Evolutionary Computation

Hristo Minkov (i6161227)[1] **and Zhecho Mitev (i6170114)**[1]

[1]Maastricht University, Department of Knowledge Engineering

Genetic algorithms use an evolutionary theory approach to computationally approximate optimal solutions for NP-hard problems. This work focuses on two such examples - the 0/1 Knapsack Problem and the Travelling Salesman Problem. There exist numerous representations, crossover operators, mutation strategies and selection procedures that can successfully tackle both problems. In this report, we implement several techniques from literature but also propose custom ones based on observations. All the newly introduced approaches are described in the respective sections. Finally, the paper reports and comments on the computational results.

genetic algorithm | travelling salesman problem | knapsack problem

Correspondence:
h.minkov@student.maastrichtuniversity.nl
z.mitev@student.maastrichtuniversity.nl

Introduction

0/1 Knapsack Problem

The well-known 0/1 knapsack problem is defined as follows: given a set of items (n), each with a weight w_i and a value v_i where i=1,..,n. The goal is to determine the items to include in the knapsack so that the total weight is less than some given capacity (C) and the total profit is maximized.

Genetic Algorithm Modelling. Each genetic algorithm has several characteristic properties, which are population, individual, reproduction, cross-over and mutation. In the case of the 0/1 Knapsack problem an individual is defined as a single knapsack, which contains a number of items each with a weight and a value. The value can be anything from 1 to n, where n is the count of all possible items and there are no two items with equal value. The weight is a pre-defined number from 1 to 10 for testing purposes, however random weights are also available in the code. The population consist of multiple knapsacks, which can be sorted by fitness (value), so that the best individuals are positioned on top and the worst are removed. If a knapsack has a total weight more than the capacity, the total value is assigned to zero because such individuals are not allowed. Such knapsacks might be penalized less severely, however it is very difficult to define a meaningful penalty since the value does not depend on the weight in our case.

Initially, we create n individuals and each of them is filled with random items so that the capacity of knapsack is not exceeded. After producing the population we can select the individuals to reproduce in several different ways (Elitism, Roulette Wheel, Tournament Selection). Since we want to get the best individuals, we choose the elitist method and the first 5 knapsacks are selected. Then, a cross-over method is executed on the selected knapsacks, which means that the knapsacks exchange items to create new knapsacks. In this paper we compare single-point crossover and uniform crossover. Moreover, a mutation occurs at a certain rate. If a knapsack mutates it loses one element and obtains another one. Various mutation rates are tested to determine what values fit best.

Monitoring the Population Properties. In order to better understand the evolution of each population we have a setting that allows us to see the value of each individual in the population after every generation. Thus, we can trace the diversity of each generation and see which knapsacks are able to survive and why. Moreover, we have a plot that shows the value of the fittest individual from every generation (see Fig.1). This illustrates how fast the population converges to a value and allows us to adjust the total generations. We noticed that when problem is simple (n < 10), around a hundred generations are enough. If n = 25, then (200 - 500) iteration are usually required, depending on the maximum capacity.

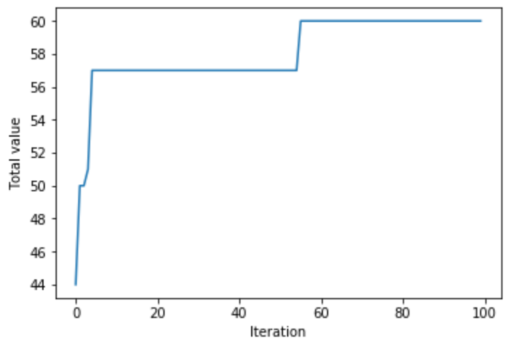


Fig. 1. Best values over generations in a single run of the genetic algorithm.

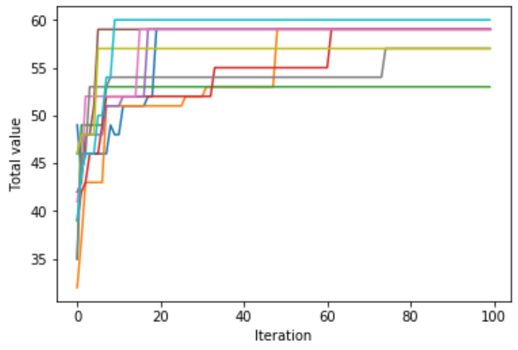


Fig. 2. Best values for several runs.

