# Project Υπολογιστικής Νοημοσύνης
# Μέρος Β

Κωνσταντίνος Βασιλαδιώτης 1088094

## Β1. Σχεδιασμός ΓΑ

### α) Κωδικοποίηση:

Για την αναπαράσταση των δεδομένων μου χρησιμοποίησα την λογική του πρώτου μέρους όπως συνίσταται δηλαδή χρήση BoW με tf-idf μορφή χρησιμοποιώντας έναν tfidfVectorizer για την μετατροπή των δεδομένων του πεδίου text σε διανύσματα tf-idf.

```
# Ανάκτηση του πεδίου text για τις επιγραφές
X_text_tfidf = filtered_data['text'].tolist()

# Αρχικοποίηση του Vectorizer με tf-idf
vectorizer = TfidfVectorizer(max_features=1678)

# Μετατροπή του κειμένου σε διανύσματα tf-idf
X = vectorizer.fit_transform(X_text_tfidf)

# Υπολογισμός του μεγέθους του λεξικού
vocabulary_size = X.shape[1]

# Ανάκτηση λεξικού token σε μια μεταβλητη
vocabulary = vectorizer.get_feature_names_out()
```

Με αυτό τον τρόπο έφτιαξα ένα λεξικό μεγέθους 1675 tokens (Δεν γνωρίζω γιατί λείπουν 3 από τα 1678 που λέει η εκφώνηση). Επίσης στην αναζήτησή μου έλαβα υπόψη και τις δυο παραμέτρους (greater Syria and the east , id = 1693).

```
# Φιλτράρουμε τις επιγραφές που έχουν το ίδιο region_main_id ή region_main == 'Greater Syria and the East'
region_id = 1693
filtered_data = data[(data['region_main_id'] == region_id) | (data['region_main'] == 'Greater Syria and the East')]
```

Όσον αφορά την κωδικοποίηση των ατόμων του πληθυσμού θεωρούμε τα εξής: 1 άτομο = 2 λέξεις του λεξικού , όπου κάθε λέξη είναι ένας ακέραιος στο διάστημα [1, 1678]. Επομένως 1 άτομο = 2 ακέραιοι του διαστήματος [1, 1678]. Με βάση αυτά επιλέγω κωδικοποίηση ακεραίων οπού θα αντιστοιχήσω τις λέξεις σε ακέραιους και αντίστροφα διότι θα χρειαστεί αργότερα.

```
# Δημιουργία αντιστοίχησης λέξεων σε ακέραιους και αντίστροφα
word_to_int = {word: i for i, word in enumerate(vocabulary)}
int_to_word = {i: word for i, word in enumerate(vocabulary)}
```

**β) Πλεονάζουσες τιμές:**

Υπάρχει η πιθανότητα να προκύψουν πλεονάζουσες τιμές, βεβαία νομίζω λόγω ακέραιας κωδικοποίησης δεν είναι απόλυτα εύκολο, παρόλαυτα για να αποφύγουμε αυτό το πρόβλημα σε περίπτωση που υπάρξει πρόσθεσα στον κώδικα έναν έλεγχο στο λεξικό ώστε να μην μπορεί να ξεπεράσει τα 1678 tokens. Με αυτό τον τρόπο αν το μέγεθος πάει να ξεπεράσει το 1678 το λεξικό σταματά εκεί χωρίς να μεγαλώνει άλλο. Επομένως το μέγεθός του θα είναι πάντα <= 1678.

```
# Ενημέρωση του λεξικού ώστε να περιέχει μόνο τα πρώτα 1678 tokens
if len(vocabulary) > 1678:
    vocabulary = vocabulary[:1678]
```

**γ) Αρχικός πληθυσμός:**

Ανεξαρτήτως των πειραμάτων που θα γίνουν στην συνέχεια εγώ προσωπικά θα επέλεγα ένα μέγεθος αρχικού πληθυσμού population_size = 100 η κοντά στο 100. Αυτό διότι ένας μεγάλος αρχικός πληθυσμός, για παράδειγμα 200, έχει υψηλή ποικιλομορφία δηλαδή πολλά δείγματα το οποίο είναι πολύ καλό, όμως από την άλλη χρειάζεται πολύς χρόνος για την εκτέλεση του αλγορίθμου κάτι το οποίο μπορεί να προκαλέσει προβλήματα. Από την άλλη πλευρά ένας αρκετά μικρός αρχικός πληθυσμός χάνει πολύ γρήγορα την ποικιλομορφία του ακόμα και πριν βρει μια καλή λύση. Αυτό συμβαίνει διότι δεν έχει αρκετά δείγματα. Γιαυτούς του λόγους θα επέλεγα μία μέση λύση κοντά στο 100, καθώς κάθε άτομο αναπαριστά μία κλάση λύσεων οπότε δεν χρειάζεται να είναι πολύ μεγάλος ο αρχικός πληθυσμός.

**δ) Υπολογισμός ομοιότητας:**

Σε αυτό το ερώτημα δούλεψα ως εξής, αρχικά ο κώδικάς μου (δηλαδή ο αλγόριθμος) πέρα από τις γνωστές παραμέτρους ( crossover_rate, mutation_rate κτλπ ) περιλαμβάνει και ως input ένα target_text το οποίο εμείς διαλέγουμε ποιο θα είναι κάθε φορά ανάλογα με το προς τα που θέλουμε να κατευθύνονται τα άτομα. Για αρχή λοιπόν βάζω ως είσοδο στον κώδικα μου την ζητούμενη επιγραφή «αλεξανδρε ουδις». Βάση αυτής της εισόδου θα βρούμε τις top-5 πιο κοντινές της σύμφωνα με την απόσταση Manhattan που χρησιμοποίησα εγώ.

```
def manhattan_distance(text1, text2):
    distance = sum(abs(ord(c1) - ord(c2)) for c1, c2 in zip(text1, text2))
    return distance

def find_top_k_closest(target_text, k=5):
    distances = [(index, manhattan_distance(target_text, text)) for index, text in enumerate(X_text_tfidf)]
    sorted_distances = sorted(distances, key=lambda x: x[1])
    top_k = sorted_distances[:k]
    return top_k
```

Επίσης η αναζήτηση μας γίνεται βάση του target_text = «αλεξανδρε ουδις» όπως είπαμε:

```
target_text = "αλεξανδρε ουδις"

num_experiments = 12
max_generations = 50
population_size = 20
crossover_rate = 0.6
mutation_rate = 0.1
```

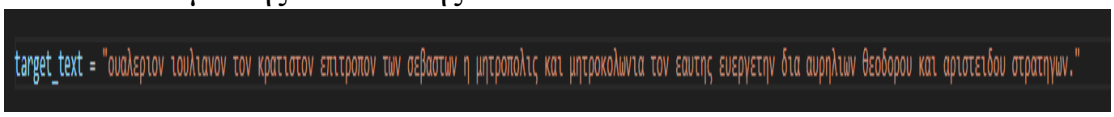
Βάση αυτών των δεδομένων τα αποτελέσματα που παίρνουμε μοιάζουν κάπως έτσι:

```
Top 5 Closest Texts:
Manhattan Distance: 94, Text: ουαλεριον ιουλιανον τον κρατιστον επιτροπον των σεβαστων η μητροπολις και μητροκολωνια τον εαυτης ευεργετην δια αυρηλιων θεοδορου και αριστειδου στρατηγων.
Manhattan Distance: 99, Text: ευξομενος δομνος υιος ζαχεου προσηνενκεν τω αγιω σεργιω χωριω β[ε]νμισωνα. πρεσβυτερος κυριακος υιος δομνου τω αγιω σεργιω επι ζηνωνος πρεσβυτερου.
Manhattan Distance: 110, Text: δαμασσιος σσαιδου επσαιςεν.
Manhattan Distance: 940, Text: επιφανους φιλ[ομητορος καλλινικου εν τοπωι] καθιερ[ωμε]ν[ωι δαιμονιν προγονικοις ονεγρα]ψεν εις χρο[νον ασυλιαι στηλαι νομους ιερους αρ]σαμειαν τα[υτην πολιν προς αμη[ς εκτισεν --].
Manhattan Distance: 1004, Text: εκυμεθη ο τρισμακαριος ιακωβος ο πρεσβυτερος μηνι υπερβερεταιου ετους υκε.
```

