Project 1: K-Nearest Neighbors, Simulated Annealing, Genetic Algorithms

Luke Runnels

February 2, 2023

1 Introduction

1.1 K-Nearest-Neighbors

K-Nearest-Neighbors is a non-parametric supervised learning algorithm which utilizes the attributes of stored data points to *classify* new data points. If the set of all the data points in a dataset have a set of n numeric attributes with an associated class name, K-Nearest-Neighbors will first calculate the *distance* between stored data points and a incoming data point. Based on the kth *closest* data points relative to the incoming data point, the incoming data point is assigned the class name of the majority from the kth *closest* data points. There are two major variations of the K-Nearest-Neighbors algorithm: the *store-all* variant and the *store-errors* variant.

1.2 Simulated Annealing

Simulated Annealing is a local search optimization algorithm. It is an adaption of the hill climbing algorithm in a generic state space domain X with a range $F : X \to Y$ by allowing for occasional *bad* steps to be taken. Specifically, the algorithm will first select a random move from the state space domain. It will always accept moves that *improve* the state space range and occasionally allow for *bad* moves that worsen the range. This adjustment allows the algorithm to break out of local minimums and plateaus in the state space in order to reach the global maximum of the state space.

1.3 Genetic Algorithm

Genetic Algorithms are local search optimization techniques for locating the global maximum in a generic state space $F : X \to Y$ through *survival of the fittest.* In Genetic Algorithms, there is a population of individuals from the state space domain. The state space range for the domain specifies the fitness of every individual. To reach the global maximum, Genetic Algorithms employ techniques to exploit the known state space domain, and explore the unknown state space domain. For exploitation, the algorithm will make *selections* of individuals relative to their fitness. The selected individuals will be *crossed over* to make children to further exploit the state space. For exploration, children will be occasionally *mutated* to explore different parts of the state space outside of the known population pool.

2 Datasets and Domains used

2.1 Datasets

The following two datasets were utilized for training and testing K-Nearest-Neighbors. They are a collection of labeled data points with associated attributes and class names.

2.1.1 Artificial Dataset

The first dataset is an artificially generated dataset. It consists of 1000 data points. Each data point consists of a class name, two attributes, and an item name. The possibilities for the class name are the integer values 0 and 1, which could be referenced directly by K-Nearest-Neighbors. The two attributes were real numbers that could range in between 0 and 1. K-Nearest-Neighbors did not take the item name into consideration.

2.1.2 EMG Physical Action Dataset

The second dataset is a dataset from sourced from the UCI Machine Learning Repository [2]. It was originally compiled by Theo Theodoridis at the University of Essex. It consists of 10,000 data points. Each data point has consists of a class and eight attributes. There are twenty possible class names for every data point. Unlike the artificial dataset, all of the attributes are integers that can range from -4000 to 4000. For this project, a subset of 5000 data points are used from this dataset, with 250 data points from each of the twenty class names.

2.1.3 Traveling Salesman Problem Datasets

Two datasets were used to experiment with Simulated Annealing and the Genetic Algorithms. Both of the datasets are sourced from TSPLib [5]. They are adjacency matrices that represent a graph with weighted edges, where both the rows and columns indices represent a city and the associated entries in the matrices represent the distance between the row city and the column city. For this project, adjacency matrices of 15 rows and columns as well as 26 rows and columns were experimented on. These represent graphs of 15 cities and 26 cities respectively.

2.2 Function Domains

For this project, several difficult benchmark function domains were tested to evaluate the performance of Simulated Annealing and the Genetic Algorithms. For these of the following functions, a n-dimensional vector $\mathbf{x}$ is inputted into a variety of function definitions with subjection to a variety of constraints. Simulated Annealing and Genetic Algorithms will attempt to find a $\mathbf{x}$ that globally minimizes each function definition while satisfying the constraints.

The functions that are tested in this project are modifications to the following functions: Sphere, Griewank's function, Shekel's foxholes, Michalewit'z function, Langermann's function, Odd square function, and Bump function. These functions were sourced from the 1st and 2nd contest of evolutionary optimization[1][4].

2.3 Function Schedules

For Simulated Annealing, the performance is heavily dependent on a temperature factor. The temperature factor controls the probability of accepting lower utility states. A Simulated Annealing schedule consists of a initial temperature T_0 and a final temperature T_N. In general, if the current temperature is near T_0, there is a high probability of accepting low utility states. As the current temperatures lowers toward T_N, the probability of accepting low utility states decreases exponentially. Three schedules will be included in the Simulated Annealing experiments, which have been sourced from [3]. These schedules are non-monotonic additive functions for adjusting the current temperature between T_0 and T_N.