Mutation Tests. Even though the elitist method seems very intuitive for this type of problem, it ceases the diversity of the population, which is a negative indication since it cannot make improvement when all individuals are the same. In our case, even with a mutation rate of 20-30%, the population becomes stuck with many duplicated individuals after the 5th generation. Increasing the mutation rate to 50% rarely solves the issue either. Therefore, we decided to change the mutation procedure. It is very often the case that two items weigh less and are more valuable than another one. For example, consider the following items:

- Item 1 : value 7 and weight 10
- Item 2 : value 5 and weight 4
- Item 3 : value 4 and weight 3.

Clearly, it is much better to choose Item 2 and 3 instead of Item 1. Therefore, we propose a mutation method that removes one item and adds another two. We test the two mutations on a set of 8 items, where the capacity is 14 and the maximum possible value is 18[1]. Running the genetic algorithm 20 times with the initial mutation method rarely outputs more than 15 as a best value, the average is 13.25. Changing the mutation to the new type makes the algorithm reach the optimal several times out of the 20 and the average becomes 17.1, which is a great improvement.

Crossover Testing. Single-point crossover is a very simple way to reproduce individuals and it is a very popular crossover choice for different genetic algorithms. However, taking a subset and merging it with another set does not seem very intuitive. Usually, the knapsack value can improve just by swapping one item for another. Therefore, we implement a "swap-one" crossover where the individuals exchange only one item. To prove that our proposed method is better, we will hypothesize that the single-point crossover performs as good as the swap-one crossover and try to reject the null hypothesis. We will test the methods with the improved mutation method. If we use the previous problem setting (n=8, C=14) the new crossover method gives a mean score of 17.4 which is better than 17.1 however, it is not significant. Therefore, we introduce problem which is more complex - n = 15, capacity is 20 and the optimum is at 60. Such setting can better show the capabilities of the algorithms. We run the genetic algorithm 100 times for each method. It turns out that the mean score of the single-point is 55.91, where the swap-one method outputs an average of 59.08. Then we run a two-sample t-test, often used for comparing the means of two samples. The test rejects the null hypothesis, therefore indeed the more simple swap-one method is performing better in the aforementioned scenario.

Travelling Salesman Problem

The travelling salesman problem (TSP) is the most well-known combinatorial optimization problem. TSP is used to find a routing of a salesman who starts from a home location, visits a prescribed set of cities and returns to the original location in such a way that the total distance travelled is minimized and each city is visited exactly once (1). This report remodels the problem such that cities are represented by points on the unit circle. Each city is described by an angle [0-360] that sets its position on the circle. The distance between two points (cities) is given by:

$$d(c1, c2) = \sqrt{2 - 2cos(\gamma)} \quad \textbf{(1)}$$

where γ is the angle between cities c1 and c2 on the unit circle.

Custom Crossover Operator. Our TSP formulation introduces representation-specific features to the problem. We take advantage of the unit circle form and propose a custom crossover operator that outperforms some of the best performing path-based algorithms. Our research compares the custom approach for solving TSP using a GA and two more: Order Crossover Operator and Modified Cycle Crossover Operator as proposed by (2).

We introduce a new crossover operator that works similar to the partially mapped crossover operator (PMX), so we call it Custom PMX (CPMX). It works as follows:

Step 1: Choose two parents for mating - P_1 and P_2.

Step 2: Choose two random cut points on parents to build an offspring.

Step 3: The individuals between the two cut points from parent 1 (P_1) are mapped directly onto the first offspring (O_1).

Step 4: Individuals from P_2 are consecutively added to the empty gaps at O_1 in case the individual is not already in O_1.

Step 5: O_2 is constructed by reversing O_1.

The main difference with the original PMX approach (3) is that our modified version uses the method only to create the first offspring while the second is the exact reversed copy of the prior. Therefore, the order is correctly preserved which helps future generations of our GA variant to quickly and more accurately learn to produce better individuals. This comes from the fact that if O1 eventually becomes the optimal solution, then O2 would be an equivalent optimal solution (within the scope of our TSP formulation). Thus, each crossover operation performed by the Custom PMX (CPMX) would result in two individuals with the exact same fitness value but with entirely different structure. This adds variety and produces fitter individuals for our solution.