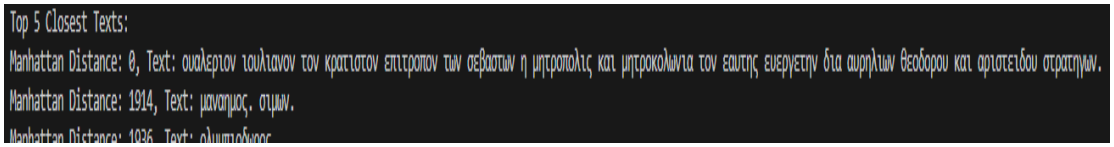
Όπου στα αριστερά έχει υπολογιστεί η απόσταση Manhattan και στα δεξιά φαίνεται το πιο κοντινό κείμενο που εντοπίστηκε. Γενικά η απόσταση Manhattan μετρά το σύνολο μονάδων που απαιτούνται ώστε

να πάμε από ένα σημείο σε ένα άλλο. Εδώ πιο συγκεκριμένα μετρά τον αριθμό διαφορετικών χαρακτήρων που πρέπει να αλλάξουν από την target_text επιγραφή για να γίνει ίδια με τις υπόλοιπες. Οι top-5 λοιπόν με ταξινομημένη σειρά εδώ είναι και οι πιο κοντινές. Άρα μεγαλύτερη Manhattan = τα κείμενα διαφέρουν περισσότερο.

Για παράδειγμα άμα βάλω ως target_text μια επιγραφή όπως είναι ( ασπούμε ότι χρησιμοποιώ την top-1 = «ουαλεριον ιουλιανον τον κρατιστον επιτροπον των σεβαστων η μητροπολις και μητροκολωνια τον εαυτης ευεργετην δια αυρηλιων θεοδορου και αριστειδου στρατηγων.» ) το αποτέλεσμα της απόστασης είναι 0.




Αποτέλεσμα:




**ε) Συνάρτηση καταλληλότητας:**

Ένα άτομο, δηλαδή μια επιγραφή, είναι πιο κατάλληλη από τις άλλες όταν είναι πιο κοντά σε ομοιότητα στις top-5 , top-10. Για να συμπληρώσω λοιπόν την επιγραφή που ζητείται και να βρω το πιο κατάλληλο άτομο έκανα το εξής: Όταν έβαλα ως target_text το «αλεξανδρε ουδις» βρήκα μέσω της απόστασης Manhattan ότι το πιο κοντινό παρεμφερές κείμενο στην ζητούμενη επιγραφή μας είναι το «ουαλεριον ιουλιανον τον κρατιστον επιτροπον των σεβαστων η μητροπολις και μητροκολωνια τον εαυτης ευεργετην δια αυρηλιων θεοδορου και αριστειδου στρατηγων.» βάση του ότι έχει την μικρότερη απόσταση Manhattan. Άρα λοιπόν οι λέξεις που αναζητούμε και το καλύτερο άτομο θα αναζητηθούν με βάση την top-1 κοντινή επιγραφή στο «αλεξανδρε ουδις». Επομένως από δω και στο εξής το target_text μας είναι η top-1 κοντινή επιγραφή. Όσον αφορά την εύρεση της απόδοσης κάθε ατόμου τώρα χρησιμοποίησα το συνολικό άθροισμα της ομοιότητας συνημιτόνου μεταξύ του target_text και μεταξύ του κειμένου που κατασκευαζεται. Αυτό σημαίνει ότι η μέγιστη τιμή της συνάρτησης καταλληλότητας είναι 1.0 το οποίο σημαίνει ότι η επιγραφή που

κατασκευάζεται είναι ακριβώς ίδια με το target_text που βάλαμε ως input. Αντίστοιχα η ελάχιστη τιμή της συνάρτησης είναι 0.0 που δηλώνει μηδενική ομοιότητα του ατόμου συγκριτικά με το target_text.

```
def fitness(individual, target_vector):
    total_similarity = 0
    for _ in range(len(individual)):
        individual_text = ' '.join(int_to_word[i] for i in individual)
        individual_vector = vectorizer.transform([individual_text])
        similarity = cosine_similarity(target_vector, individual_vector)
        total_similarity += similarity[0][0]
    return total_similarity
```

Και ένα παράδειγμα κειμένου που κατασκευάστηκε με υπολογισμό απόδοσης:

```
Restored Text: επι αυρηλιων
Best Fitness: 0.42356186426365905
Generations: 50
```

Συγκριτικά πάντα με το target_text που εισάγαμε.

**στ) Γενετικοί τελεστές:**

i) Επιλογή και χρήση ρουλέτας με βάση το κόστος λειτουργεί με βάση την απόδοση του κάθε ατόμου ως μετρητή και τείνει στο να επιλέγει το 1ο άτομο όταν το πρώτο έχει μεγάλη απόσταση με το τελευταίο. Επιλογή και χρήση ρουλέτας με βάση την κατάταξη, βαραίνει όλα τα άτομα με τον ίδιο τρόπο χωρίς να λαμβάνει υπόψη την απόσταση. Γενικά η επιλογή της ρουλέτας είναι απλή στην κατανόηση και στην εφαρμογή, όμως ευνοεί υπερβολικά τα άτομα του πληθυσμού με πολύ υψηλή απόδοση κάτι που μπορεί να οδηγήσει σε πρόωρη σύγκλιση. Τέλος υπάρχει η επιλογή τουρνουά την οποία χρησιμοποίησα εγώ. Σε αυτή την επιλογή δημιουργείται ένα τυχαίο δείγμα ατόμων από τον πληθυσμό μεγέθους k το οποίο εμείς το ορίζουμε και είναι το μέγεθος του τουρνουά. Στην συνέχεια τα άτομα στο τουρνουά συγκρίνονται μεταξύ τους και το άτομο με την μεγαλύτερη απόδοση επιλέγεται ως νικητής του τουρνουά. Η διαδικασία επαναλαμβάνεται μέχρι να επιλεχθούν τα απαραίτητα άτομα για την επόμενη γενιά. Η επιλογή τουρνουά έχει το μειονέκτημα ότι απαιτεί πολλές συγκρίσεις, όμως από την άλλη αποφεύγει την υπερβολική εύνοια στα καλύτερα άτομα.