2.3.1 Non-monotonic additive linear schedule

$$T_t = T_N + (T_0 - T_N)(\frac{k-t}{k}) \tag{1}$$

where T_t is the current temperature, t is the current time in the schedule, and k is the number of iterations in the schedule.

2.3.2 Non-monotonic additive quadratic schedule

$$T_t = T_N + (T_0 - T_N)(\frac{k-t}{k})^2 \tag{2}$$

where T_t is the current temperature, t is the current time in the schedule, and k is the number of iterations in the schedule. Unlike the linear schedule, T_t is cooled is a faster rate. Therefore, this schedule has less opportunity to accept low utility states.

2.3.3 Non-monotonic additive trigonometric schedule

$$T_t = T_N + \frac{1}{2}(T_0 - T_N)(1 + cos(\frac{k\pi}{t})) \tag{3}$$

where T_t is the current temperature, t is the current time in the schedule, and k is the number of iterations in the schedule. Unlike the linear schedule or the quadratic schedule, the rate of cooling plateaus as T_t is approaching either T_0 or T_N, while it increases as T_t moves away from T_0 or T_N.

2.4 N Fold Cross Validation

For evaluating K-Nearest-Neighbors, N Fold cross validation is employed to test the training and testing accuracy of a labeled vector $\mathbf{x}_n$. The labeled vector is roughly evenly divided into n sub blocks. If x_i is a considered a block of *size* n from the labeled vector, then the majority of the blocks are assigned to the training vector

$$x^i_{train} \leftarrow x_n \setminus x_i \tag{4}$$

and x_i is assigned the testing vector

$$x^i_{test} \leftarrow x_i \tag{5}$$

The average accuracies are computed by

$$acc_{train} = \sum_{i=0}^{n} eval(x^i_{train})/n \tag{6}$$

for the training vectors and

$$acc_{test} = \sum_{i=0}^{n} eval(x^i_{test})/n \tag{7}$$

for the testing vectors, where *eval* is the KNN classification for a vector $\mathbf{x}_n$.

3 Algorithm Explanations

3.1 K-Nearest-Neighbors

3.1.1 Classification Strategy

In K-Nearest-Neighbors, a new n-dimensional $\mathbf{x}$ are classified based on the k lowest *distances* of previously classified n-dimensional vectors. The *distance* between an incoming vector and previous vectors is calculated by the Minkowski distance.

$$d(\mathbf{x}^i_n, \mathbf{x}^j_n) = (\sum_{k=1}^{n} \|x_{ik} - x_{jk}\|^p)^{\frac{1}{p}} \tag{8}$$

where p specifies particular cases of distance. Based on Kilian Q. Weinberger's explanation from the second lecture of CS 4/5780 at Cornell University[7], the majority label of an incoming $\mathbf{x}^i_n$ is assigned by $\mathbf{x}^j_n$ subjected to the constraint

$$d(\mathbf{x}^i_n, \mathbf{x}^j_n) \geq \max_{(\mathbf{x}^i_n, \mathbf{x}^k_n) \subseteq D} d(\mathbf{x}^i_n, \mathbf{x}^k_n) \tag{9}$$

where D is the set of all closest k vectors. The assigned majority label is the classification of the incoming vector $\mathbf{x}^i_n$.

3.1.2 Store All Training

For every incoming vector, the *store-all* training strategy stores all incoming vectors, without comparison against any of the previously stored vectors. This training strategy makes for low time complexity for training the algorithm. However, there is higher time complexity for classifying new vectors after the training strategy and higher space complexity.

3.1.3 Store Error Training

For every incoming vector, the *store-error* training strategy stores the first k vectors of a particular classification. Afterwards, incoming vectors are stored only if they are misclassified by the previously stored vectors. This training strategy makes for higher time complexity for training, but lower time complexity for classifying new incoming vectors. In addition, the algorithm will have lower space complexity with this strategy.

