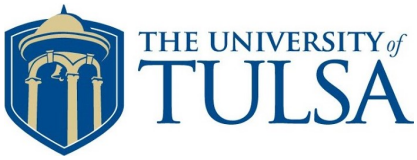


Independent Study: Topics in AI
CS-7993-SE

Project 1: Adaptation

Instructor:
Prof. Sandip Sen

Author:
Selim Karaoglu

Feb 5, 2023

1 Introduction

In this project, the main focus is the adaptation with nearest neighbor and local search methods. We present the experiment in two parts; in first part we implement the k-Nearest Neighbors algorithm and test it on an artificial and a real life datasets. After we conduct the experiments, we measure the accuracy of the models with cross-validation. Furthermore, we present the local search algorithms Simulated Annealing and Generic Algorithms in the second part of this project. We implement both algorithms with the provided pseudo-code and test these methods on different 3d surfaces on maximization and minimization problems provided with several functions.

2 k Nearest Neighbor (kNN) Supervised Learner

In this section, we implemented kNN algorithm with two different training instance storing options; store all instances and store first k misclassified instances for each class. With this implementation, we tested kNN algorithm with varied parameters on two datasets. To provide insight for kNN algorithm, we implemented N-fold cross validation with the pseudo-code provided in the assignment paper. The kNN algorithm is trained on two datasets; labeled examples and glass identification datasets.

2.1 Labeled Examples Dataset

This dataset is provided with the homework files. Since the original dataset file had no file extension, we imported the dataset in MS Excel and converted it to a .csv file. This dataset contains 1000 instances with a single feature value of 2 coordinates and a target value. Feature values are continuous, floating numbers between 0 and 1 and target values are binary, 0 and 1.

2.2 Glass Identification Dataset

This dataset is glass identification dataset by USA Forensic Science Service; containing 6 types of glass; defined in terms of their oxide content (i.e. Na, Fe, K, etc) provided in UCI Machine Learning Repository dataset archives [1]. This dataset is provided to analyze different glass types gathered from the crime scenes and classify them to possibly use as an evidence. The attributes of this dataset are (in order): Id, RI: refractive index, Na: Sodium (unit measurement: weight percent in corresponding oxide, as are attributes 4-10), Mg: Magnesium, Al: Aluminum, Si: Silicon, K: Potassium, Ca: Calcium, Ba: Barium, Fe: Iron. As the dataset's targets, type of glass: (class attribute) with 6 different classes (1 class does not have any samples); 1: building windows float processed, 2: building windows non float processed, 3: vehicle windows float processed, 4: vehicle windows non float processed (none in this database), 5: containers, 6: tableware, 7: headlamps. There are 214 instances in this dataset, the features are float numbers that represent the percentages. We selected this database to challenge our kNN implementation with multiple classes. The Labeled Examples dataset provides the binary classification, here by using a dataset with multivariate targets, we utilize our implementation for a more challenging task.

2.3 Banknote Authentication Dataset

The banknote authentication dataset is another dataset provided in UCI Machine Learning Repository dataset archives [2]. The dataset contains 4 continuous features and a binary target extracted from 400x400 banknote images. The attributes of the dataset are; variance of wavelet transformed image, skewness of wavelet transformed image, curtosis of wavelet transformed image and entropy of image.

2.4 kNN Algorithm with Cross-Validation

The k Nearest Neighbor algorithm implementation for this project required several adjustments on the traditional method. First, we implemented two different options to store training instances; store every instance or store every first mislabeled k instances for each class. Furthermore, we employed two different distance metrics; Euclidean and Manhattan. With these two modifications and the k value, we have 3 different parameters we can alternate. By varying these parameters, we can observe the effect of each parameter on accuracy.

[1] https://archive.ics.uci.edu/ml/datasets/glass+identification
[2] https://archive.ics.uci.edu/ml/datasets/banknote+authentication

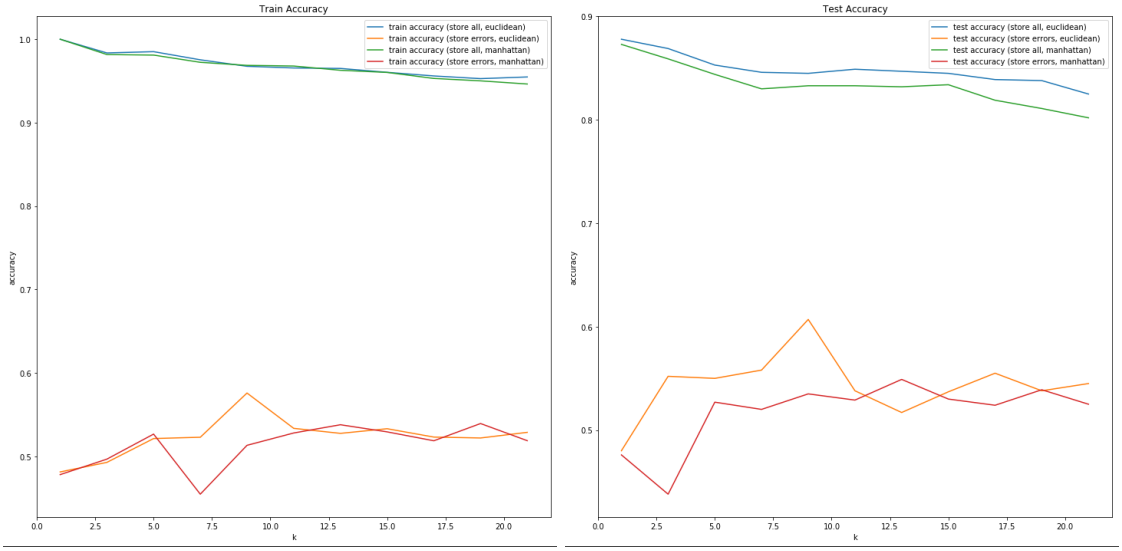
This part of the experiment requires the implementation of N-fold cross validation with given pseudo-code: With cross validation, the data is split into N equal sized blocks and the classifier is tested on these blocks

Algorithm 1 N-fold Cross-Validation on dataset, D

procedure CROSS-VALIDATION(N, D)
    Partition D into N roughly equal splits, $D_i; i = 1, ..., N$
    $Acc_{train} \leftarrow 0$, $Acc_{test} \leftarrow 0$
    for i = 1 to N **do**
        $D_{train} \leftarrow D\ D_i$, $D_{test} \leftarrow D\ D_i$
        $M_i \leftarrow trainModel(D_{train})$
        $Acc_{train} \leftarrow Acc_{train} + eval(M_i, D_{train})$
        $Acc_{test} \leftarrow Acc_{test} + eval(M_i, D_{test})$
    Return $(Acc_{train}/N, Acc_{test}/N)$

while alternating the training and test blocks. The training and test accuracy of the classifier is stored for each fold and average training and test accuracy over N iterations are returned. In this part, we implemented kNN models with 11 different k values (1,3,5,...,21), two different distance metrics and two different storing options. Further, we applied 10-fold cross validation for each different kNN model on three datasets presented above.