ii) Διασταύρωση Μονού σημείου, την οποία και χρησιμοποίησα στον κώδικά μου, όπου επιλέγεται ένα τυχαίο σημείο για διασταύρωση κάτι που καθιστά αρκετά απλοική και κατανοητή την εφαρμογή αυτής της τεχνικής. Επίσης σε κάποιες περιπτώσεις μπορεί να είναι χρήσιμη καθώς διατηρεί μεγάλες ακολουθίες γονιδίων ανέπαφες, ειδικά όταν αυτές είναι αποδεδειγμένα επιτυχημένες. Το αρνητικό είναι ότι η εξερεύνηση που εφαρμόζει αυτή η τεχνική είναι περιορισμένη αφού αλλάζει μόνο ένα σημείο. Έπειτα έχουμε την διασταύρωση πολλαπλού σημείου όπου προφανώς εφαρμόζει μια καλύτερη εξερεύνηση σε έναν ευρύτερο χώρο αναζητήσεων. Το αρνητικό είναι ότι είναι πιο περίπλοκη τεχνική στην εφαρμογή της και επιπλέον υπάρχει κίνδυνος να χαλάσουν επιτυχημένες ακολουθίες γονιδίων που ήταν ωφέλιμες όπως αναφέραμε προηγουμένως. Τέλος η ομοιόμορφη διασταύρωση έχει επίσης έναν υψηλό χώρο εξερεύνησης κάτι που είναι καλό, όμως από την άλλη πλευρά αυτή η τεχνική περιλαμβάνει υπερβολική τύχη στις διασταυρώσεις κάθε σημείου κάτι που μπορεί να έχει ως αποτέλεσμα την δημιουργία απογόνων με χαμηλή καταλληλότητα.

iii) Μετάλλαξη και χρήση ελιτισμού. Προσωπικά χρησιμοποίησα την τεχνική του ελιτισμού κάτι που εξασφαλίζει ότι οι καλύτερες λύσεις(άτομα) της τρέχουσας γενιάς περνούν αμετάβλητες στην επόμενη. Αυτός ο τρόπος εξασφαλίζει ότι ο πληθυσμός θα βελτιώνεται σταθερά, άρα και ο αλγόριθμος συγκλίνει ταχύτερα προς μια βέλτιστη λύση.

```
def mutate(individual, mutation_rate):
    for i in range(len(individual)):
      if individual != best_individual:
        if randint(0, 100) < mutation_rate * 100:
            individual[i] = np.random.choice(len(vocabulary))
    return individual
```

## B2. Υλοποίηση ΓΑ:

Σε αυτό το ερώτημα παραθέτω το link ως προς τον κώδικα python για την υλοποίηση του σχεδιασμού του ΓΑ που σχεδιάσαμε νωρίτερα στην σελίδα μου στο github.

Link:
https://github.com/kostisvass/NEURAL_NETWORKS_PART_A.git
και ο κώδικας python είναι στο αρχείο my_ga.py

## Β3. Αξιολόγηση και Επίδραση Παραμέτρων:

| Α/Α | ΜΕΓΕΘΟΣ ΠΛΗΘΥΣΜΟΥ | ΠΙΘΑΝΟΤΗΤΑ ΔΙΑΣΤΑΥΡΩΣΗΣ | ΠΙΘΑΝΟΤΗΤΑ ΜΕΤΑΛΛΑΞΗΣ | ΜΕΣΗ ΤΙΜΗ ΒΕΛΤΙΣΤΟΥ | Μ.ΑΡΙΘΜΟΣ ΓΕΝΕΩΝ |
|---|---|---|---|---|---|
| 1 | 20 | 0.6 | 0.00 | 0.15705920103815255 | 50 |
| 2 | 20 | 0.6 | 0.01 | 0.1533069887166633 | 45 |
| 3 | 20 | 0.6 | 0.10 | 0.31150509671322796 | 45 |
| 4 | 20 | 0.9 | 0.01 | 0.15755017994836262 | 62 |
| 5 | 20 | 0.1 | 0.01 | 0.15512708382142892 | 62 |
| 6 | 200 | 0.6 | 0.00 | 0.490455707761918 | 39 |
| 7 | 200 | 0.6 | 0.01 | 0.5606458744229067 | 39 |
| 8 | 200 | 0.6 | 0.10 | 0.7024842288406511 | 67 |
| 9 | 200 | 0.9 | 0.01 | 0.674985891663385 | 54 |
| 10 | 200 | 0.1 | 0.01 | 0.6296738859644042 | 49 |

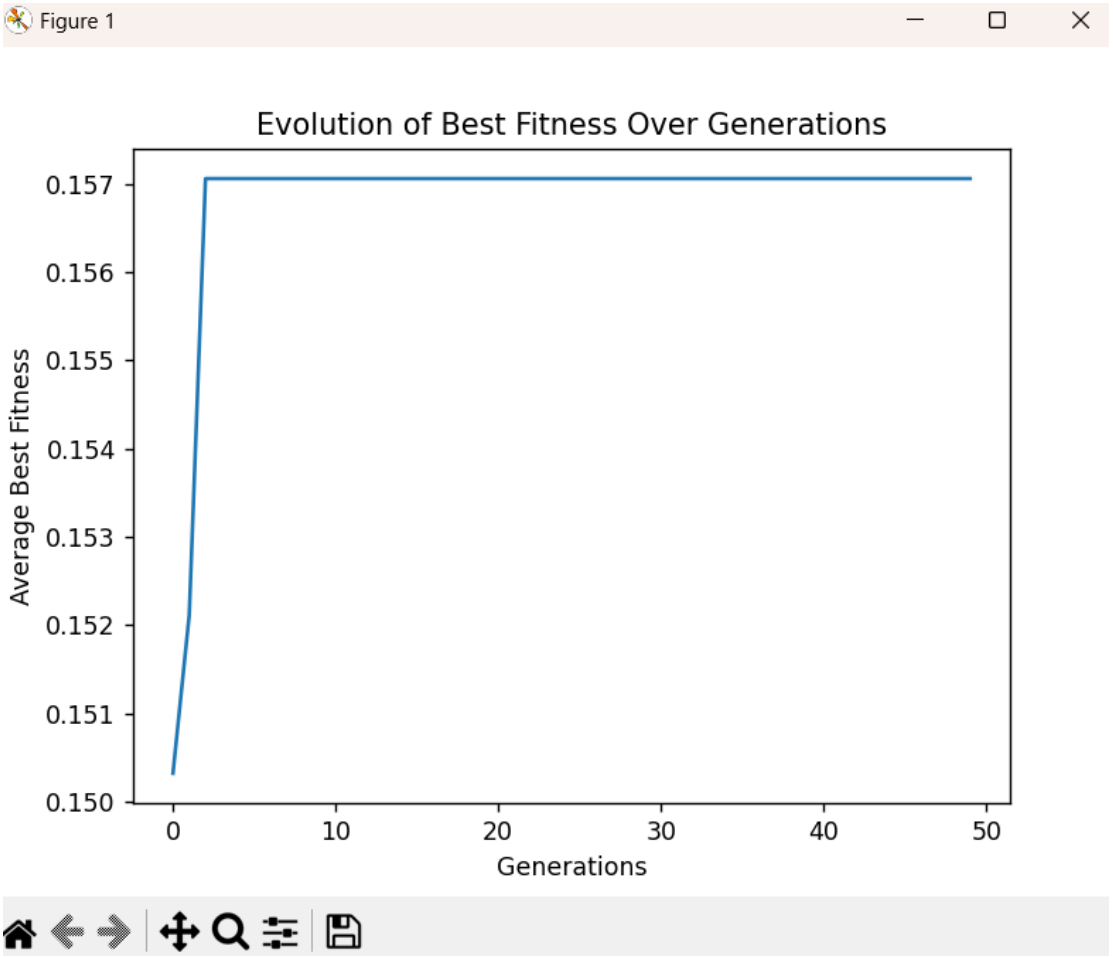
**α)**

Να σημειωθεί ότι η μέση τιμή βελτίστου είναι ο μέσος όρος από τα best fitness (δηλαδή τα καλύτερα σε απόδοση άτομα) που προκύπτουν από κάθε τρέξιμο του αλγορίθμου. Σε κάθε περίπτωση εκτελούσα 12 πειράματα, δηλαδή έτρεχα 12 φορές τον αλγόριθμο με τις ίδιες

παραμέτρους και έπαιρνα διαφορετικό best fitness κάθε φορά. Αυτός λοιπόν είναι ο μέσος όρος. Επίσης άλλαζα σε κάθε περίπτωση χειροκίνητα τον αριθμό γενεών που θέλω κάθε φορά.