3.2 Simulated Annealing

Based in a state state X with a value function $F : X \to Y$, Simulated Annealing attempts a find state whose evaluation is the global maximum. The algorithm is subjected a scheduling scheme with a domain variable T and a schedule $S : (T_0, T_N) \to T$ where T_0 and T_N are both the global minimum and maximum of S. According to the algorithm for Simulated Annealing laid out by Russell et al. [6] in Figure 4.5, while $T \geq T_0$ according to the schedule, the algorithm will choose a random successor state $x_{new} \in X$ from a previously selected state x_{cur}.
The difference in utility from the value function is calculated as

$$\Delta E = F(x_{cur}) - F(x_{new}) \tag{10}$$

The probability of accepting x_{new} is given by

$$P(x_{cur}, x_{new}, T) = \begin{cases} 1 & \text{if } \Delta E > 0 \\ e^{-\frac{\Delta E}{T}} & \text{if } \Delta E \leq 0 \end{cases} \tag{11}$$

where T is the current variable from the schedule. According to (11), any state x_{new} that is closer to the global maximum is always accepted. However, for states farther away, the probability is indirectly influenced by T. If T is a higher value, then x_{new} has a higher probability of being accepted, x_{new} has a lower probability of being accepted if T is a lower value.

3.3 Genetic Algorithms

Based in a state space X with a value function $F : X \to Y$, a Genetic Algorithm attempts find a state whose evaluation is a global maximum. Unlike Simulated Annealing though, this algorithm is subjected to a *evolutionary* scheme. Based on explanation by Russell et al. [6], a Genetic Algorithm keep tracks of a collection of n-dimensional vectors, which is known as the population pool. The *fitness* of each of the vectors are calculated by the value function

F. To find the global maximum of F, a Genetic Algorithm employs a direct metaphor of evolutional biology. For a set number of generations, the population goes through *selection*, *crossover*, and *mutation* operations.

3.3.1 Selection

The selection process is gathering a new population pool from the existing population. The algorithm will randomly select vectors from the existing population proportionally to their *fitness* value. There are common interpretations for the selection operation. Two common interpretations are *roulette* selection and *rank* selection.

In *roulette* selection, two individuals from the population pool are selected based on a probability that is directly proportional to its *fitness* value. In *rank* selection, all of the individuals are assigned a *rank* value based on the value of their *fitness* value. For example, the highest fitness may be assigned rank 1, the second highest fitness may be assigned rank 2, and so on. Individuals from the population pool are then selected based on a probability that is proportional to the *rank* values.

3.3.2 Crossover

10	12	13	14	15	16	→	10	11	12	20	21	22
17	18	19	20	21	22	→	17	18	19	14	15	16

Table 1: An example crossover for two integer representations with a single splice in the middle

The crossover process is a recombination procedure from two random individuals from the population pool that was generated in the selection process. There are many different implementations for a crossover operation.

A common interpretation is illustrated in Table 1. A random splicing point is selected between two vectors. The section of the first recombined vector before the splicing point contains the part the first vector before the splicing and the section after the splicing point contains the part of the second vector after the splicing point. Conversely, in the second recombined vector, the combination order for the first and second vectors are reversed.

3.3.3 Mutation

0	1	1	0	0	→	0	0	1	0	0

Table 2: An example mutation for a binary representation

After the crossover operation, there is a small probability that an individual undergoes a mutation. Just like with the crossover operation, there are many different implementations that can be done. Table 2 illustrates of an example of a mutation for a binary representation.

A random point in the representation can be selected and the selected gets its bit number flipped. In general, the mutation operation should not happen very often. If a individual is selected for a mutation, then the change to its structure should be minimal.

4 Performance and Analysis

4.1 Performance and Analysis of K-Nearest-Neighbors

The following two figures illustrate the average training and testing accuracies for the *store-all* and *store errors* of K-Nearest Neighbors. The full artificial dataset of 1000 data points was used for training the algorithm for Table 1, but only a subset of the EMG dataset of the first 250 datapoints from all 20 classes was used for training the algorithm in Table 2. The parameters that were experimented on was shuffling the dataset or not, and setting the value of k from 1 and 30. It should be noted that in the following figures, k is always odd, which ensures that there are no ties in label assignments for the classification stage of an incoming vector. In addition, the Euclidean distance(p = 2) was used to calculate the *distance* between all incoming vectors and previously stored vectors.