Labeled Examples dataset is used to train kNN model with varying parameters as our first cross validation experiment. The number of folds for the cross validation is set to 10 for all dataset experiments. Figure 1 shows training and test accuracy results for each model configuration. The x axis represents the accuracy and the y axis are the k values. Figure 1.a suggests the training accuracies are divided into two groups where models with Euclidean distance and store all option and Manhattan distance and store all option (blue and green) reached as high as 0.95 to 1 but the models that stores only errors received scores only slightly higher than 0.5, in other words a coin flip. However, we expect the accuracy to get higher for bigger k values owing to the fact that it will increase the training instance pool size. Test accuracy follows a similar trend to training accuracy, only this time the difference between models are slightly narrower. Test accuracy for kNN models with store errors option increases as the value of k gets bigger, this supports our expectations from the model's behavior. The best test accuracy is 0.878 and achieved by the model with $k = 1$, Euclidean distance and store all option configurations. The model with Manhattan distance and same remaining conditions resulted as the runner-up with 0.873 accuracy. These results suggest, this model can yield high accuracy on Labeled Examples dataset if we store every instance rather than just errors regarding other parameters. Another note to we made from the outputs is the effect of k value. Both training and test accuracy results for store all option models achieved lower accuracy as the value of k increases. However, the effects of the k value is not as clear for the models that adopt store errors option.

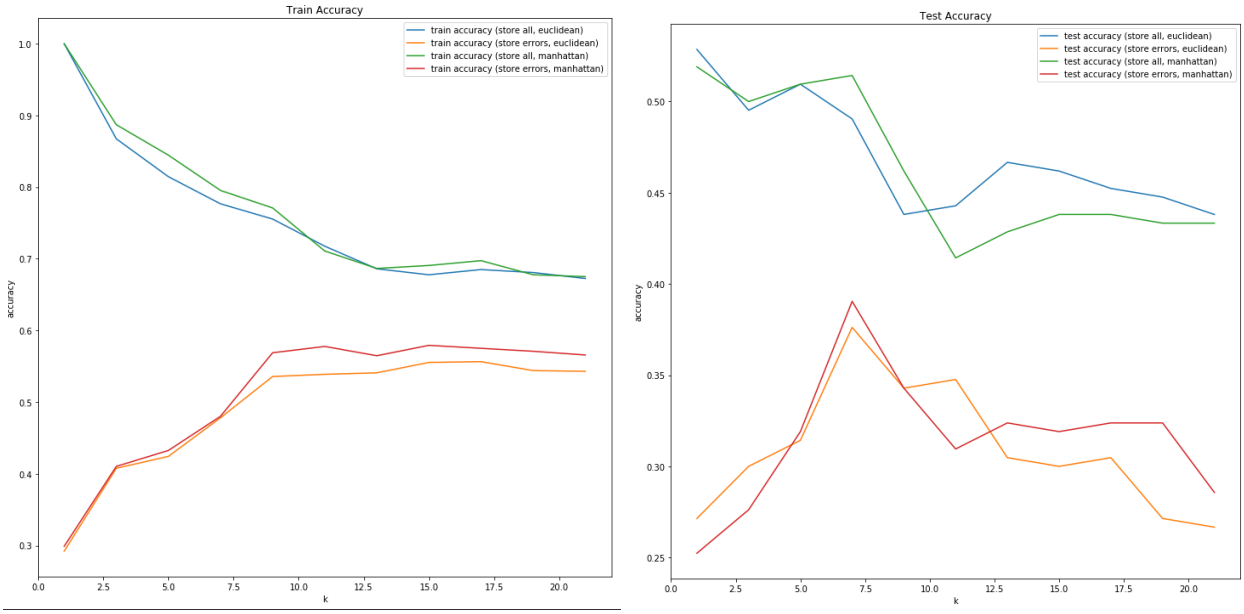


(a) Training accuracy

(b) Test accuracy

Figure 1: Labeled Examples dataset accuracy results

Glass Identification dataset is one the real life datasets we experimented with. This dataset poses a harder challenge for our kNN model due to the fact that it is multivariate unlike Labeled Examples and Banknote Authentication datasets which are univariate. The same expectations we presented above also holds true for this dataset; store errors models achieved higher accuracy as k increases and store all models (overall) received lower accuracy for bigger k values. This dataset also undercover this behavior clearly as can be seen in the Figure 2, the training accuracy for models gets closer as the k gets bigger while store all option models get lower and store errors option models get higher. The cross validation results shows that the kNN model achieved very low test accuracy for any model implemented even though there are some reasonably good training accuracy results achieved. This is possibly the effect of the insufficient training - small dataset with low number of instances since there are only 214 instances but 10 attributes. The kNN model does not achieve satisfactory results for this dataset.