**β)** Στα παρακάτω διαγράμματα παρουσιάζω την εξέλιξη του average best fitness κατά την διάρκεια των γενεών. Επίσης κάτω από κάθε διάγραμμα έχω σημειώσει την επιγραφή που προκύπτει στο καλύτερο τρέξιμο από τα 12, με βάση το καλύτερο fitness που προέκυψε:

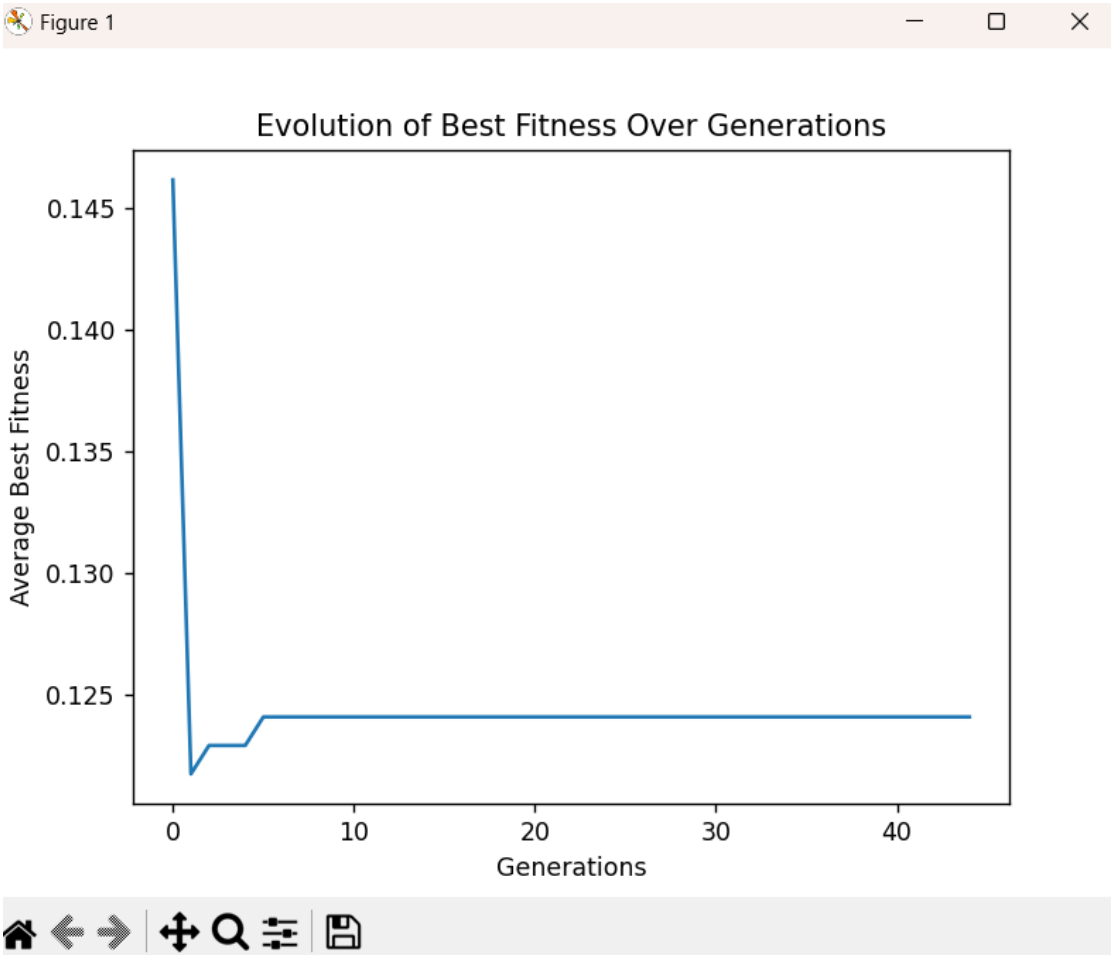
**1η περίπτωση:**




**Επιγραφή:**

```
Restored Text: τον μετα
Best Fitness: 0.46771005689261785
Generations: 50
```

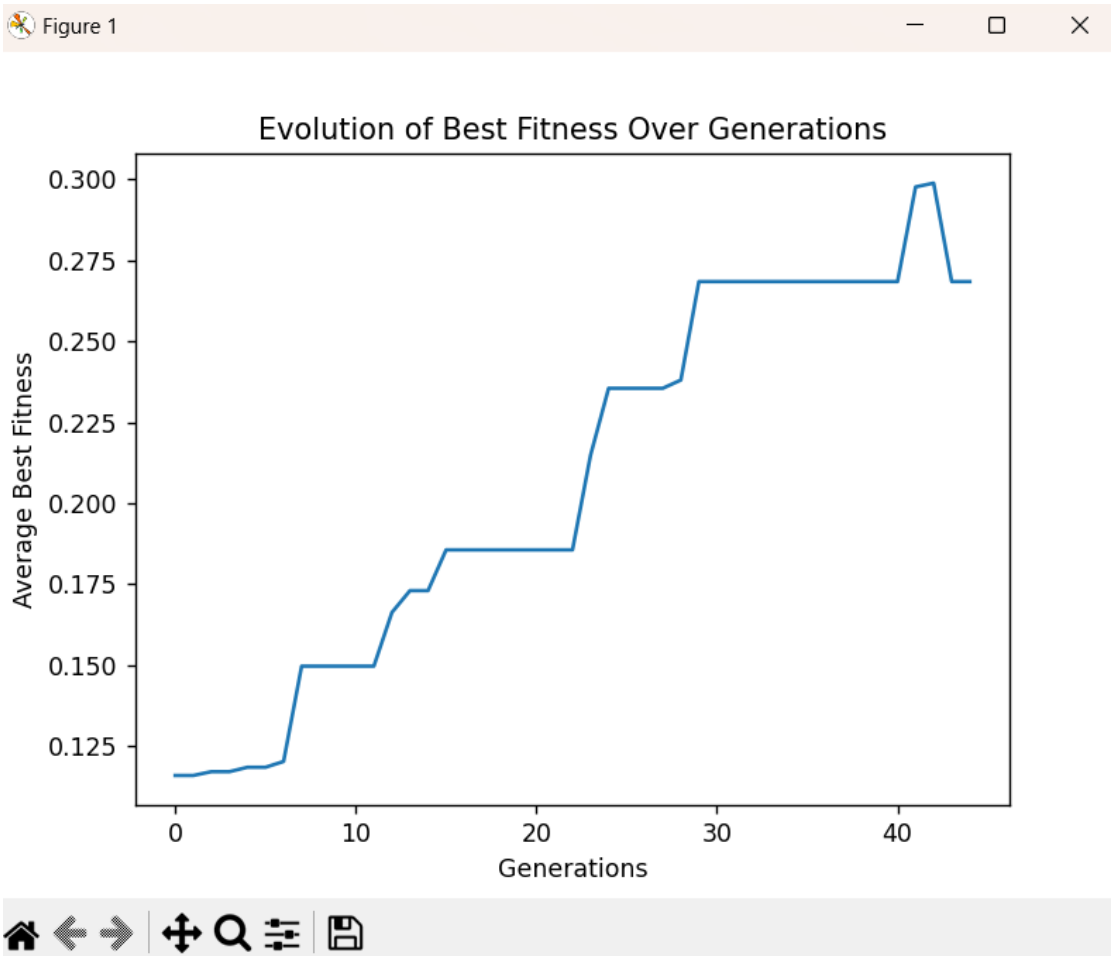
**2η περίπτωση:**




**Επιγραφή:**

```
Restored Text: τους επιτροπον
Best Fitness: 0.40864147626179664
Generations: 45
```

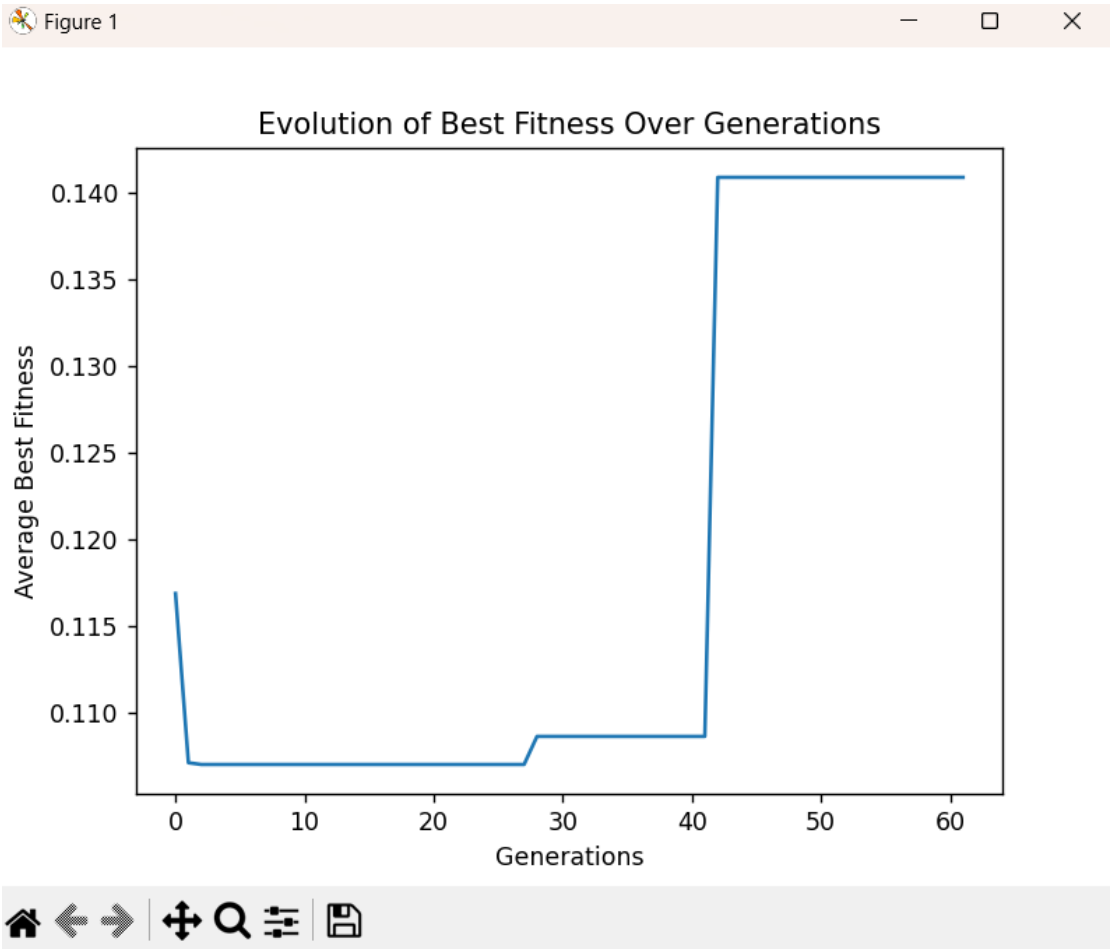
**3η περίπτωση:**




**Επιγραφή:**

```
Restored Text: την τον
Best Fitness: 0.4891586215968199
```

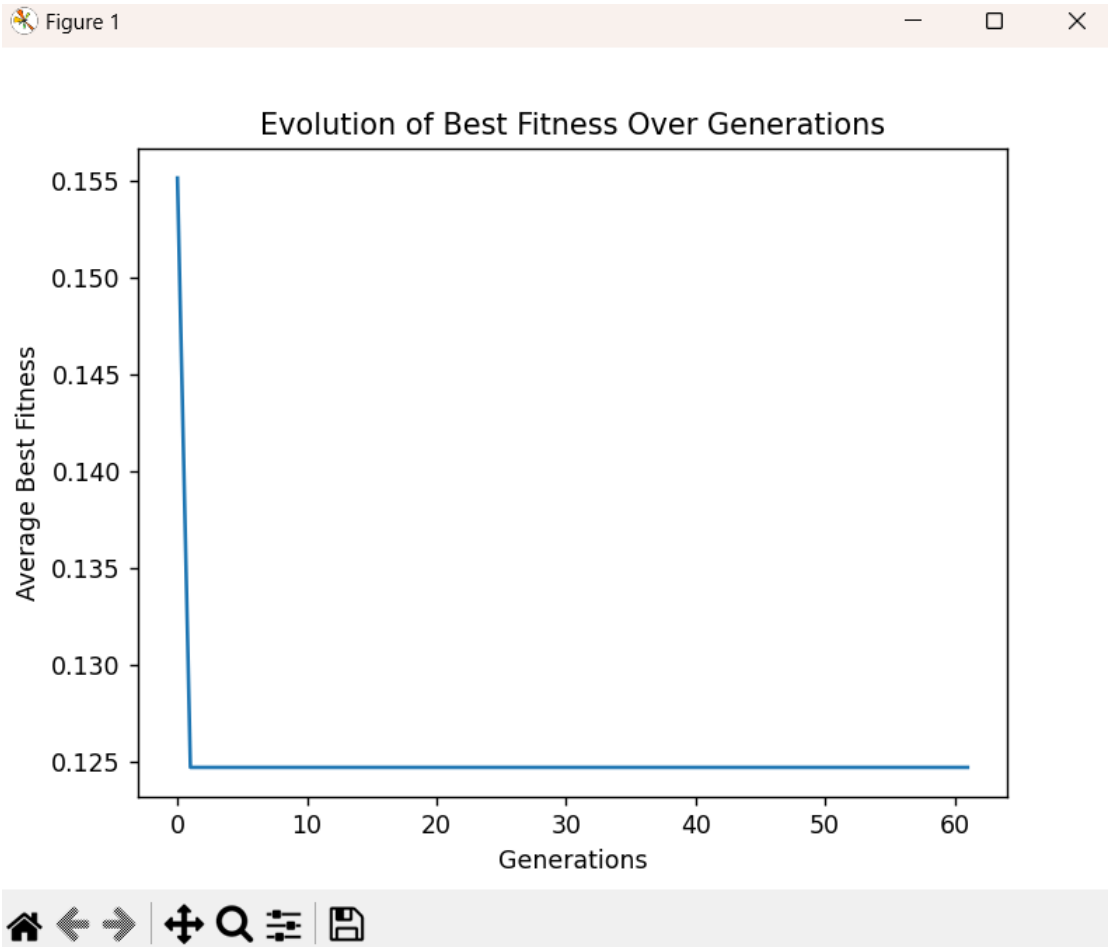
**4η περίπτωση:**




**Επιγραφή:**

```
Restored Text: στρατηγων της