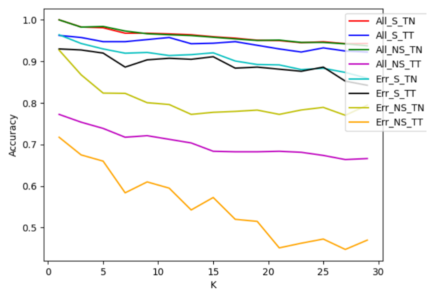

Figure 1: Average training/testing accuracies(TN/TT) for the store-all/store-errors(All/Err) with the shuffled/non-shuffled(S/NS) artificial Dataset, N = 5 for N Fold cross validation training and testing runs

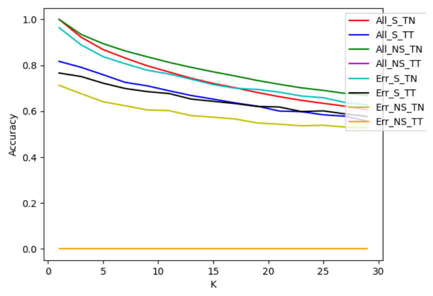

Figure 2: Average training/testing(TN/TT) accuracies for the store-all/store-errors(All/Err) with shuffled/non-shuffled(S/NS) EMG Physical Action Dataset, N = 5 for N Fold cross validation training and testing runs

For both the artificial dataset and the EMG dataset, as the value of k grows, the average accuracy decreases. This is expected, as K-Nearest-Neighbors is an algorithm that is fundamentally prone to overfitting and underfitting. This phenomenon is particularly prominent with the *store-error* algorithm trained on the non-shuffled EMG dataset, where the testing set is entirely misclassified. Also, Russell el al. [6] mentions the curse of dimensionality, where data instances with more attributes can suffer from more overfitting or underfitting. This will certainly prevalent with the EMG dataset, where all of the data instances have eight attributes while the artificial dataset has data instances with only two attributes.

4.2 Documented minimums for function domains and the traveling salesman problem

The approximate global minimums for the function domains and the traveling salesman problem are documented in the following table. They will be used as a basis for comparison and evaluation for the Simulated Annealing and Genetic Algorithm tests. It must be emphasized that the experiments were conducted in controlled state spaces, with fixed ranges for the states $\mathbf{x}$. Likewise, the values in the following table, besides the lengths calculated for the 15 and 26 city traveling salesman problem, are only approximations. These values may not reflect the absolute global minimum for these functions, where $\mathbf{x}_n \in \mathbf{R}^n$.

Problem	Minimum
Sphere	0
Griewank	0
Shekel	$-\infty$
Michalewitz	-1.5
Langermann	-1.5
Odd Square	-1.0
Bump	0
TSP 15	291
TSP 26	937

Table 3: The global minimums for the domains in a fixed range for this project

4.3 Performance and Analysis of Simulated Annealing

For the following tables, the final predicted minimum of Simulated Annealing is documented. For common ground, all of the tests established T_0 and $T_N = 1000$ as constants. The value k, which determines the number of iterations, and the type of schedules, which were described in section 2.3.2, are the parameters that were varied for the experiments. All of the best minimums that were found from the tests are highlighted in green.

4.3.1 Simulated Annealing Performance for function domains

For the function domains, all successor states are n dimensional vectors in these experiments are calculated by adding the previously selected vectors with a random number from a Gaussian Distribution with $\mu = 0$ and $\sigma = 1$.

For most of the function domains, the trigonometric schedule seemed to be the best schedule for approaching the global minimum. Surprisingly, the quadratic schedule actually predicted the best minimum for Griewank's function and Langermann's function. This is particular surprising for Griewank's function, which is a noisy monotonic function. Nevertheless, the trigonometric schedule was probably the best schedule overall, because ∇S is low if $t \to 0$, which implies that $T \to T_N$. This most likely allowed the algorithm to thoroughly explore the state space range before converging a local minimum, thus yielding better results. Most of the function domains are quite noisy, so these results do make sense. In fact, on average, the trigonometric schedule predicted better results than the quadratic schedule for Langermann's function. The algorithm most likely got lucky with selecting random states on this run test.