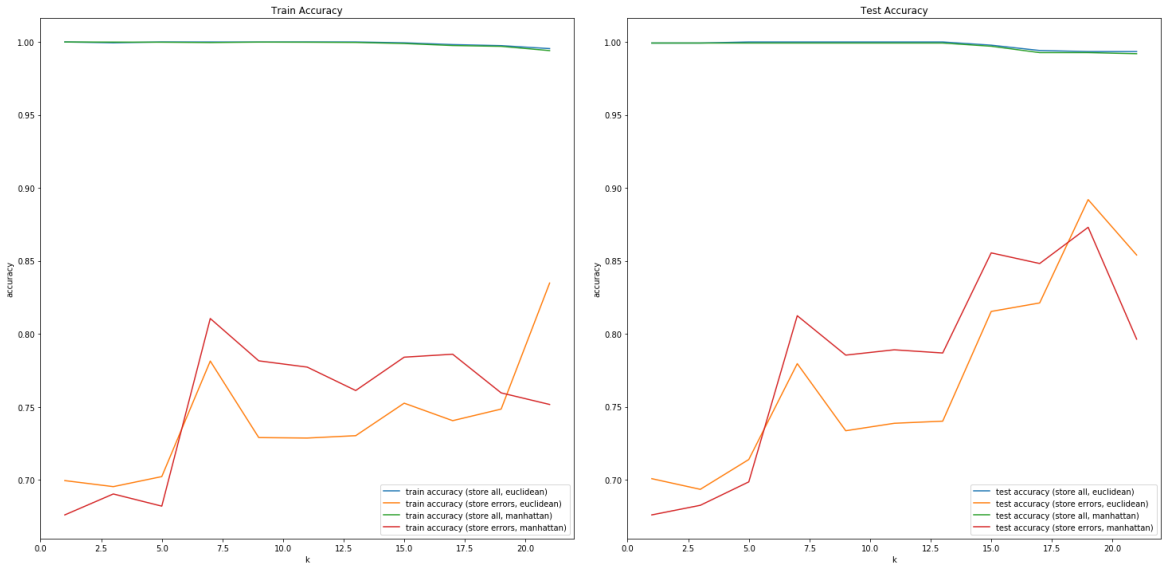
(a) Training accuracy

(b) Test accuracy

Figure 2: Glass Identification dataset accuracy results

Banknote Authentication dataset is the last cross validation experiment conducted in this project. Our expectations holds for this dataset for the error storing models as their accuracy increase with the increasing k value. The opposite effect for the store all models is almost invisible since they achieved almost perfect results on both training and test (Figure 3). The kNN algorithm achieved the highest accuracy in this dataset rather than Glass Identification and Labeled Examples datasets. Even the worst performance by kNN on the Banknote Authentication dataset achieved reasonable results; the lowest accuracy being 0.676 ($k = 1$, Manhattan distance and storing errors) and several models with perfect accuracy on the upper bound.

To summarize our findings; we cross validated numerous kNN models with varying k value, distance metric and training instances storing option. Our experiments showes that for all three dataset the accuracy difference between two training instance storing options causes the most drastic separation where store all option achieves higher scores than store errors option. This is an expected outcome since the store options only stores first k mislabeled instances for each class. For example, the banknote dataset is univariate and there are only 2 classes, for a low k value such as 3, there are only 6 instances used in training while store all option can store 900 instances in training. Furthermore, for the same reason, error storing models should achieve higher accuracy for higher values since this also increase the number of stored instances in store error option models. Experiment results support this suggestion and it can be observed through the cross validation results of all datasets (Figure 1, 2 and 3).



(a) Training accuracy

(b) Test accuracy

Figure 3: Banknote Authentication dataset accuracy results

3 Local Search Algorithms

In this section, we implement two stocasthic optimization techniques: Simulated Annealing and Genetic Algorithm and test them on several optimization problems created with numerous specific mathematical functions. The functions provided with the homework are; a basic sphere model, Griewank's, Modified Shekel's Foxholes, Michalewitz's, Langermann's, odd square and bump functions. In addition to the functions provided with the homework, we implemented four other functions that create an optimization problem; Himmelblau's, Ackley's, Holder's table and six hump camelback functions. According to the surface shape created by these functions, we adjusted our algorithms to minimize or maximize the solution.

3.1 Simulated Annealing

Our simulated annealing algorithm implementation follows the pseudo-code presented in the book from Stuart Russell, Peter Norvig - Artificial Intelligence; A Modern Approach as Figure 4.5, page 115:

Algorithm 2 Simulated Annealing Algorithm

function SIMULATED-ANNEALING(*problem*, *schedule*) returns a solution state
  current ← problem.INITIAL
  for t = 1 to ∞ **do**
    T ← *schedule*(t)
    if T = 0 **then return** *current*
    next ← a randomly selected successor of *current*
    ΔE ← VALUE(*current*) – VALUE(*next*)
    if ΔE > 0 **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{-\Delta E/T}$

There are several adjustments made in our implementation to make algorithm work with more control over it. Our implementation takes maximum and minimum values for x and y axes created with the mesh function provided with utility functions for the project to use them as boundaries. The function is also gathered as a parameter to calculate the utility values - fitness within the function. The t and T_max parameters are used used by schedule function to adjust the temperature (T), default value of t is 0 and T_max is 100000. T_max or maximum heat is a parameter to set to upper bound and t or the step count controls the lower bound for temperature (T). The argument that controls if the algorithm searches for the maximum or minimum is called maximize and by default it's value is set to True, hence if this parameter is not adjusted our algorithm will try to maximize the utility. Lastly, the verbose parameter (default is False) when set to True, the algorithm will

return the list of best solutions of each iterations in a list, this adaptation is needed to trace the results of the algorithm.

3.2 Genetic Algorithm

Our genetic algorithm implementation follows the pseudo-code presented in the book from Stuart Russell, Peter Norvig - Artificial Intelligence; A Modern Approach as Figure 4.8, page 119:

Algorithm 3 Genetic Algorithm

function GENETIC-ALGORITHM($population, fitness$) returns an individual
  repeat
    $weights \leftarrow$ WEIGHTED-BY($population, fitness$)
    $population2 \leftarrow$ empty list
    for i = 1 to SIZE($population$) **do**
      $parent1, parent2 \leftarrow$ WEIGHTED-RANDOM-CHOICES($population, weights, 2$)
      $child \leftarrow$ REPRODUCE($parent1, parent2$)
      if (small random probability) **then** $child \leftarrow$ MUTATE($child$)
      add $child$ to $population2$
    $population \leftarrow population2$
  until some individual is fit enough, or enough time has elapsed
  return the best individual in $population$, according to $fitness$

function REPRODUCE($parent1, parent2$) returns an individual
  $n \leftarrow$ LENGTH($parent1$)
  c $\leftarrow$ random number from 1 to n
  return APPEND(SUBSTRING($parent1, 1, c$), SUBSTRING($parent2, c+1, n$))

Our mutation implementation (not from the book):

function MUTATE($individual, boundaries$)
  for t = 1 to ∞ **do**
    $n \leftarrow$ LENGTH($individual$)
    i $\leftarrow$ random number from 0 to n
    Flip the ith bit on individual bitstring
    if $individual$ is within $boundaries$ **then**
      return $individual$

Just like the simulated annealing algorithm, genetic algorithm also required us to make adjustments on the function parameters. These parameters of the genetic algorithm share exactly same purposes with the simulated annealing parameters: maximize, verbose, X and Y maximum and minimum boundaries, t (step count) and T (instead of T_max). In addition to these parameters, genetic algorithm requires: population parameter created with *initial_population* function, threshold parameter to set the satisfying utility lower bound and *mp* as the mutation probability (default is 0.05).