Best Fitness: 0.4210397181564982
Generations: 62
```

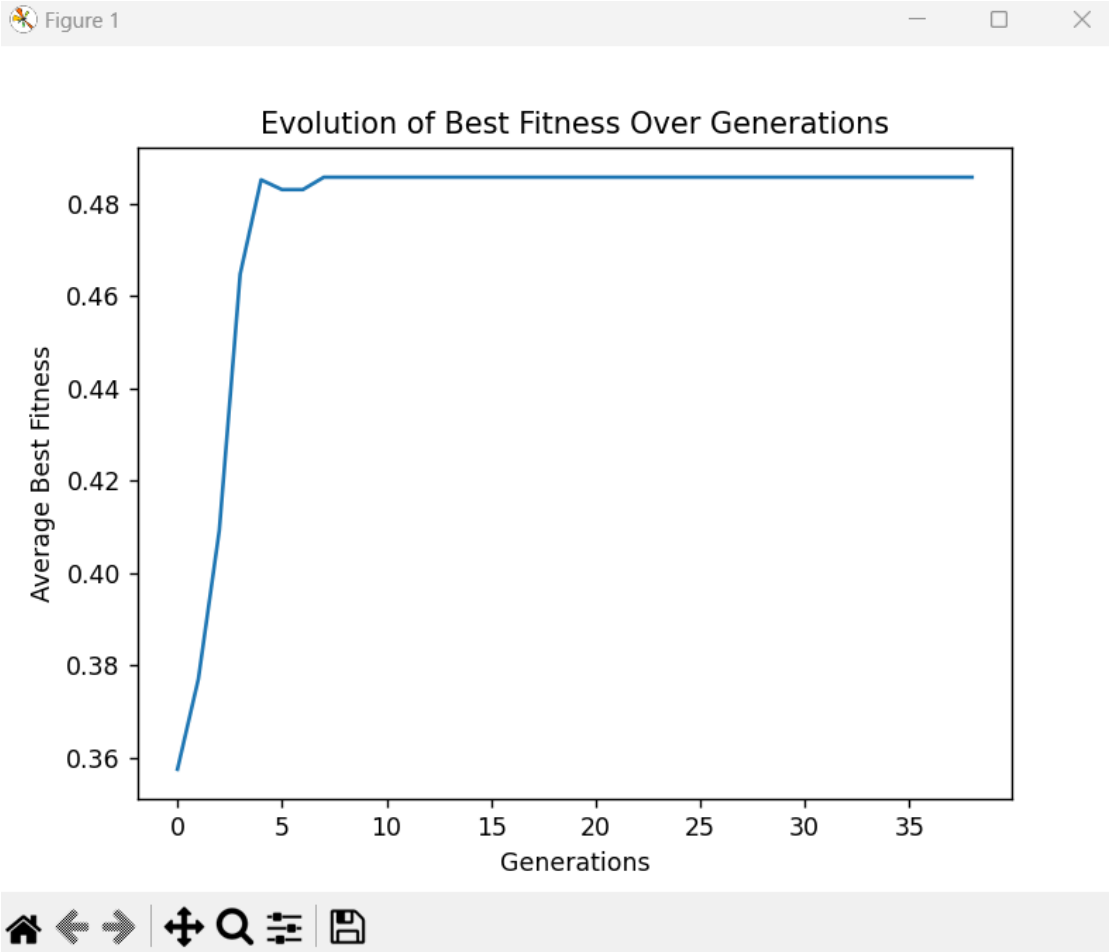
**5η περίπτωση:**




**Επιγραφή:**

```
Restored Text: τον εργον
Best Fitness: 0.40837912310414953
```

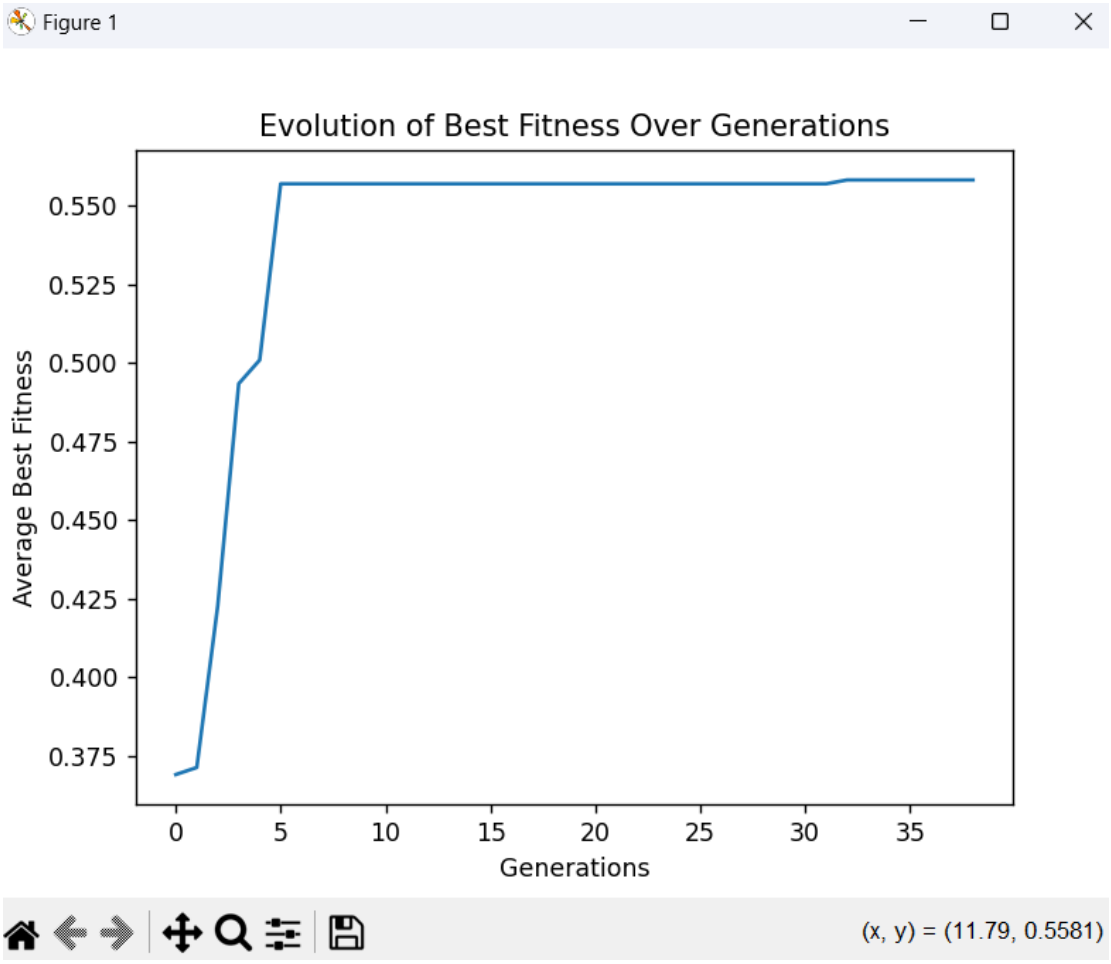
**6η περίπτωση:**




**Επιγραφή:**

```
Restored Text: ουαλεριον τον
Best Fitness: 0.7955265841476473
Generations: 39
```

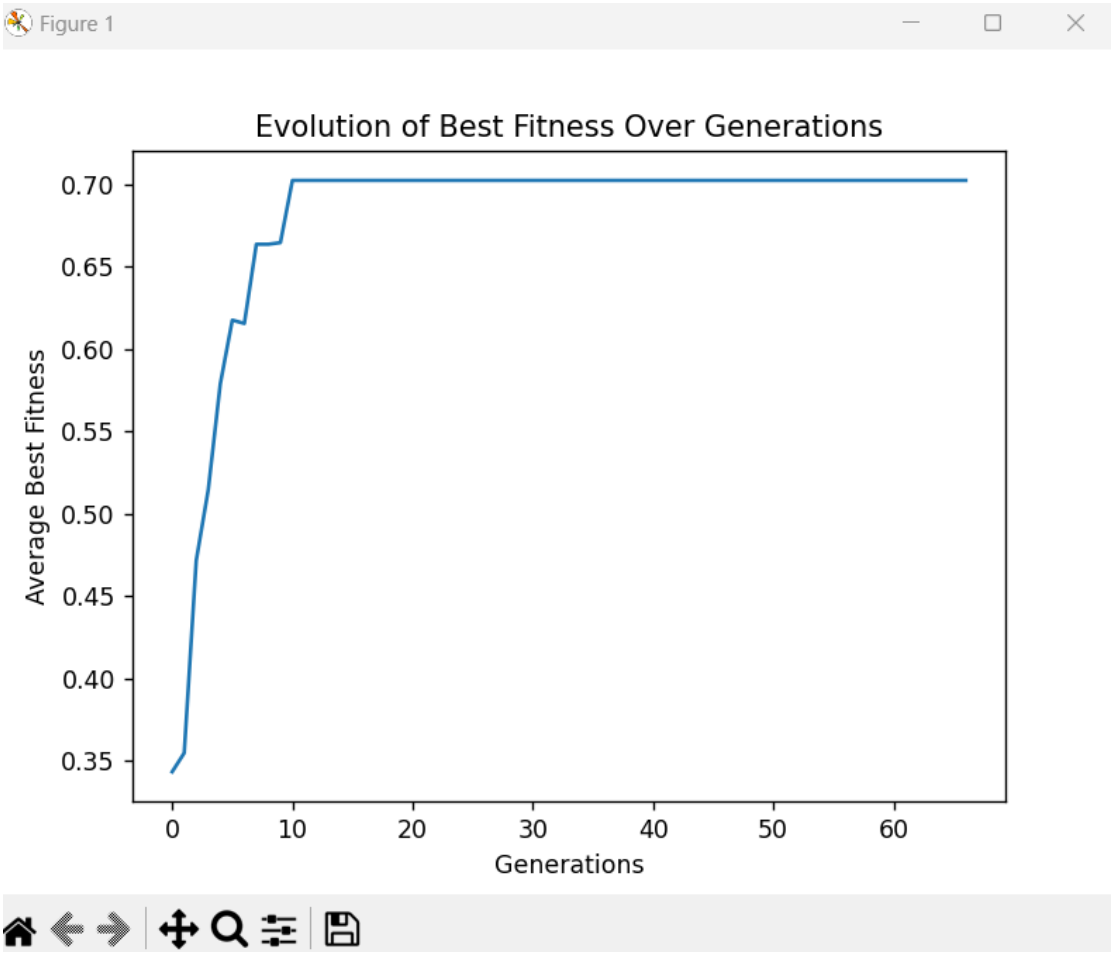
**7η περίπτωση:**




**Επιγραφή:**

```
Restored Text: μητροκολωνια τον
Best Fitness: 0.7955265841476473
Generations: 39
```

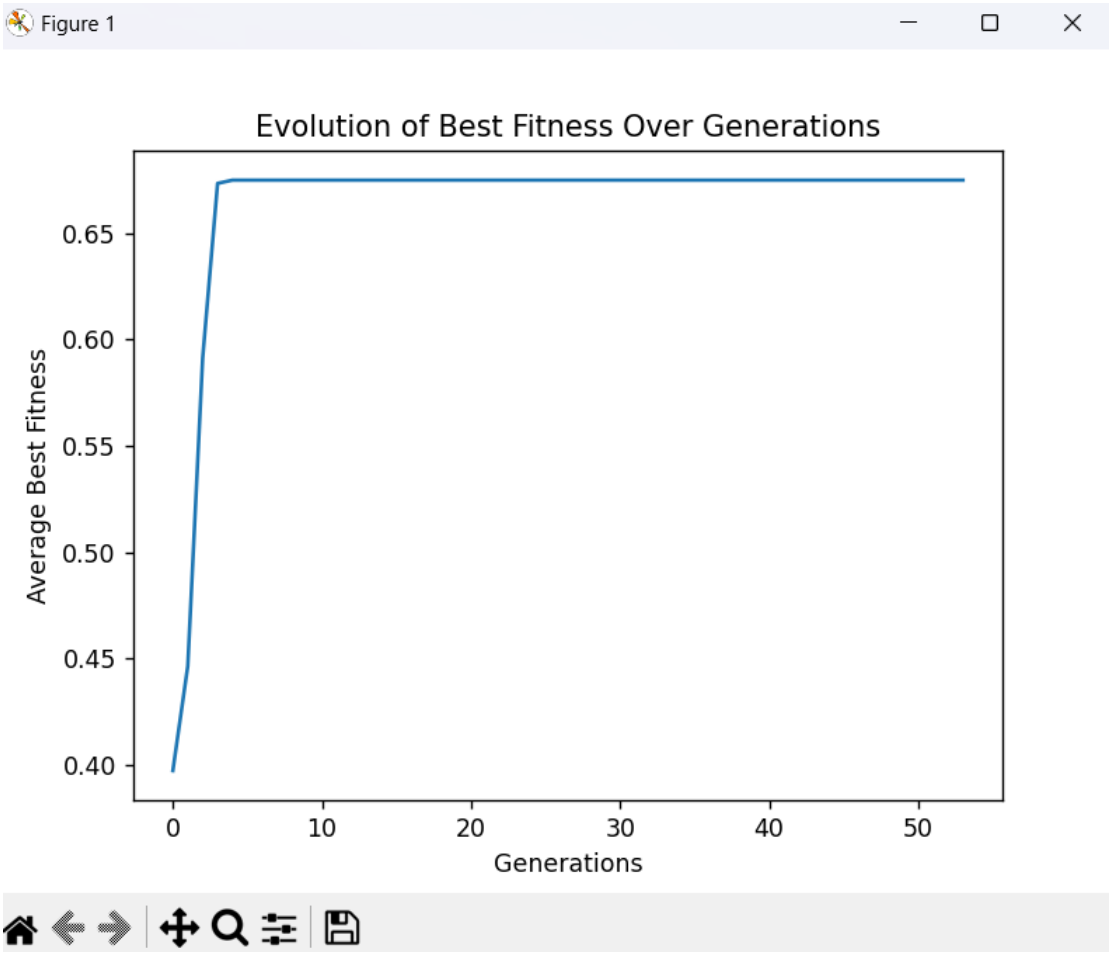
**8η περίπτωση:**




**Επιγραφή:**

```
Restored Text: τον θεοδορου
Best Fitness: 0.7955265841476473
```

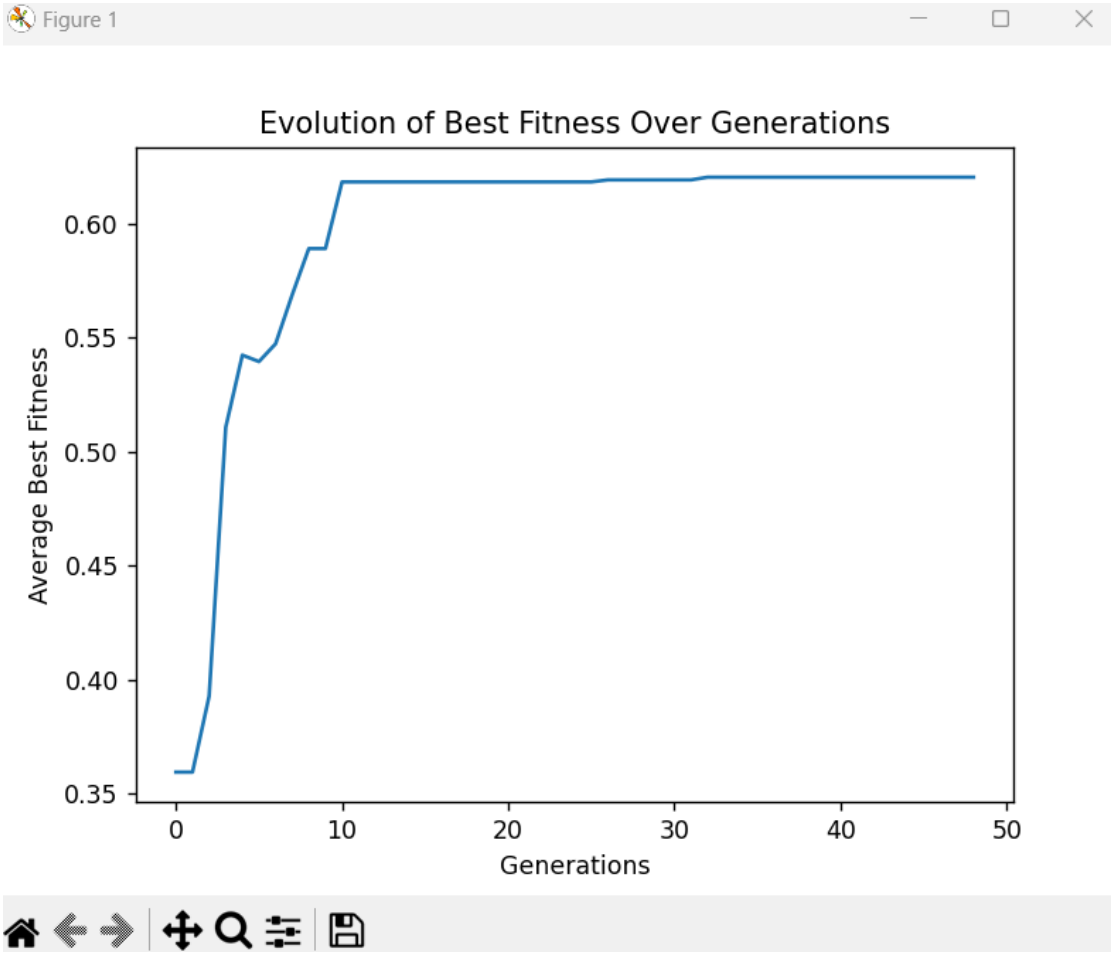
**9η περίπτωση:**




**Επιγραφή:**

```
Restored Text: τον επιτροπον
Best Fitness: 0.7955265841476473
```