Consider the two selected parents as mentioned in Step 1 with the randomly one cut point between 3rd and 4th bits and the other cut point between 6th and 7th bits (both marked with '|') - Step 2:

[1]To see the exact setting read the comment of the code

$$P_1 = (3\ 4\ 8 \mid 2\ 7\ 1 \mid 6\ 5) \tag{2}$$
$$P_2 = (4\ 2\ 5 \mid 1\ 6\ 8 \mid 3\ 7) \tag{3}$$

The individuals between the two cut points are mapped directly to O_1 (Step 3):

$$O_1 = (\cdot\ \cdot\ \cdot \mid 2\ 7\ 1 \mid \cdot\ \cdot) \tag{4}$$

The consecutive elements from P_2 are 4, 2, 5, 1, 6, 8, 3 and 7. We start filling in the missing parts (marked by '$\cdot$') of O_1 with those numbers as we also skip numbers that are already in O_1 (Step 4). The first offspring looks as follows:

$$O_1 = (4\ 5\ 6 \mid 2\ 7\ 1 \mid 8\ 3) \tag{5}$$

Following, we reverse O_1 and thus, obtain the second offspring (Step 5):

$$O_2 = (3\ 8\ 1 \mid 7\ 2\ 6 \mid 5\ 4) \tag{6}$$

Hence, we've obtained our two offsprings O_1 and O_2 that become part of the next generation.

Computational Experiments & Discussion. This report uses a genetic algorithm in Python to compare the performance of the proposed crossover operator (CPMX) to the two implemented in (2). Additionally, the tests are performed on some other GA parameters: number of cities (N), mutation rate and number of generations. The genetic algorithm uses Elitist selection with 25% size (of the total population) of the 'elite' throughout all experiments. The mutation operation is also fixed and swaps the positions of two points in the individual. All experiments report on the resulting fitness value which is given by:

$$f(I) = \frac{1}{d(I)} \tag{7}$$

where d is the sum of the measured distances between points within an individual I.

Table 1 summarizes the results and shows that the proposed algorithm (CPMX) outperforms the other two already existing crossover operators on the test example with M=10 runs (measuring by the average result within 10 runs of the genetic algorithm). Appendix 1 provides more elaborate experiment results. Appendix 2 shows the plots for genetic distance over generations for the summary table experiments.

ID	N	Crossover	Average (10 runs)
1	20	CX2	0.1613
13		OX	0.1613
6		CPMX	0.1613
30	30	CX2	0.1323
29		OX	0.1498
25		CPMX	**0.1599**
38	45	CX2	0.0903
40		OX	0.0975
43		CPMX	**0.1473**
56	60	CX2	0.0788
58		OX	0.0775
62		CPMX	**0.1206**

Table 1. Summary table for all designed experiments with N = [20, 30, 45, 60] showing the best average results for each crossover operator.

The simplest experiment where N = 20 shows that every crossover operator is able to reach the optimal solution (with test IDs 1, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17 outputing the best solution every single run out of 10 in total).

In Table 3, 4 and 5 from Apendix 1, the average results illustrate the advantage of using the proposed crossover operator over the other two from literature. The CPMX approach has the highest fitness value for all the three experiments. However, this applies to all the tests where the mutation rate is 0.01. As the number of points increases (Tables 4 and 5 where N = 45 and N = 60 respectively), we can observe that the OX operator scores higher than the CPMX for measurements where the mutation rate is a lot higher: 0.075.

This suggests that, generally, the newly proposed CPMX operator outperforms the baseline CX2 and OX operators. However, if we test all three of them on a large test example ($N >= 45$) and execute the genetic algorithm with a significantly high value for the mutation rate, the OX operator manages to output higher fitness score than the recommended CPMX operator.

Conclusions

Genetic Algorithms provide a wide variety of parameters to adjust in order to successfully and quickly reach optimality. These GA specifications vary greatly depending on the problem definition and size. This report focuses on the 0/1 Knapsack and the Travelling Salesman problems. Our research shows that specific GA parameters are of greater interest based on the problem formulation. The 0/1 Knapsack section puts accent on the mutation and crossover properties of the GA. We show that a mutation that removes two items and adds a one other is much more suitable for our problem than removing only one item. Moreover, we proved that a single-point crossover performs worse than our "swap-one" crossover in a complex 0/1 Knapsack setting. The TSP section introduces a modified crossover operator and emphasises on the general problem representation. We illustrate that the proposed CPMX crossover operator outperforms the other two algorithms from the literature given the correct mutation rate. Future work suggestions from this report are to thoroughly examine the performance of the CPMX over different GA approaches and formulations.