Schedules			
k	Linear Sch.	Quad. Sch.	Trig. Sch.
500	2.82262E+00	5.74078E-01	9.55547E-01
1000	1.55068E+00	2.50674E-01	2.29857E-01
2500	2.11722E+00	3.78072E-02	5.65081E-02
5000	7.58838E-01	7.29444E-03	8.98146E-02
10000	1.35461E-02	5.34385E-02	2.11988E-02

Table 4: Simulated annealing results for the Sphere function with different k's and schedules. $T_0 = 0$ and $T_N = 1000$

Schedules			
k	Linear Sch.	Quad. Sch.	Trig. Sch.
500	5.05786E+00	5.02946E-01	4.19384E-01
1000	4.38613E+00	4.29340E+00	4.03054E+00
2500	8.85622E+00	1.92107E+00	1.18412E+01
5000	2.21910E+00	3.85103E+00	4.11344E+00
10000	2.94498E+01	3.13626E+00	2.17859E+00

Table 5: Simulated annealing results for Griewank's function with different k's and schedules. $T_0 = 0$ and $T_N = 1000$

Schedules			
k	Linear Sch.	Quad. Sch.	Trig. Sch.
500	-1.97007E+00	-1.54440E+02	-1.89825E+02
1000	-1.96905E+02	-5.14487E+02	-6.19683E+01
2500	-3.87862E+03	-1.25617E+03	-1.77523E+03
5000	-6.55932E+03	-4.78547E+02	-5.29488E+05
10000	-8.57415E+03	-7.26988E+03	-4.81397E+02

Table 6: Simulated annealing results for Shekel's foxholes with different k's and schedules. $T_0 = 0$ and $T_N = 1000$

Schedules			
k	Linear Sch.	Quad. Sch.	Trig. Sch.
500	1.28777E-01	-7.75789E-01	-3.38036E-01
1000	-1.00344E-04	-6.93837E-01	-8.72850E-01
2500	1.36231E-01	-9.56611E-01	-9.27678E-01
5000	2.99972E-04	-7.48408E-01	-1.32437E+00
10000	-5.53509E-01	-9.28648E-01	-9.81883E-01

Table 7: Simulated annealing results for Michalewitz's function with different k's and schedules. $T_0 = 0$ and $T_N = 1000$

Schedules			
k	Linear Sch.	Quad. Sch.	Trig. Sch.
50	-2.41250E-01	-6.79313E-01	-9.12364E-02
1000	4.14794E-02	-5.86179E-01	-5.90701E-01
2500	3.40820E-01	-9.77574E-01	-1.18787E+00
5000	4.55565E-01	-1.29916E+00	-8.57774E-01
10000	-5.62270E-01	-8.53476E-01	-1.25589E+00

Table 8: Simulated annealing results for Langermann's function with different k's and schedules. $T_0 = 0$ and $T_N = 1000$

Schedules			
k	Linear Sch.	Quad. Sch.	Trig. Sch.
500	-7.79932E-19	-8.87331E-19	-3.25219E-05
1000	-9.11460E-04	1.68003E-20	-3.11482E-01
2500	3.15801E-06	-6.60190E-17	-1.50234E-14
5000	-1.20950E-05	-6.29412E-01	-5.28057E-12
10000	-9.98607E-02	1.76827E-10	2.86481E-08

Table 9: Simulated annealing results for odd square function with different k's and schedules. $T_0 = 0$ and $T_N = 1000$

Schedules			
k	Linear Sch.	Quad. Sch.	Trig. Sch.
500	6.21381E-03	1.41942E-05	6.06891E-03
1000	5.39846E-03	2.06923E-04	1.15364E-03
2500	1.97822E-02	6.30140E-05	6.15382E-05
5000	3.04394E-04	4.63078E-03	9.54551E-07
10000	7.21343E-06	1.66102E-05	1.60466E-04

Table 10: Simulated annealing results for the bump function with different k's and schedules. $T_0 = 0$ and $T_N = 1000$

4.3.2 Simulated Annealing Performance for the traveling salesman problem

Unlike the function domains, successor states were determined by simply switching two random elements in a tour. All of the best minimum tours that were found from the tests are highlighted in green.

In constrast to the function domains, the quadratic schedule predicted the best minimum tours. This is reasonable, because the graph size of the traveling salesman problem are quite small. Hence, the *value* functions from these problems are easily exploitable, compared to some of the function domains. So, the trigonometric schedule's early exploration is not particular powerful here. The quadratic schedule is quick to converge to the local minimum, where appears to work for both of the graph sizes.

