CCVRP optimization with Genetic

- ## 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.

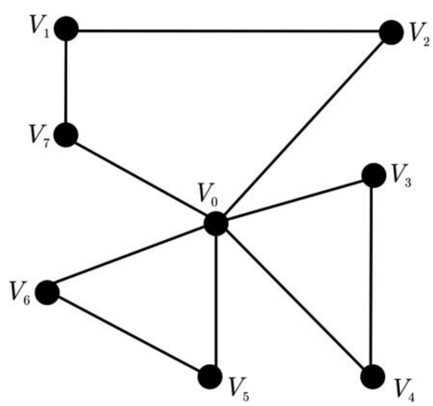
- ## Global subproblem:

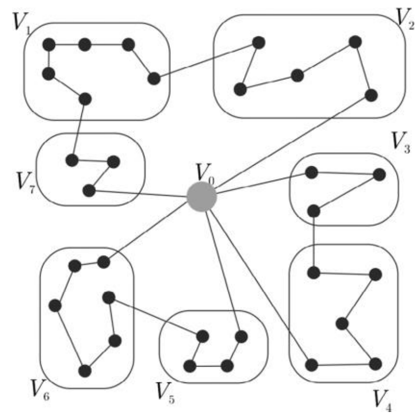
As mentioned, we solve this subproblem using GA, so we describe out GA as follow parts:

- 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.

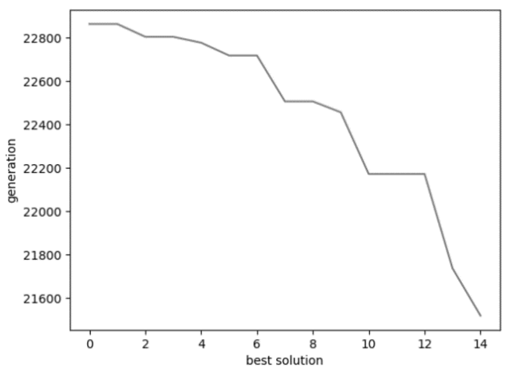
1. The set of nodes of Gp and Gp′ are the same.
2. The entries of the cost matrix Gp′ are defined as follows:
   (a) if $v_i, v_j \in V_k$ then $c'(v_i, v_j) = c(v_i, v_j)$;
   (b) if $v_i \in V_k, v_j \in V_l$ with $k \neq l$ then $c'(v_i, v_j) = c(v_i, v_j) + M$;
   where $M > \Sigma \in c(v_i, v_j)$.

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

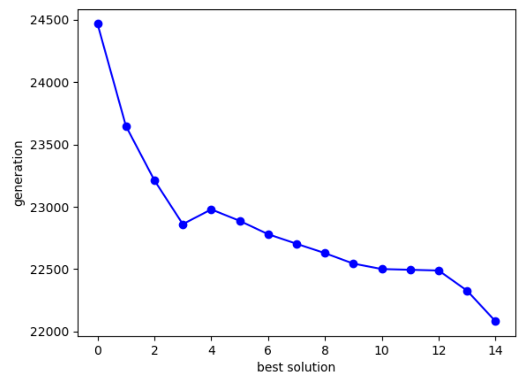
For solving local subproblem open source TSP solver used as bellow.

```
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

- 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

- 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

- 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.