**10η περίπτωση:**




**Επιγραφή:**

```
Restored Text: αριστειδου τον
Best Fitness: 0.7955265841476473
```

**γ)** Συμπεράσματα με βάση τα διαγράμματα, αποτελέσματα πίνακα και restored επιγραφές για την επίδραση των παραμέτρων στον ΓΑ:

Αρχικά όπως είχα αναφέρει και στο ερώτημα Β1.γ για τον αρχικό πληθυσμό, η αρχική τιμή που θα δώσουμε παίζει σημαντικό ρόλο καθώς βλέπουμε ότι με την χαμηλή τιμή 20 η μέση τιμή των best fitness των ατόμων δεν ξεπερνά το 0.35 κάτι που είναι αρκετά χαμηλό και οφείλεται στην έλειψη ποικιλίας δειγμάτων. Επομένως στις επόμενες περιπτώσεις που αυξήσαμε τον αρχικό πληθυσμό σε 200, βλέπουμε αρκετά μεγάλη αύξηση στην απόδοση του καλύτερου ατόμου κάθε φορά. Γενικά όσον αφορά τις άλλες δύο παραμέτρους παρατήρησα το εξής, αρχικά παρατηρόντας κάθε γενιά αναλυτικά κατέληξα στο συμπέρασμα ότι η πιθανότητα μετάλλαξης πρέπει να είναι