The workflow of the genetic algorithm requires several utility functions to call each other in order to proceed, the pseudo-code presents the main algorithm, *reproduce* and *mutate* functions. In addition to these functions presented, we created: $fitness$, $individual\_fitness$, $weighted\_by$, $weighted\_random\_choices$, $initial\_population$. The $initial\_population$ function takes X and Y boundaries and number of individuals m as parameters to create a random population with m individuals represented with 64 Boolean bits. This function creates the initial population to be employed in the genetic algorithm with *population* parameter. When the genetic algorithm function is called, the first utility function needed is the $weighted\_by$ function. This function simply calculates the utility values, scales them to 0 and 1 range and assigns these values to corresponding individuals as weights, therefore higher utility scores means bigger weights and thus higher chance to be used to reproduce. Following, the genetic algorithm function creates the new generation with the *reproduce* function which represents the cross-over concept in biology and randomly merges the bitstring of two parents. Beside these functions, we created $b2f$ and $f2b$ that converts bitstring to X and Y coordinates and vice-versa,

randomb function that creates a random bitstring that represents an individual within the given boundaries and some plotting functions for convenience.

3.3 Experiment Results

This section provides detailed information about how we implemented the simulated annealing and genetic algorithms on several different optimization problems and their results. There are 7 functions presented in the utility functions file provided with this assignment, in addition to that, we implemented 4 different functions that create different challenges for our algorithms. For all of the figures presented in this project, we used red color for simulated annealing algorithm and blue color for genetic algorithm.

Sphere Function provides a rather simple bowl shape in 3 dimensions. This function has a single global minimum, a single global maximum and 3 local maxima. The maximum utility value of the Sphere function is 144 and minimum is 0. Minimization tasks are directed to a single minimum, however maximization can be harder for this graph since all the local maxima are separated on the edges of the graph. We tested simulated annealing and genetic algorithms for both maximization and minimization tasks on the Sphere Function problem with $t = 0$ for both algorithms, T_max and $T = 100000$. The boundaries are set to -5, 5 for both axes. The results are shown in Figure 4 for maximization and minimization tasks respectively. For maximization task; the simulated annealing (red) almost reached the global maximum with 143.75 where genetic algorithm (blue) ended up on the middle of the hill of global maximum with the given Temperature adjustments achieving 64.92 utility. For the minimization task, both methods achieved to reach the global minimum within their Temperature limits where simulated annealing algorithm received 0.05 and genetic algorithm received 0.25 utility. This minimization task is the simplest one amongst all optimization problem functions.

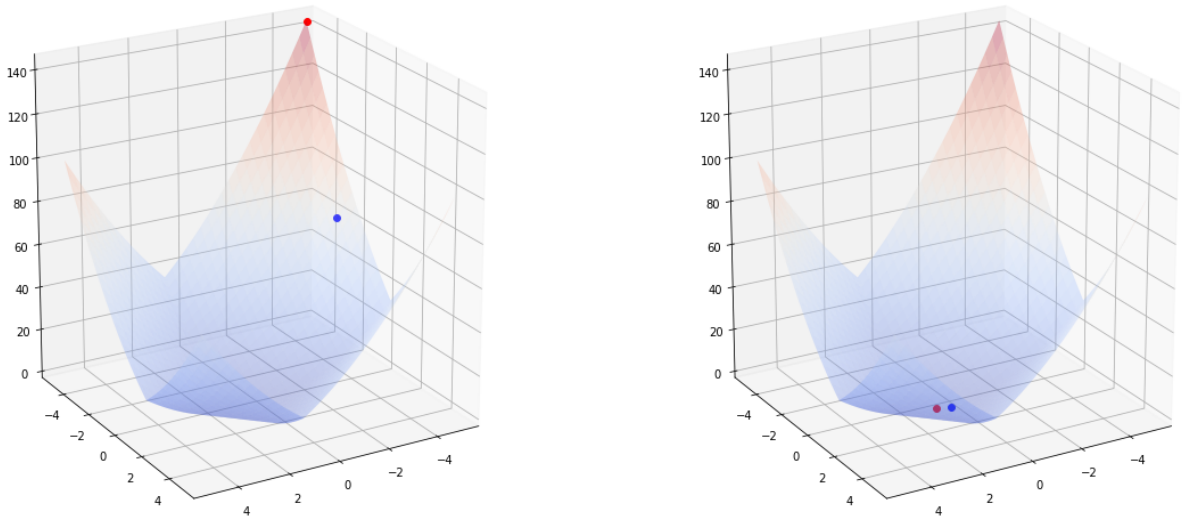


Figure 4: SA and GA maximize and minimize on Sphere

Griewank's Function is very similar to the Sphere Function but shows more noise through each point in the surface and has 4 global maxima. This information inclines the maximization and minimization tasks are expected to be slightly harder but still similar to the Sphere Function. Figure 5 shows the maximum and minimum utilities found by simulated annealing and genetic algorithms. As the figure suggests, the results are very similar to Sphere Function results; simulated annealing and genetic algorithms both achieved very close the global minimum (0) with 0.08 and 0.14 respectively. The maximization problem is covered with the simulated annealing, however the genetic algorithm function again stuck on the hill with utilities 10.27 and 4.78 respectively (global maximum is 11.41). At this point, we altered the Temperature with larger values to increase the iterations of the genetic algorithm to see if it can achieve higher utilities in maximization. When the T value is set to 1000000, the utility is increased to 5.89 (initial population is randomized for each run) and when $T = 10000000$ the algorithm achieved 6.63 utility. This suggests with the given computation time, genetic algorithm can achieve higher utilities, however, there is no guarantee that it can achieve the global maximum with limited computation power.

