

מעבדה בבינה מלאכותית

דו"ח מעבדה 01 – אלגוריתמים גנטיים

מגיש 1 – עומר צ'רניא 318678620

מגיש 2 – עומר גרהכני 322471145

משימה 1:

במסגרת סעיף זה, נדרשנו לישם את האלגוריתם הגנטי שנלמד בהרצאה, ולתעד עברו כל דור את ביצועי האוכלוסייה באמצעות חישוב:

- Fitness ממוצע
- סטיית תקן
- Fitness הטוב ביותר והגרוע ביותר
- טווח ה-Fitness באוכלוסייה

לצורך כה, מימושנו את הפונקציה `print_generation_stats`, אשר מקבלת את האוכלוסייה הנוכחיות ומספר הדור, ומדפיסה את הנתונים הסטטיסטיים הבאים:

```
# ----- Task 1 -----
def print_generation_stats(population, generation): #Prints the generation stats
    fitness_values = [ind.fitness for ind in population] #Gets the fitness values of the population
    best = population[0] #Gets the best individual in the population
    worst = population[-1] #Gets the worst individual in the population
    avg_fitness = sum(fitness_values) / len(fitness_values) #Calculates the average fitness of the population
    std_dev = statistics.stdev(fitness_values) #Calculates the standard deviation of the fitness of the population
    fitness_range = worst.fitness - best.fitness #Calculates the fitness range of the population

    print(f"Gen {generation}: Best = '{best.string}' (Fitness = {best.fitness})") #Prints the best individual in the population
    print(f" Avg Fitness = {avg_fitness:.2f}") #Prints the average fitness of the population
    print(f" Std Dev = {std_dev:.2f}") #Prints the standard deviation of the fitness of the population
    print(f" Worst Fitness = {worst.fitness}") #Prints the worst fitness of the population
    print(f" Fitness Range = {fitness_range}") #Prints the fitness range of the population
    print()
```

דוגמא לתוצאות הרצה (הרצה שבה צולמו רק דורות 28 עד 30):

```
Gen 28: Best = 'Iello World!' (Fitness = 1)
Avg Fitness = 16.32
Std Dev = 27.54
Worst Fitness = 114
Fitness Range = 113

Gen 29: Best = 'Iello World!' (Fitness = 1)
Avg Fitness = 14.48
Std Dev = 26.59
Worst Fitness = 116
Fitness Range = 115

Gen 30: Best = 'Hello World!' (Fitness = 0)
Avg Fitness = 13.51
Std Dev = 25.82
Worst Fitness = 114
Fitness Range = 114
```

בסעיף 2 אנו נדרשים:

1. לחשב זמן ריצה (Runtime Clock Ticks) בכל דור – כמובן, זמן העיבוד/הUPS שבו האלגוריתם הגנטי פועל מדור לדור.
2. לבדוק על זמן ההתכנסות – כמה זמן בפועל (או כמה דורות) נדרשים עד שהאלגוריתם מגיע לפתרון טוב/אופטימלי, או נתקע במינימום לוקאלי.

הוסףנו מדידות זמן בכל דור באמצעות מודול `time`. בתחילת האלגוריתם, אנו שומרים את הזמן התחלה, ולאחר כל דור מחשבים נתונים זה מודפס יחד עם ערכי הפיטנס של האוכלוסייה באותו דור. `.current_time = time.time() - start_time`

```
# ----- Task 2 -----
tick_end = timeit.default_timer() #Ends the tick
tick_duration = tick_end - tick_start #Calculates the duration of the tick
total_elapsed = tick_end - start_time #Calculates the total elapsed time
print(f"Tick Duration (sec) = {tick_duration:.4f}") #Prints the duration of the tick
print(f"Total Elapsed Time (sec) = {total_elapsed:.4f}") #Prints the total elapsed time
```