οπωσδήποτε >0.0 δηλαδή να συμβαίνει μετάλλαξη, καθώς έτσι έχουμε πού καλύτερη πιθανότητα να πάρουμε υψηλό best fitness σε κάθε γενιά. Μόλις αφαιρέσουμε το mutation rate αυτομάτως τα best fitness μεινώνονται και μάλιστα σε πολλές γενιές τα best fitness ηταν 0.0 που σημαίνει καμία ομοιότητα. Όσον αφορά μια πιο συγκεκριμένη τιμή για το mutation rate, τα καλύτερα αποτελέσματα βγήκαν με την τιμή 0.1 και όχι με την 0.01. Η πιθανότητα 0.01 είναι υπερβολικά μικρή και με βάση το ότι διαλέγουμε r μεταξύ 0 και 1 για να συγκρίνουμε r< 0.01 φαίνεται πως με τόσο μικρή πιθανότητα οι μεταλλάξεις είναι λίγες, επομένως καλύτερα αποτελέσματα με 0.1. Τέλος όσον αφορά την πιθανότητα διασταύρωσης κρίνοντας με βάση τις περιπτώσεις 7 και 9 όπου πληθυσμός = 200 και mutation rate = 0.01 και στις δύο, βλέπουμε ξεκάθαρα ότι η βέλτιστη λύση δίνεται με crossover