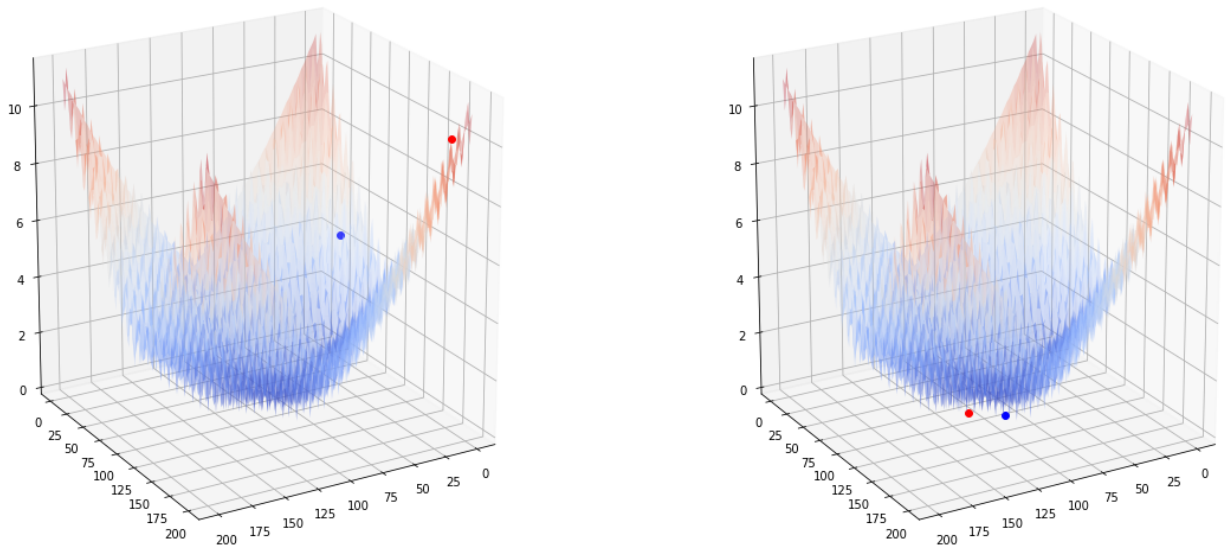


Figure 5: SA and GA maximize and minimize on Griewank's

Shekel's Function is a very obscure example of a minimization problem. There are countless local maxima, several local minima and a single global minimum with -5103.45 utility is the important features of this function. Consequently, maximization for this function is unnecessary and we only experimented on minimization task. The global minimum is very hard to reach for algorithms if they are not randomly placed near the global minimum, in addition to that, without a large random step, it seems almost impossible for our algorithms to escape from the local minima. After several attempts, we increased the Temperature for both algorithms (T and $T\_max = 1000000$) to increase the chances of finding at least a local minima if not the global minimum. With the higher temperatures, simulated annealing algorithm managed to fall in one of the local minima and achieved -693.07 utility. Unlike simulated annealing algorithm, genetic algorithm did not found any pits and resulted with -6.63 utility (Figure 6). This function posed a great challenge for both algorithms, even with the higher temperatures, both methods failed to find the global minimum.

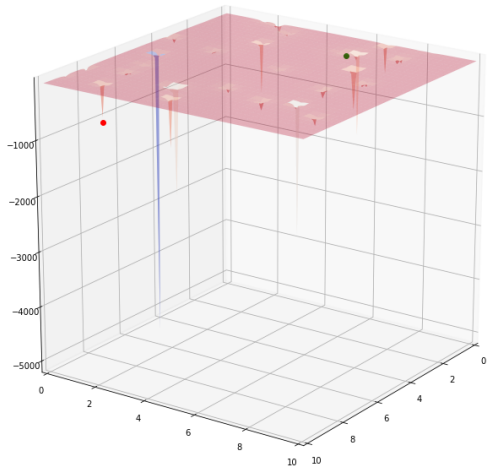


Figure 6: SA and GA minimize on Shekel's

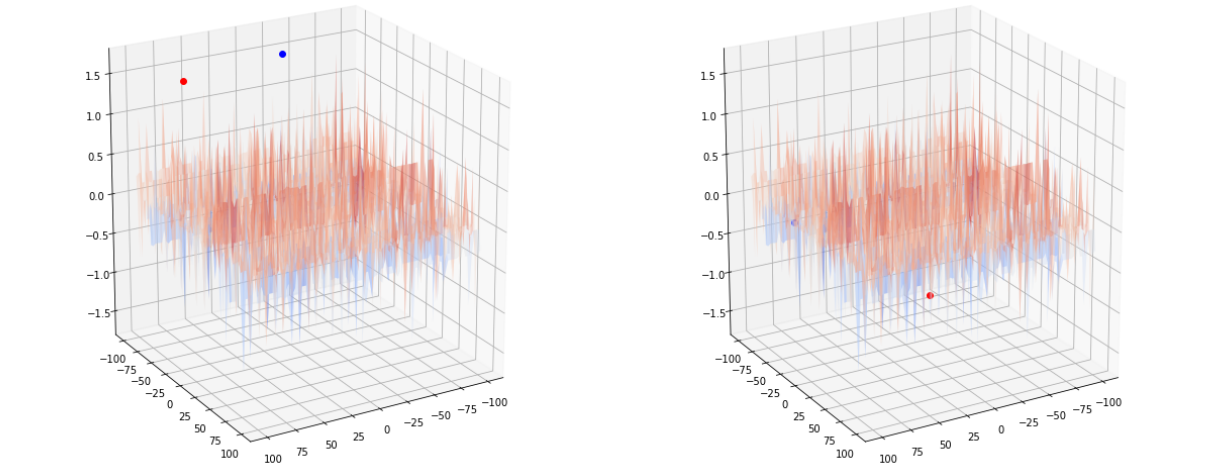


Figure 7: SA and GA maximize and minimize on Michalewitz's

Michalewitz's Function is a noisy function with numerous local minima and maxima, therefore both maximization and minimization can be tested with the challenge of noisy surface provided by the function. The bounds are -100, 100 for both axes, the maximum and minimum utilities in the function are 1,73 and -1,73 respectively. The structure of the function turns maximization and minimization problems identical, hence we implemented different temperatures for maximization and minimization with T values 1000000 and 100000 respectively. As shown in Figure 7; on the maximization task, simulated annealing achieved 1.58 and genetic algorithm achieved 1.45 utility. Both algorithms found a local maxima with a very close value to the global maximum. For minimization task, simulated annealing obtained -1.44 and genetic algorithm received -0.38 utility. These results show the genetic algorithm suffered the lower temperature more than the simulated annealing algorithm, but given enough time (higher temperature), genetic algorithm also achieves good utilities.