Bibliography

1. Chetan Chudasama, S. M. Shah, and Mahesh Panchal. Comparison of Parents Selection Methods of Genetic Algorithm for TSP. 2011.
2. Abid Hussain, Yousaf Shad Muhammad, M. Nauman Sajid, Ijaz Hussain, Alaa Mohamd Shoukry, and Showkat Gani. Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator. 2017.
3. D. Goldberg and R. Lingle. Alleles, Loci and the Traveling Salesman Problem. 1985.

Appendix 1: TSP Experiment Results with Various Setups

ID	N	OPT	Crossover	Mutation rate	Number of Generations	Worst	Average	Best	OPT Solution
0	20	0.1613	CX2	0.01	200	0.1062	0.1503	0.1613	YES
1	20	0.1613	CX2	0.01	300	0.1613	**0.1613**	0.1613	YES
2	20	0.1613	CX2	0.01	500	0.0899	0.1342	0.1613	YES
3	20	0.1613	OX	0.01	200	0.0947	0.1272	0.1613	YES
4	20	0.1613	OX	0.01	300	0.0939	0.1478	0.1613	YES
5	20	0.1613	OX	0.01	500	0.0924	0.1354	0.1613	YES
6	20	0.1613	CPMX	0.01	200	0.1613	**0.1613**	0.1613	YES
7	20	0.1613	CPMX	0.01	300	0.1613	**0.1613**	0.1613	YES
8	20	0.1613	CPMX	0.01	500	0.1613	**0.1613**	0.1613	YES
9	20	0.1613	CX2	0.075	200	0.1613	**0.1613**	0.1613	YES
10	20	0.1613	CX2	0.075	300	0.1613	**0.1613**	0.1613	YES
11	20	0.1613	CX2	0.075	500	0.1613	**0.1613**	0.1613	YES
12	20	0.1613	OX	0.075	200	0.1052	0.1449	0.1613	YES
13	20	0.1613	OX	0.075	300	0.1613	**0.1613**	0.1613	YES
14	20	0.1613	OX	0.075	500	0.1613	**0.1613**	0.1613	YES
15	20	0.1613	CPMX	0.075	200	0.1613	**0.1613**	0.1613	YES
16	20	0.1613	CPMX	0.075	300	0.1613	**0.1613**	0.1613	YES
17	20	0.1613	CPMX	0.075	500	0.1613	**0.1613**	0.1613	YES

Table 2. TSP experiment results with N = 20 number of points.

ID	N	OPT	Crossover	Mutation rate	Number of Generations	Worst	Average	Best	OPT Solution
18	30	0.1599	CX2	0.01	200	0.0755	0.0915	0.1078	NO
19	30	0.1599	CX2	0.01	300	0.0894	0.0956	0.1007	NO
20	30	0.1599	CX2	0.01	500	0.0948	0.1271	0.1599	YES
21	30	0.1599	OX	0.01	200	0.0801	0.0853	0.0925	NO
22	30	0.1599	OX	0.01	300	0.0776	0.1068	0.1599	YES
23	30	0.1599	OX	0.01	500	0.0897	0.1214	0.1599	YES
24	30	0.1599	CPMX	0.01	200	0.0995	0.1478	0.1599	YES
25	30	0.1599	CPMX	0.01	300	0.1599	**0.1599**	0.1599	YES
26	30	0.1599	CPMX	0.01	500	0.1599	**0.1599**	0.1599	YES
27	30	0.1599	CX2	0.075	200	0.073	0.1171	0.1547	NO
28	30	0.1599	CX2	0.075	300	0.0752	0.1277	0.1599	YES
29	30	0.1599	CX2	0.075	500	0.1092	0.1498	0.1599	YES
30	30	0.1599	OX	0.075	200	0.0896	0.1323	0.1599	YES
31	30	0.1599	OX	0.075	300	0.0864	0.0936	0.1043	NO
32	30	0.1599	OX	0.075	500	0.0726	0.1111	0.1599	YES
33	30	0.1599	CPMX	0.075	200	0.1191	0.1492	0.1581	NO
34	30	0.1599	CPMX	0.075	300	0.1455	0.155	0.1599	YES
35	30	0.1599	CPMX	0.075	500	0.1413	**0.1515**	0.1599	YES

Table 3. TSP experiment results with N = 30 number of points.

