**מעבדה 2 חלק א**

**סאמר חרעובה – 209050202**

**עבד אלרחמן אבו חוסין – 208517631**

**סעיף 1:**

implemented in fitness selector

Text

Description automatically generated

Implementation

**Text

Description automatically generated**

self.gene\_dist is the distance function that we get from fitness type class , so that we can use it in the algorithm as we please

Text

Description automatically generated

**סעיף 2:**

previously the ranking system was done in each selection function , now we changed it to be done on the population and just send a probabilities array to the selection function

selection pressure:

* in case of RWS and SUS and Tournment: we use the probability of selection that exists in fitness array(it’s a probability array of each individual) to find the probability of selecting best individual and the average individual

Text

Description automatically generated

**Diversity :**

* to calculate diversity we calculate the distance of each individual with the full population , then average it out and store it
* then add all individual diversities and average them out to indicate the diversity of the whole population

**Text

Description automatically generated**

**סעיף 3:**

**תוצאות :**

|  |  |  |  |
| --- | --- | --- | --- |
| **population size** | **exploration to exploitaion** | **Hyper triggered** | **mutation parameter=0.25** |
| **100** | **generation:33 ,runtime: 31** | **generations:28 ,runtime:32** | **generation:35 ,runtime: 35.7** |
| **300** | **generation:21 ,runtime: 209** | **generations:19 ,runtime:182** | **generation:20 ,runtime: 195** |

**As we have seen in previous lab ,our perfect mutation rate was 0.25 so we used it , as we have tried lower or higher , and it wasn’t optimal**

**Results :**

* **when the population is small both Hyper triggered and linear exploration to exploitation both work better than a fixed mutation rate.**
* **When the population is bigger:**

1. **we can see that the linear exploration to exploitation is worst than both , but this result does not reflect the full truth , as in bigger populations our linear function should be adjusted to the appropriate ratio between population size and the linear value that we add to the original mating probability**
2. **Hyper triggered is faster than a fixed mutation rate. And gets out of local optima much faster**

מעבר הדרגתי מ exploration ל exploitation :

Text

Description automatically generatedPopulation size=100

**Text

Description automatically generated**Population size=300

Hyper triggered:

Population size=100:  
Text

Description automatically generated

Population size=300

Text

Description automatically generated

Fixed mutation parameter:

Mutation rate=0.25:

Population size=100:

Text

Description automatically generated

Population size=300:

Text

Description automatically generated

**סעיף 4:**

Implementation of GA with speciation:  
Text

Description automatically generated

Explanation of each parameter :

1. Groups is an array that has k clusters where k is defined from either threshold speciation or k-means clustering .
2. distance\_factor : is a minimal distance metric between two identical genes that one of them was mutated.
3. Speciation : a function that returns clusters in an array .
4. SpeciationType : dictates which algorithm to use for speciation ,i.e. threshold speciation or k-means clustering
5. Self.threshold: is used either as k or as the current threshold based on the speciationType

Explanation of each function :

1. how\_many\_of\_each\_speicy(self,size)

* specifies the number and positions of the elite member in each cluster/specie

1. def mate(self, gen):

* treats each specie as a singl population and mates only the ones that are in the cluster/specie
* after going over all cluster , this function puts all genes of all kind of species in the buffer as separate genes.(not as clusters)

Text

Description automatically generated

1. def init\_population(self)

* this function initiates the population as it was done in previous implementations and calculates the distance\_factor or k so that we can use those parameters in the speciation itself

1. def algo(self, i):

* This is the algorithm itself:
  + First calculate diversity for all population
  + Sort them by fitness
  + Change current threshold based on previous iteration
  + Rank population for selection methods
  + Use the new mate function to mate the population
  + Then new hash is created for distance measurements

Text

Description automatically generatedImplementation of speciation types:

1. def spicieation\_threshold

* returns the new threshold

1. def Minimal\_distance(gene, distance\_function, prob\_spec)

* returns the minimal distance to reduce or add to threshold

1. def threshhold\_speciation(self, hash, thresh, pop):

* retuns the new clusters
* basically starts with the first gene checks if the threshold condition applies then adds member to a cluster
* after insertion of first 2 genes in a specific cluster, the distance check is used on all members of the specific cluster
* continues to add clusters and members until it finishes going over the full population

1. def optimal\_k\_clustering(self, hash, k, pop):

* starts with k=3 up to maximum k
* uses k-means clustering until silhouette\_score retuns True , i.e. until k is optimal

1. def k\_means\_clustering(self, hash, k, pop):

* Creates iterations of sub\_k\_means until population converges ,i.e. until centroids do not change from previous iteration

1. def sub\_k\_means(self, hash, pop, centroids, clusters, clusters\_means, distances):

* Appoints genes to centroids based on minimal distance from centroid
* Returns new clusters with mean of distances in each cluster

Comparison between Threshold Speciation and k-means Clustering Speciation :

Runs on strings matching (bul pgiaa):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Threshold | | | K-means clustering |
| population size | 5 % of poulation size | 15% of population | 30 |  |
| 100 | generation:30 runtime: 34.5 | generations:34 runtime:30 | generation:90 runtime: 90 | generation:28 runtime: 25 |
| 200 | generation:30 runtime: 118 | generations:29 runtime:129 | generation:33 runtime: 143.5 | generation:27 runtime: 111 |
| 512 | generation:17 runtime: 502 | generations:23 runtime:637 | generation:17 runtime: 492 | generation:11 runtime: 340 |