Langermann's Function is another noisy function, however there is a global minimum and global maximum provided and these points are located near each other. We experimented minimization for this function and achieved similar results to our previous experiments. The boundaries for Langermann's functions are 0, 10 for X and Y axis and the global minimum is -1.7. When the temperature is default (100000), simulated annealing gained -1.29 utility. As can be clearly seen from the Figure 8, simulated annealing stuck in a local minima on the edge of the X axis. The genetic algorithm got -0.52 utility and repeated the scenario of finishing on the middle of the climb due to temperature. This experiment also showed the default temperature value employed in this function (100000) is insufficient for genetic algorithms given noisy surfaces.

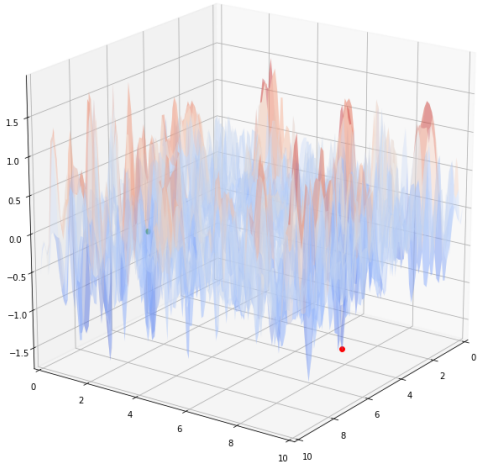


Figure 8: SA and GA min. on Langermann's

Odd Square Function has a slightly different structure from the previous functions, it has a global minimum and a global maximum placed very close to each other, surrounded with several local minima and maxima around and all covered inside a big straight plateau. The X and Y axes are bounded within -5π and 5π, the global maximum utility is 1.14 and global minimum utility is -1.02. We tested our algorithms for both maximization and minimization tasks. As Figure 9 shows, both algorithms achieved utilities close to global maximum on maximization, however, genetic algorithm got stuck on the plateau with 0 utility on the minimization task while simulated annealing received almost perfect -1.01 utility. Since the minimization and maximization is almost identical tasks for this function, this shows the importance of the start point for the genetic algorithm. With the same temperatures, genetic algorithm achieved 0.87 on maximization but 0 on minimization owing to the fact that the algorithm never managed to escape the plateau. This shows, with small step sizes (small chance of randomness), it is very important where algorithm is first started since if it's further away from the noisy part, it might not be able to find higher utilities in lower temperatures.