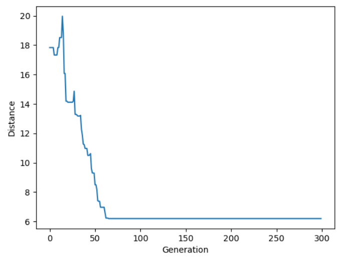
ID	N	OPT	Crossover	Mutation rate	Number of Generations	Worst	Average	Best	OPT Solution
36	45	0.1599	CX2	0.01	200	0.0474	0.0711	0.0947	NO
37	45	0.1599	CX2	0.01	300	0.0707	0.0881	0.1198	NO
38	45	0.1599	CX2	0.01	500	0.0765	0.0903	0.1026	NO
39	45	0.1599	OX	0.01	200	0.0604	0.0774	0.0933	NO
40	45	0.1599	OX	0.01	300	0.0661	0.0975	0.1599	YES
41	45	0.1599	OX	0.01	500	0.0685	0.0779	0.0828	NO
42	45	0.1599	CPMX	0.01	200	0.0894	0.1201	0.1599	YES
43	45	0.1599	CPMX	0.01	300	0.0966	**0.1473**	0.1599	YES
44	45	0.1599	CPMX	0.01	500	0.0832	0.1262	0.1599	YES
45	45	0.1599	CX2	0.075	200	0.0389	0.0473	0.0539	NO
46	45	0.1599	CX2	0.075	300	0.0429	0.0643	0.0986	NO
47	45	0.1599	CX2	0.075	500	0.0495	0.0739	0.0937	NO
48	45	0.1599	OX	0.075	200	0.0628	0.073	0.0836	NO
49	45	0.1599	OX	0.075	300	0.0701	0.0791	0.091	NO
50	45	0.1599	OX	0.075	500	0.0641	**0.0822**	0.0991	NO
51	45	0.1599	CPMX	0.075	200	0.0498	0.0528	0.0589	NO
52	45	0.1599	CPMX	0.075	300	0.0528	0.0558	0.0603	NO
53	45	0.1599	CPMX	0.075	500	0.0572	0.0586	0.0604	NO

Table 4. TSP experiment results with N = 45 number of points.

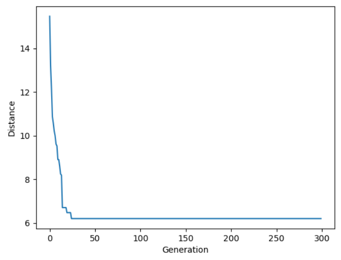
ID	N	OPT	Crossover	Mutation rate	Number of Generations	Worst	Average	Best	OPT Solution
54	60	0.1595	CX2	0.01	200	0.0317	0.0543	0.076	NO
55	60	0.1595	CX2	0.01	300	0.0357	0.0572	0.0774	NO
56	60	0.1595	CX2	0.01	500	0.0537	0.0788	0.1012	NO
57	60	0.1595	OX	0.01	200	0.0553	0.071	0.1091	NO
58	60	0.1595	OX	0.01	300	0.0686	0.0775	0.1005	NO
59	60	0.1595	OX	0.01	500	0.0402	0.0609	0.0806	NO
60	60	0.1595	CPMX	0.01	200	0.0705	0.0817	0.0874	NO
61	60	0.1595	CPMX	0.01	300	0.0886	0.1138	0.1595	YES
62	60	0.1595	CPMX	0.01	500	0.0897	**0.1206**	0.1595	YES
63	60	0.1595	CX2	0.075	200	0.0213	0.0272	0.0394	NO
64	60	0.1595	CX2	0.075	300	0.0225	0.0268	0.0332	NO
65	60	0.1595	CX2	0.075	500	0.0222	0.0275	0.0318	NO
66	60	0.1595	OX	0.075	200	0.0341	0.0361	0.0421	NO
67	60	0.1595	OX	0.075	300	0.0364	0.0396	0.042	NO
68	60	0.1595	OX	0.075	500	0.0354	**0.0412**	0.0467	NO
69	60	0.1595	CPMX	0.075	200	0.0283	0.0296	0.0312	NO
70	60	0.1595	CPMX	0.075	300	0.0286	0.031	0.0349	NO
71	60	0.1595	CPMX	0.075	500	0.0306	0.032	0.0345	NO

Table 5. TSP experiment results with N = 60 number of points.