Schedules			
k	Linear Sch.	Quad. Sch.	Trig. Sch.
500	498	431	401
1000	498	368	432
2500	421	358	293
5000	367	321	329
10000	357	286	311

Table 11: Simulated annealing results for the 15 city traveling salesman problem with different k's and schedules. $T_0 = 0$ and $T_N = 1000$

Schedules			
k	Linear Sch.	Quad. Sch.	Trig. Sch.
500	1850	1565	1581
1000	1460	1443	1342
2500	1451	1243	1302
5000	1475	1015	1180
10000	1332	1059	1128

Table 12: Simulated annealing results for the 26 city traveling salesman problem with different k's and schedules. $T_0 = 0$ and $T_N = 1000$

4.4 Performance and Analysis of Genetic Algorithms

For the following tables, the fittest predicted minimum of the Genetic Algorithm's final population pool is documented. For common ground, all of the domains are experimented on with 10 individuals and 100 individuals in the population pool, with a 100 generations per test. Individuals are selected from the population pool via *rank* selection described 3.3.1. The two fittest individuals are always copied from the last generation.

4.4.1 Genetic Algorithm Performance for function domains

For the function domains, all individuals are represented as both real numbers and bitstrings. Real number representation is used to evaluate each individual with the fitness function, while bitstring representation is used for the crossover and mutation operations. In the crossover operation, the *random splice* method from section 3.3.1 is used on the bitstring representations. In the mutation operation, the *bit flip* method from section 3.3.3 is used.

There are some interesting patterns from emerged from the results. For most of the functions, the mutation rate and the size of the population seem to interchange each other. For the relatively monotonic functions such as the Sphere function and Griewank's function, the higher mutation rate in the smaller population predicted better. However, for the rest of the functions, the higher population predicted a better result.

Somewhat surprisingly, the noisy functions such as Michalewitz's function and Langermann's function depended less on the mutation rate. It would be assumed that since there is more local minimums in these functions, then a higher mutation rate would allow the population pool to explore the state space and converge to the best possible minimum, but that does not seem to be the case here. In fact, the smoother functions seem to benefit from a higher mutation rate. Nevertheless, as the mutation rate grows, the more random actions are taken by the Genetic Algorithm. So, these results do seem reasonable.

Unlike the mutation rate, a higher rate and noisiness correlation could be assumed for the crossover rate. The smooth functions seemed to converge with lower crossover rates, while the noisy functions converged with higher crossover rates.

	Mutation			
Crossover	0	0.05	0.1	0.25
0	1.1531E+01	1.9480E-01	2.9264E-01	4.7826E-07
0.5	8.0086E+00	4.0226E+00	2.5513E-01	4.1725E-03
0.75	4.6805E+00	2.1772E+00	2.6088E-01	1.3096E-01
1	1.0717E+00	4.7377E+00	1.5703E+00	6.6571E-02

Table 13: Genetic Algorithm results for the sphere function with 10 individuals and elitism applied.

	Mutation			
Crossover	0	0.05	0.1	0.25
0	2.2063E-01	1.1289E+00	3.9063E-03	6.4468E-02
0.5	2.5733E-01	2.4605E-04	4.0293E-03	6.1035E-03
0.75	1.8657E-01	6.2500E-02	2.5000E-01	6.2500E-02
1	6.0179E-01	6.4468E-02	1.0020E+00	3.9368E-03

Table 14: Genetic Algorithm results for the sphere function with 100 individuals and elitism applied.

	Mutation			
Crossover	0	0.05	0.1	0.25
0	6.1413E-01	5.3218E-01	4.7296E-01	1.5271E-02
0.5	1.3199E+00	9.6845E-01	4.8568E-01	1.7774E-01
0.75	6.2589E-01	3.4376E-01	7.7030E-02	1.9722E-02
1	1.8713E+00	2.7875E-01	6.7753E-02	3.0900E-01

Table 15: Genetic Algorithm results for Griewank's function with 10 individuals and elitism applied.

	Mutation			
Crossover	0	0.05	0.1	0.25
0	2.1375E-01	3.0115E-01	1.6196E-01	3.0238E-02
0.5	3.5735E-01	1.9151E-02	5.7770E-02	5.9430E-02
0.75	2.8879E-01	6.7751E-02	1.6061E-01	1.4458E-02
1	7.3272E-01	1.5251E-02	2.0561E-01	3.6152E-01

Table 16: Genetic Algorithm results for Griewank's function with 100 individuals and elitism applied.