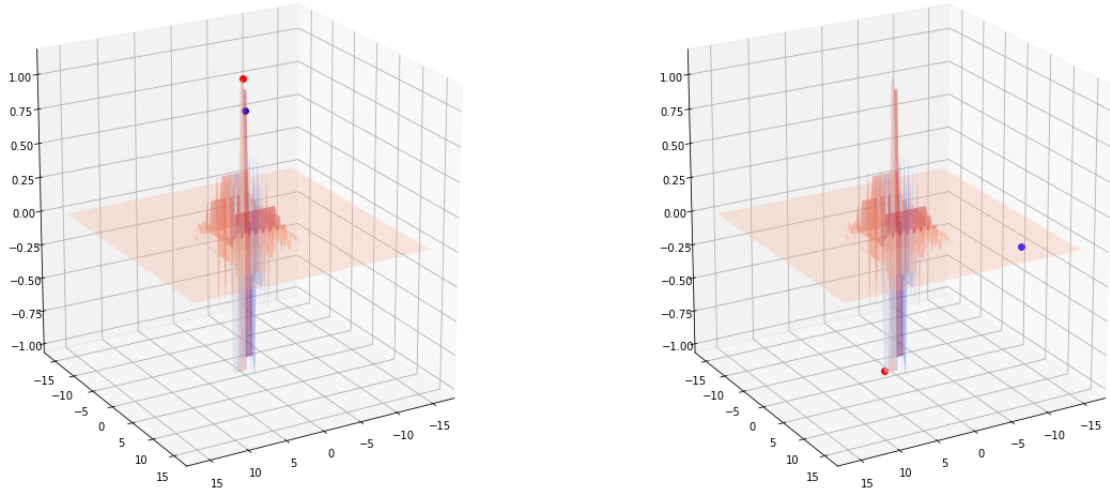


Figure 9: SA and GA maximize and minimize on Odd Square

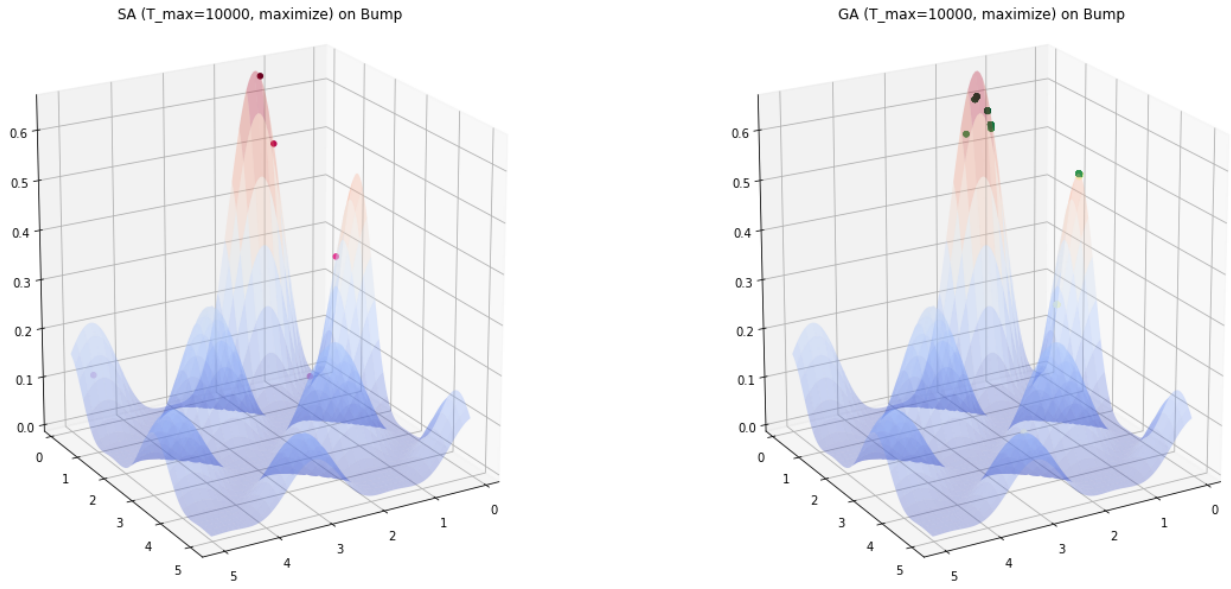
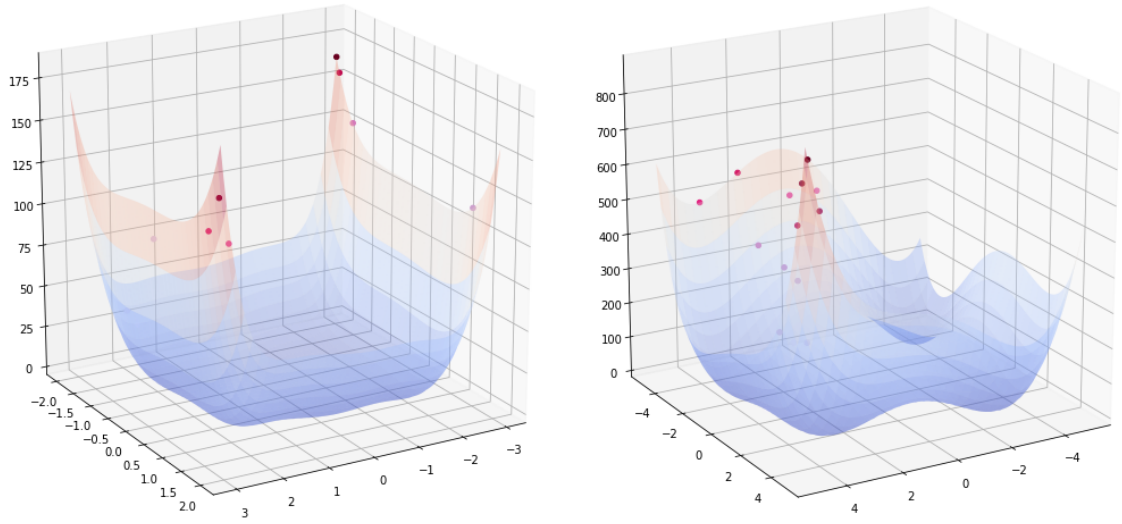


Figure 10: SA and GA maximize on Bump with tracing

Bump Function is the last function provided for experiment. The smoothness of this function can possibly provide the best visual representations of how the utility changes through time. To achieve this goal, we implemented *trace* parameter in our local search algorithms, by setting this Boolean parameter to True, the functions output all best values obtained in each iteration instead of a single value. To clarify how the algorithm results change through time, we used gradient color change (yellow to red and yellow to green) on the scatter plots, however, owing to there are many points in the same area, the nodes overlap on each other. Nevertheless, it allows us to see how the algorithm performs better though time. Figure 10 shows the simulated annealing and genetic algorithms' traces and ending points. Simulated annealing algorithm started at around (0,5) point and later landed on the local maxima around (2,1) point and finally reached the hill of global maximum and climbed it to the peak with 0.64. Genetic algorithm started on the local maximum and similar to simulated annealing, managed to jump to the global maximum with 0.61. Since the global maximum is 0.65, we can safely say both methods were successful on finding global maximum even though they both visited the local maxima.



(a) SA maximization on Six Hump Camelback (b) SA maximization on Himmelblau's

Figure 11: Simulated Annealing maximization trace

Six Hump Camelback Function is similar to the Sphere and Griewank's functions, a bowl shape with a plateau around the global minimum. The function boundaries are -3, 3 for X axis and -2, 2 for Y axis, the global maximum is 185.86 and the global minimum is -1.03. We experimented with simulated annealing algorithm function and traced the best values through the iterations. Figure 11.a shows the progress that simulated annealing algorithm followed through. It is clear from the figure that the algorithm jumped between three hills before achieving it's highest utility in given temperature with 162.9. The trace shows us the first point achieved 45.14 utility and the utility kept increasing until it reached to 162.9 when the algorithm is finished.

Himmelblau's Function is another bowl shaped function with a single global maximum, 3 local maxima and several local minima. The function is placed within -5,5 boundary for X and Y axes, the utility of global maximum is 890 and the global minimum is 0. The simulated annealing algorithm is experimented on this function with the task of maximization, as Figure 11.b shows, the algorithm moves through local maxima and finds the global maximum in given temperature of $T = 100000$.

