --- | --- |
| Rosenbrock |  |  |
| n | best |  |
| 10 | 0 | [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0] |
| 30 | 79.348 | [0.988, 0.977, 0.955, 0.912, 0.833, 0.694, 0.482, 0.229, 0.028, -0.627, 0.906, 1.047, 1.029, 1.016, 1.012, 1.013, 1.021, 1.04, 1.081, 1.04, 1.021, 1.012, 1.01, 1.013, 1.024, 1.046, 1.054, 1.192, 1.416, 2.004] |
| 50 | 2658.411 | [1.114, 1.152, 1.177, 1.194, 1.07, 0.895, 0.325, 0.17, 0.091, 0.067, -0.073, -0.042, 0.386, 0.979, 0.729, -0.009, -0.893, 1.079, 0.857, 0.564, 0.022, 1.372, 2.304, 1.548, 1.059, 0.665, 0.029, 0.104, -0.187, 0.369, 0.113, -0.439, 0.846, 1.016, 1.331, 1.091, 0.787, 0.721, 0.48, 0.234, 0.201, 0.301, 0.032, -0.035, 0.172, 0.048, -1.323, 0.772, -0.193, 0.234] |
|  |  |  |
| Step |  |  |
| n | best |  |
| 10 | 0.0 | [-0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5] |
| 30 | 61661.144 | [-5.858, 5.602, -27.037, 88.178, -3.974, -55.965, 34.315, -43.495, 61.281, 5.569, 17.742, 5.192, 24.106, 20.063, -88.951, 6.362, 25.86, 3.915, -47.065, 45.11, -18.366, -108.129, -15.52, -82.158, -95.238, 44.802, -12.759, 91.51, -50.229, -50.748] |
| 50 | 0.005 | [-0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.432, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.502, -0.5, -0.5, -0.5, -0.502, -0.5, -0.485, -0.5, -0.5, -0.5, -0.498, -0.5, -0.5, -0.5, -0.5, -0.496, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.502, -0.5, -0.501, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5] |
|  |  |  |
| Ackley |  |  |
| n | best |  |
| 10 | 0 | [0.0, -0.0, -0.0, 0.0, -0.0, 0.0, 0.0, 0.0, 0.0, 0.0] |
| 30 | 19.007 | [1.0, -14.998, -38.995, -16.998, -21.997, -6.999, -11.999, 29.996, -7.999, 4.0, 7.999, 0.0, -10.999, -6.999, -2.0, -8.999, -7.999, -8.999, -12.998, 19.998, -1.0, 7.999, -18.998, -7.999, 7.999, -9.999, -25.997, -4.999, 13.998, -21.997] |
| 50 | 20.659 | [13.406, 24.386, 12.43, -30.978, -33.597, 8.828, 22.632, -10.782, 17.49, 2.285, -15.615, -28.179, 6.809, -8.869, -30.112, -20.477, -6.574, 8.922, 14.934, 5.427, 11.063, -5.604, 14.658, 11.491, -34.686, -22.39, -3.332, 6.457, 17.169, -16.106, -11.711, 21.702, -6.79, 13.995, -9.997, 14.195, -33.611, -1.921, -14.841, 17.232,  5.132, -5.156, -6.028, -34.355, 19.097, -7.047, 12.109, -4.295, -12.327, 26.478] |
|  |  |  |
| Griewank |  |  |
| n | best |  |
| 10 | 0.221 | [9.42, -0.0, -10.866, 6.271, -14.015, -0.0, 16.566, 8.85, -9.383, 0.0] |
| 30 | 4.011 | [3.14, 0.0, -5.433, 12.541, 0.0, -0.0, 0.0, 17.695, -0.0, -0.0, 10.361, -0.0, 123.787, -0.0, -0.0, 0.0, -0.0, -0.0, 0.0, -0.0, 0.0, 0.0, -0.0, -0.0, 0.0, 0.0, 0.0, -0.0, -0.0, -0.0] |
| 50 | 979.237 | [-92.36, 254.498, 415.037, -343.41, -31.687, 28.422, -526.851, -658.36, 444.411, -406.794, 500.601, -214.744, 570.503, 66.734, 61.593, -235.204, -448.856, -273.234, -132.496, -206.161, -327.569, -226.706, -161.922, -5.701, 269.226, -155.727, 140.683, -497.707, -473.065, 199.034, -582.401, 74.843, 244.42, -175.244, 42.905,  22.555, -648.864, 327.402, 133.435, -271.31, -351.635, -192.33, 53.969, -322.471, -265.427, 86.231, -183.941, -29.449, -94.98, -346.925] |
|  |  |  |
| Rastrigin |  |  |
| n | best |  |
| 10 | 15.919 | [-0.0, -1.99, -0.0, -0.0, -2.985, 0.995, -0.995, -0.0, 0.995, 0.0] |
| 30 | 147.252 | [-2.985, 0.0, -0.995, 0.995, -2.985, 2.985, -0.995, 0.0, 0.995, 1.99, 0.0, -0.0, 0.0, -6.964, 1.99, -0.0, 0.995, -0.995, 0.0, 0.995, -0.995, -0.0, -2.985, -0.995, -0.0, -0.995, -1.99, -0.995, -5.97, -1.99] |
| 50 | 1889.528 | [2.983, 2.003, 0.991, -5.973, -0.01, 0.006, -3.96, 0.986, 1.079, 2.992, 37.44, -1.002, 1.994, -0.997, -0.947, 1.987, 2.984, 0.988, -1.017, 0.993, 0.001, -6.04, 4.998, -2.979, -4.091, 1.013, 5.974, -0.991, -0.986, 0.008, 1.995, -1.999, -1.984, -0.006, 3.977, 2.013, 0.005, 1.988, -0.987, 0.996, 4.966, -0.007, -0.993, 1.027,  -1.99, -0.002, -6.965, 2.921, 0.984, -9.504] |
|  |  |  |
| Generalized Penalized | |  |
| n | best |  |
| 10 | -2.449 | [-6.881, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0] |
| 30 | 476743812.755 | [2.916, -45.029, -32.395, 1.811, -49.305, 23.057, -27.949, -39.153, -42.559, -13.423, -1.431, -46.387, -39.859, 11.13, 7.566, -22.278, 19.796, 23.79, -52.133, 17.245, 3.086, -29.625, -18.617, -10.062, 15.482, -11.785, -7.635, 29.141, -19.703, -6.618] |
| 50 | 807707467.747 | [-18.073, -17.976, 1.63, 9.591, -17.199, 18.514, 19.88, -24.707, -22.914, 9.877, -11.192, -20.418, 8.075, 43.82, -41.172, 26.551, -32.989, 23.419, -29.898, -37.092, -0.446, -1.641, -16.665, 8.251, 6.151, 4.04, -49.188, 8.808, -34.595, 2.383, -7.025, -12.166, -9.567, 0.085, 38.619, -41.997, -15.842, -3.27, -4.7, -16.262, 6.173, -12.139, -16.655, 21.764, -15.11, 20.135, -27.764, -53.662, 3.093, -38.806] |

* PSO Algorithm analysis:

As algorithm description shows it has a lot of properties to be set and tune well and it need excremental researches to find the best possible configuration of algorithm.

Its good practice to set a little higher exploration rate(c1) for problems with unknown search space

But in opposite higher exploitation rate(c2) is more suitable for known problems search area.

Types of inertial weight update methods have their own spatial characters and it’s hard to compare their performance, I think the choice could be dependent to search space area (Hight local optima positions or low).

Overall, I think this approach advantages would be good searching process (act good to find global optima) and simple implementation.

But its disadvantage would be lot of properties to config and manage.