	Mutation			
Crossover	0	0.05	0.1	0.25
0	-5.8684E+00	-4.4340E+01	-3.0439E+01	-1.6987E+02
0.5	-1.8834E+01	-8.7485E+00	-5.4305E+02	-6.9022E+02
0.75	-4.5750E+00	-2.1338E+01	-7.8382E+01	-6.5432E+01
1	-5.9177E+00	-1.3039E+01	-9.5498E+00	-1.6002E+06

Table 17: Genetic Algorithm results for Shekel's Foxholes with 10 individuals and elitism applied.

	Mutation			
Crossover	0	0.05	0.1	0.25
0	-3.8716E+01	-3.0719E+06	-1.1222E+03	-1.8461E+03
0.5	-7.1333E+01	-1.3309E+03	-1.1820E+08	-4.6397E+03
0.75	-3.2798E+01	-7.1706E+03	-2.0462E+08	-2.1154E+03
1	-7.4748E+01	-2.9759E+02	-1.9193E+05	-5.4364E+03

Table 18: Genetic Algorithm results for Shekel's Foxholes with 100 individuals and elitism applied.

	Mutation			
Crossover	0	0.05	0.1	0.25
0	-9.7039E-01	-1.3806E+00	-1.4813E+00	-1.3223E+00
0.5	-9.3197E-01	-1.4542E+00	-1.1644E+00	-1.9913E+00
0.75	-6.5610E-01	-7.2485E-01	-1.0518E+00	-9.0078E-01
1	-9.9545E-01	-1.0539E+00	-1.7368E+00	-8.7388E-01

Table 19: Genetic Algorithm results for Michalewitz's Function with 10 individuals and elitism applied.

	Mutation			
Crossover	0	0.05	0.1	0.25
0	-1.1139E+00	-1.8368E+00	-1.8295E+00	-1.5823E+00
0.5	-1.6645E+00	-1.5053E+00	-1.7203E+00	-1.4519E+00
0.75	-9.3254E-01	-9.9710E-01	-1.0002E+00	-1.8536E+00
1	-1.3482E+00	-1.7043E+00	-1.4255E+00	-9.7724E-01

Table 20: Genetic Algorithm results for Michalewitz's Function with 100 individuals and elitism applied.

	Mutation			
Crossover	0	0.05	0.1	0.25
0	-1.1615E+00	-9.9475E-01	-9.6315E-01	-1.0597E+00
0.5	-1.1873E+00	-5.5653E-01	-1.0708E+00	-1.3535E+00
0.75	-1.5999E+00	-9.3825E-01	-1.2722E+00	-1.0312E+00
1	-7.8208E-01	-7.7466E-01	-9.4052E-01	-1.4108E+00

Table 21: Genetic Algorithm results for Langermann's Function with 10 individuals and elitism applied.

	Mutation			
Crossover	0	0.05	0.1	0.25
0	-1.5587E+00	-1.1089E+00	-1.3327E+00	-1.4169E+00
0.5	-1.6370E+00	-1.7365E+00	-1.0248E+00	-1.4499E+00
0.75	-1.2251E+00	-1.3666E+00	-1.3512E+00	-1.3948E+00
1	-1.3465E+00	-8.9348E-01	-1.4362E+00	-1.4574E+00

Table 22: Genetic Algorithm results for Langermann's Function with 100 individuals and elitism applied.

	Mutation			
Crossover	0	0.05	0.1	0.25
0	-2.0352E-03	-7.8384E-03	-6.5476E-08	-5.7197E-01
0.5	-1.0111E+00	-4.7791E-03	-3.5821E-04	-2.6288E-01
0.75	-9.3852E-01	-1.3726E-01	-2.2387E-07	-1.1891E-02
1	-2.7986E-01	-4.2310E-03	-3.5938E-01	-4.5760E-03

Table 23: Genetic Algorithm results for the Odd Square function with 10 individuals and elitism applied.

	Mutation			
Crossover	0	0.05	0.1	0.25
0	-4.1757E-01	-1.0229E+00	-7.4490E-01	-1.0073E+00
0.5	-2.8506E-01	-7.4477E-01	-4.1753E-01	-1.0230E+00
0.75	-2.0857E-01	-7.4490E-01	-5.5828E-02	-7.4147E-01
1	-2.1642E-01	-7.4497E-01	-7.4497E-01	-7.4496E-01

Table 24: Genetic Algorithm results for the Odd Square function with 100 individuals and elitism applied.