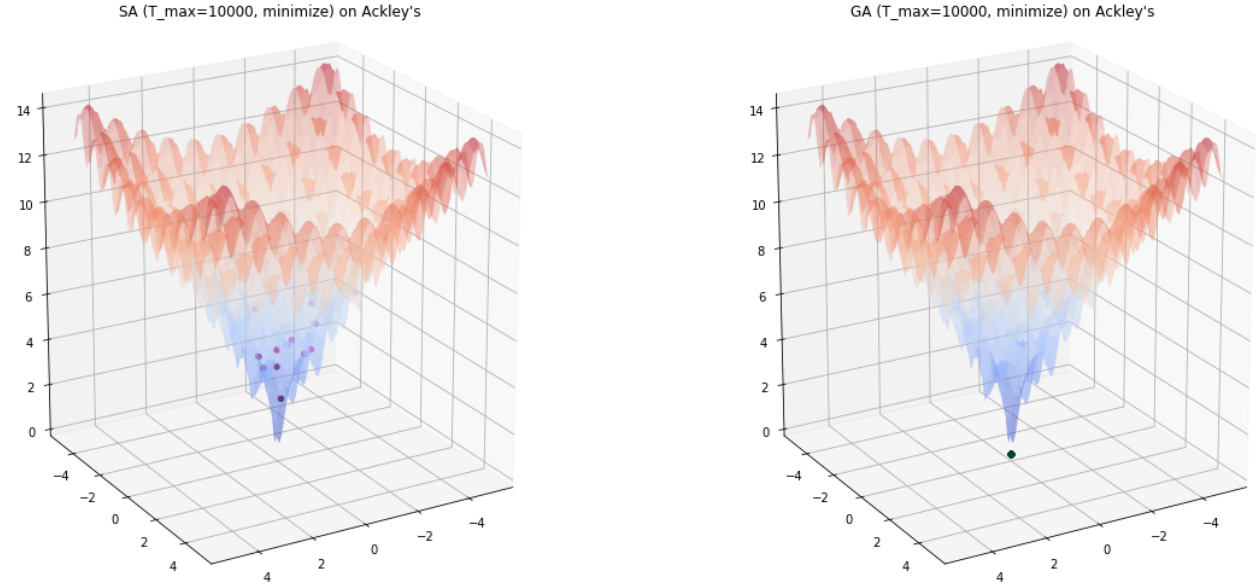


Figure 12: SA and GA minimize on Ackley's with tracing

Ackley's Function shows an upside down mountain shape with lot of noise, a global minimum at 0 and 4 local maxima with 14.03 utility placed within -5, 5 boundaries for X and Y axes. With experimenting on this function, we traced the minimization process with our local search algorithms. Figure 12 shows the steps taken by simulated annealing without an issue, however, there's a single point for the genetic algorithm. Looking at the utility values, we observed that for this case, the genetic algorithm started around 10 utility, but jumped to global minimum in an early step (41th generation).

Holder's Table Function is one of the most distinctive surfaces in this project placed inside -10, 10 boundaries for X and Y axes with 4 global minima with -19.14 utility and numerous local minima. This function provides a challenging minimization task for our local search algorithms. Figure 13 shows simulated annealing algorithm was succesful but genetic algorithm was failed to reach one of the global minima in given temperature $T = 100000$. Simulated annealing algorithm jumped in between several local minima and reached the hill for a global minima and achieved -18.9 utility while genetic algorithm also visited several local minima and ended up on the hill to a global minima and achieved -14.02 utility. This strongly suggest that genetic algorithm also could've reached to a global minima if the temperature was higher.

In this section, we implemented two local search algorithms; simulated annealing and genetic algorithms with given pseudo-codes. These search algorithms later employed to either minimize or maximize the utility with a given function. To test our implementations, we used 7 provided functions and adapted 4 other functions. Throughout our experiments we set the temperature T and T_max to 100000. Our experiments showed that with this temperature, simulated annealing method achieved better results than the genetic algorithm. We also

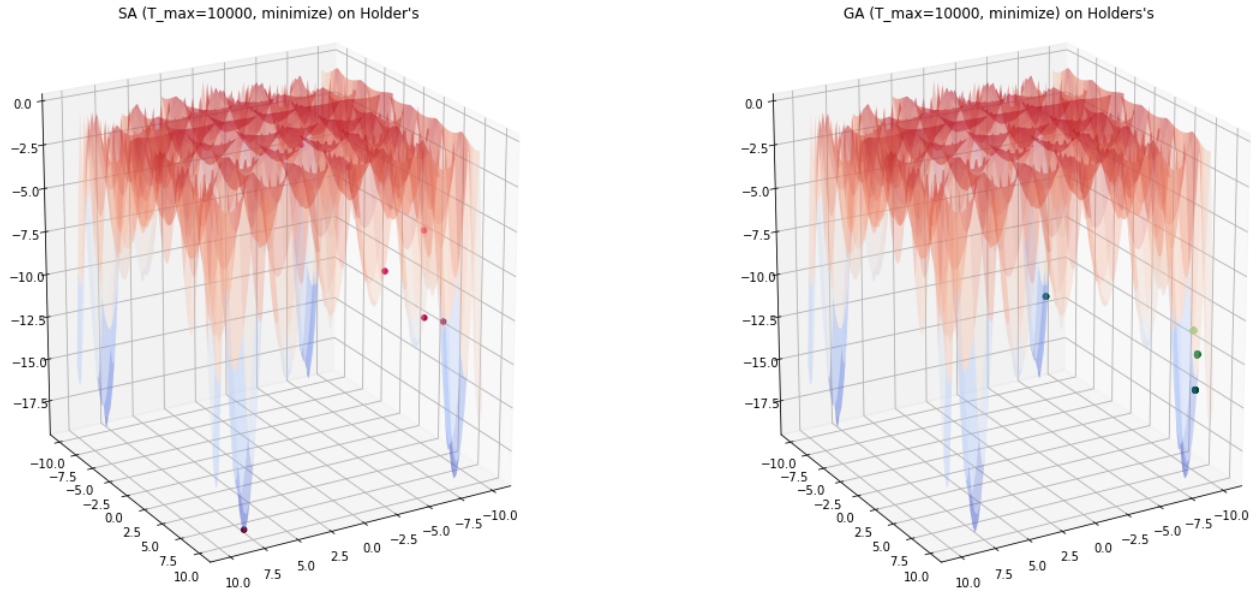


Figure 13: SA and GA minimize on Holders's with tracing

altered the temperature values for different cases and showed that the genetic algorithm also improves if given enough time (higher temperatures). Both algorithms work according to their purposes but genetic algorithm need more search to reach the goal. For almost every task, we achieved either the perfect utility value or some value close to that. To provide a better insight on our algorithms, we utilized tracing as a parameter and showed the movement of each algorithm on 3 dimensional surface plots.

The code file for this project is also provided here:
https://gist.github.com/skaraoglu/2f3f0e8aa3be458709d613f06479191b