There are 2 main metrics to consider when comparing the two algorithms

1. Time to get to the solution
2. The quality of the solution

The results above are quite clear:

1. The time to get to the solution in k means is a lot faster than threshold speciation ,although in small populations it doesn’t amount to much difference but in larger populations we can see a significant time reduction /
2. The quality of the solution on both algorithms is outstanding ,as we see in the mock trials below of the code running :

* On both we can see that the initial mating of the species produces better solutions, we can see that in the fitness value and mean of the fitness value in both algorithms
* But we can see that k means clustering handles local minima much better than threshold speciation as it takes 3 to 4 iterations on average for threshold speciation to get out of local minima, yet K-means hardly stumbles on local minima

Threshold=5 , population size=100

Text

Description automatically generated

Text

Description automatically generated

Threshold=10 , population size=200

Text

Description automatically generated

Text

Description automatically generated

Threshold=25 , population size=512

Text

Description automatically generated

Text

Description automatically generated

Threshold=30 , population size=100

Text

Description automatically generated

Threshold=30 , population size=200

Text

Description automatically generated

Threshold=32 , population size=200

Text

Description automatically generated

Threshold=76 , population size=512

Text

Description automatically generated

K-means clustering with optimal k each iteration:

Population size =200

Text

Description automatically generated

Population size=512

Text

Description automatically generated

Text

Description automatically generated

**סעיף 5:**

**תוצאות כלליות בשביל ניתוח סופי :**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **population size** | | **exploration to exploitaion** | | **Hyper triggered** | | **mutation parameter=0.25** | |
| **100** | | **generation:33 ,runtime: 31** | | **generations:28 ,runtime:32** | | **generation:35 ,runtime: 35.7** | |
| **300** | | **generation:21 ,runtime: 209** | | **generations:19 ,runtime:182** | | **generation:20 ,runtime: 195** | |
|  | Threshold | | | | | | K-means clustering |
| population size | 5 % of poulation size | | 15% of population | | 30 | |  |
| 100 | generation:30 runtime: 34.5 | | generations:34 runtime:30 | | generation:90 runtime: 90 | | generation:28 runtime: 25 |
| 200 | generation:30 runtime: 118 | | generations:29 runtime:129 | | generation:33 runtime: 143.5 | | generation:27 runtime: 111 |
| 300 |  | |  | |  | | Generations:16 runtime=163 |
| 512 | generation:17 runtime: 502 | | generations:23 runtime:637 | | generation:17 runtime: 492 | | generation:11 runtime: 340 |

**k-means-clustering + population 300:**

**Text

Description automatically generated**

**סעיף 6+7:**

מכיוון שלא הצלחנו לשחזר את random immigrants שאיבדנו עם רוב חלק ב אזי נשווה את הפתרון האופטימלי שקיבלנו מ k-means clustering :

1. שלימות : האלגוריתם עם k means תמיד מביא לפתרון סופי , כיוון שהוא תמיד יוצא מ local optima ובכל האיטירציות שהרצנו , למראת שגם כן האלגוריתם הקודם תמיד היה מוציא פתרון , אך לא היה מוגבל מבחינת ריצות ואז היה רץ עד max iterations אך בסופו של דבר שניהן מגיעות לפתרון גם אם לא היה האופטימלי.
2. מכיוון ש k-means יוצא מ local optima מצאנו שבכל המקרים שעברנו עליהן מול האלגוריתם הישן היה תמיד מוצא את הפתרון האופטימלי אך בפחות צעדים , ספציפית עם אוכלוסייה קטנה כמו 100 האלגוריתם הישן לא היה תמיד מגיע לפתרון למראת ה k-means.
3. כמו שאמרנו בסעיפים קודמים , הפתרונות הן איכותיות יותר למשל ב k-means עם אוכלוסייה קטנה size=100 ,הגענו לפתרון תוך 28 generations למראת האלגוריתם הישן שהגיע תוך 35 generations שהיה עם mutation rate=0.25 . (כל התוצאות הללו נמצאות בטבלות בסעיפים הקודמים ) ,גם כן נראה שבאוכלוסיות יותר גדולות לאלגוריתם החדש לוקח 16 generations למראת הישן שלוקח 20 generation .   
   ד. מבחינת זמן אין הרבה מה להשוות כיוון שכל אטירציה שיש בה חישוב של diversity של כל האוכלוסייה לוקחת יותר זמן מהאלגוריתם הישן כדי למצוא תשובה .   
   הכוונה שהאלגוריתם של המעבדה קודמת היה מסיים בפחות משניה לפעמים למרות האלגוריתם החדש שלוקח לא פחות מ 5 שניות במקרה הטוב ביותר .

שיפורים אפשריים בעתיד:

1. Distance function ליניארי , או לפחות יותר טוב ממה שיש לנו עכשיו , אולי שימוש ב hamming distance או Euclidian distance היו אופציות יותר הגיוניות .
2. אלגוריתם random immigrants והאלגוריתם הממטי היה מדהים ומעניין , אך לא הצלחנו לשחזר אותו ,לכך עלינו להיות יותר זהרים בעתיד.
3. למראת שכל הפונקציות כלליות עלינו לעשות אותם מודולריות בעתיד.

חלק ב :

כמו ששלחנו קודם בבקשה לאיחור ,לא הספקנו לשחזר אותו .