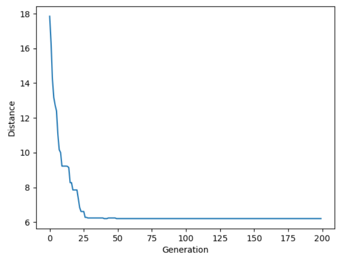
Appendix 2: TSP Summary Experiments Distance Over Generations Visualizations



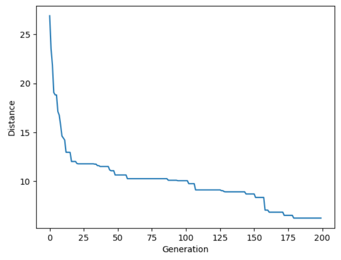
(a) ID = 1



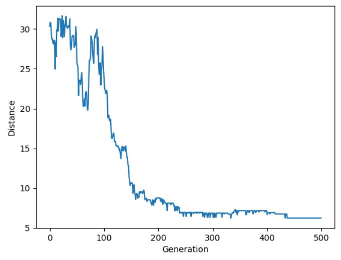
(b) ID = 13



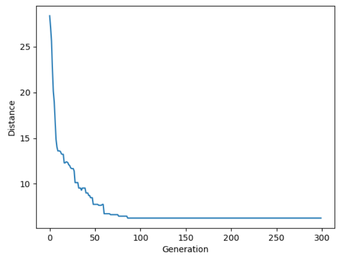
(c) ID = 6



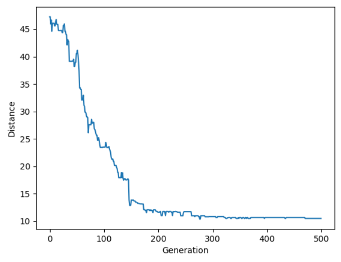
(d) ID = 30



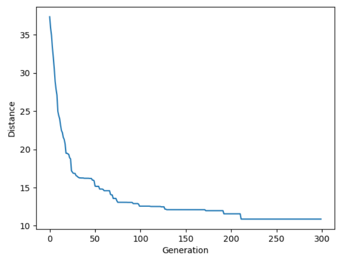
(e) ID = 29



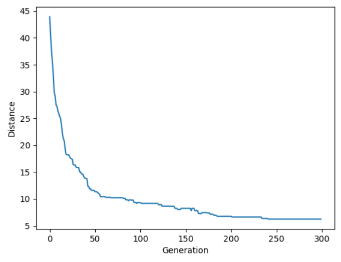
(f) ID = 25



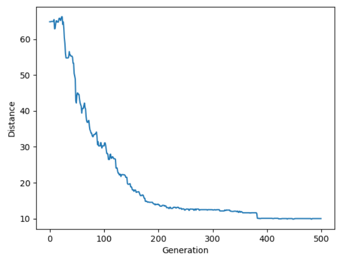
(g) ID = 38



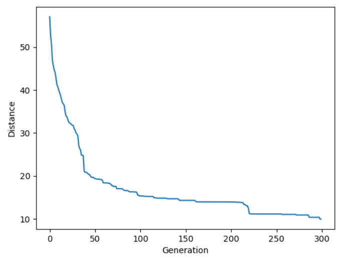
(h) ID = 40



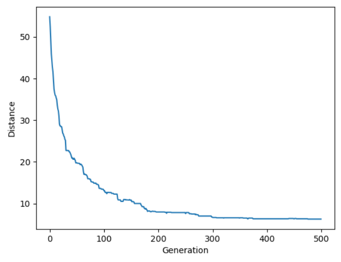
(i) ID = 43



(j) ID = 56



(k) ID = 58



(l) ID = 62

Fig. 3. Summary experiments visualizations based on distance over generations.