rate = 0.9 δηλαδή το υψηλότερο. Παρολαυτα η βέλτιστη λύση από τις 10 περιπτώσεις βλέπουμε πως είναι η 8 όπου crossover rate = 0.6 και mutation rate = 0.1. Το συμπέρασμα είναι πως δεν αρκεί μια από τις δυο πιθανότητες να είναι υψηλή για βέλτιστο αποτέλεσμα, αλλά χρειάζονται και τα δυο εξίσου.

Γενικά υψηλότερες τιμές crossover rate & mutation rate παρέχουν την δυνατότητα για μεγαλύτερη εξερεύνηση στον χώρο αναζήτησης και βοηθούν στην απόκτηση πιο ποικίλων λύσεων. Χαμηλότερες τιμές μπορεί να οδηγήσουν σε γρηγορότερη σύγκλιση, εξασφαλίζουν σταθερότητα και εστιάζουν περισσότερο στις ήδη υπάρχουσες λύσεις. Προσωπικά πιστεύω πως στο ζητούμενο πρόβλημα των επιγραφών χρειαζόμαστε οπωσδήποτε έναν αρκετά μεγάλο χώρο αναζήτησης γιαυτό και είναι προτιμότερες οι πιο μεγάλες τιμές για τις παραμέτρους καθώς δίνουν και πιο ικανοποιητικές λύσεις.