	Mutation			
Crossover	0	0.05	0.1	0.25
0	4.9046E-05	1.4965E-05	5.5625E-06	1.7984E-06
0.5	2.8536E-02	2.9417E-06	1.2260E-08	5.4738E-06
0.75	1.0228E-04	3.9709E-06	1.0771E-10	2.1070E-08
1	1.5495E-04	8.7526E-04	1.0071E-03	3.5923E-06

Table 25: Genetic Algorithm results for the Bump function with 10 individuals and elitism applied.

	Mutation			
Crossover	0	0.05	0.1	0.25
0	7.2699E-09	1.0345E-08	7.6939E-08	2.7402E-11
0.5	3.2805E-09	5.3633E-12	2.1847E-09	1.8780E-13
0.75	9.2560E-13	4.1021E-10	1.6221E-11	2.6746E-11
1	8.8334E-08	1.7474E-08	1.4507E-07	1.9330E-09

Table 26: Genetic Algorithm results for the Bump function with 100 individuals and elitism applied.

4.4.2 Genetic Algorithm Performance for the Traveling Salesman Problem

For the traveling salesman problem, all individuals are represented as permutations of integers. The crossover operation is similar to the *random splice* method but with integer instead of bits. However, the mutation operation is similar to the successor operation similar from the Simulated Annealing experiments, where two random elements from the permutations are switched.

Just like the Simulated Annealing experiments, the population of 100 ended up predicted the best minimum tour. For the 15 city graph, no crossover and an interchange between the mutation rate and the population indicates that this problem's fitness function is relatively. However, unlike the function domains and the 15 city graph, the algorithm predicted the best tour with a higher crossover rate and mutation rate. This most likely indicates the noisiness of the problem's fitness function, as well as the larger state space, compared to all of the function domains.

	Mutation			
Crossover	0	0.05	0.1	0.25
0	488	438	310	339
0.5	519	382	389	334
0.75	508	408	423	351
1	540	409	382	305

Table 27: Genetic Algorithm results for the 15 city traveling salesman problem with 10 individuals and elitism applied.

	Mutation			
Crossover	0	0.05	0.1	0.25
0	457	285	300	299
0.5	487	357	381	338
0.75	480	295	352	302
1	494	294	336	309

Table 28: Genetic Algorithm results for the 15 city traveling salesman problem 100 individuals and elitism applied.

	Mutation			
Crossover	0	0.05	0.1	0.25
0	2368	1727	1649	1370
0.5	2132	1914	1498	1340
0.75	2344	1772	1793	1447
1	2140	1881	1604	1234

Table 29: Genetic Algorithm results for the 26 city traveling salesman problem with 10 individuals and elitism applied.

	Mutation			
Crossover	0	0.05	0.1	0.25
0	2159	1361	1138	1139
0.5	2011	1370	1203	1009
0.75	1962	1196	1072	1052
1	2069	1401	1208	1145

Table 30: Genetic Algorithm results for the 26 city traveling salesman problem 100 individuals and elitism applied.

5 Conclusion

K-Nearest-Neighbors is a simple, yet powerful supervised learning algorithm. It can be utilized efficiently for small datasets with fewer attributes. However, its pitfalls quicly become apparent as larger datasets with more attributes.

Simulated Annealing and Genetic Algorithms are powerful techniques for converging to a local minimum. However, their effectiveness heavily dependents on the problem's function smoothness, which will require parameters geared toward either more exploration or exploitation. Disappointingly, most of the tests ran in this project did not reach the best possible global minimum, although they did come close.

References

[1] Hugues Bersini, Marco Dorigo, Stefan Langerman, Gregory Seront, and L Gambardella. Results of the first international contest on evolutionary optimisation (1st iceo). In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 611–615. IEEE, 1996.

[2] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[3] José Fernando Díaz Martín and Jesús M Riaño Sierra. A comparison of cooling schedules for simulated annealing. In *Encyclopedia of Artificial Intelligence*, pages 344–352. IGI Global, 2009.

[4] Kenneth V Price. Differential evolution vs. the functions of the 2/sup nd/iceo. In *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*, pages 153–157. IEEE, 1997.

[5] G Reinhelt. Tsplib: a library of sample instances for the tsp (and related problems) from various sources and of various types. *URL: http://comopt. ifi. uniheidelberg. de/-software/TSPLIB95*, 2014.

[6] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearsons, 4 edition, 2021.

[7] Kilian Weinberger. Lecture 2: K-nearest neighbors (curse of dimensionality). *Retrieved Oct*, 21:2021, 2021.