תוצאות הריצה לדוגמא:

```
Tick Duration (sec) = 0.0060
Total Elapsed Time (sec) = 0.3538
Gen 29: Best = 'Hello!World!' (Fitness = 1)
    Avg Fitness = 19.08
    Std Dev = 27.85
    Worst Fitness = 120
    Fitness Range = 119

Tick Duration (sec) = 0.0063
Total Elapsed Time (sec) = 0.3654
Gen 30: Best = 'Hello!World!' (Fitness = 1)
    Avg Fitness = 17.13
    Std Dev = 26.75
    Worst Fitness = 116
    Fitness Range = 115

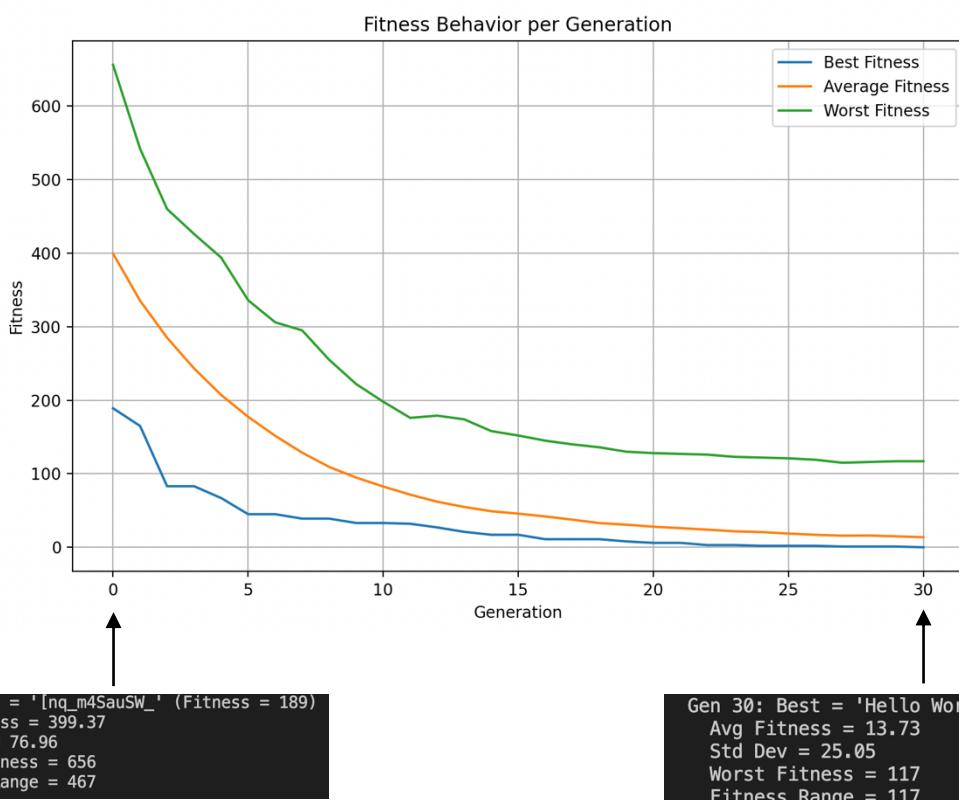
Tick Duration (sec) = 0.0060
Total Elapsed Time (sec) = 0.3766
Gen 31: Best = 'Hello World!' (Fitness = 0)
    Avg Fitness = 16.03
    Std Dev = 26.71
    Worst Fitness = 115
    Fitness Range = 115
```

בסיוף זה של המטלה התקשנו להפיק גרף קווי (Line Graph) המציג בכל דור את ערכי הפיטנס הבאים:
 – הפרט הטוב ביותר באולוסייה באותו דור. Best Fitness •
 – הממוצע של כל הפיטנסים בדור. Average Fitness •
 – הפרט הגרוע ביותר בדור. Worst Fitness •

בתמונות המצורפות, ניתן לראות שהוסףנו קטע הקוד האחראי ל:
 1. איסוף נתונים פיטנס בכל דור לטור מבנה נתונים (רשימת generation).
 2. חישוב מדדי סטטיסטיקה (ממוצע, מקסימום, מינימום, חציון וכו').
 3. שימוש בספרייה גרפית (matplotlib) כדי ליצור את הגрафים.

```
# ----- Task 3_A -----
generations = list(range(len(best_fitness_list)))
plt.figure(figsize=(10, 6))
plt.plot(generations, best_fitness_list, label="Best Fitness")
plt.plot(generations, avg_fitness_list, label="Average Fitness")
plt.plot(generations, worst_fitness_list, label="Worst Fitness")
plt.xlabel("Generation")
plt.ylabel("Fitness")
plt.title("Fitness Behavior per Generation")
plt.legend()
plt.grid(True)
plt.show()
```

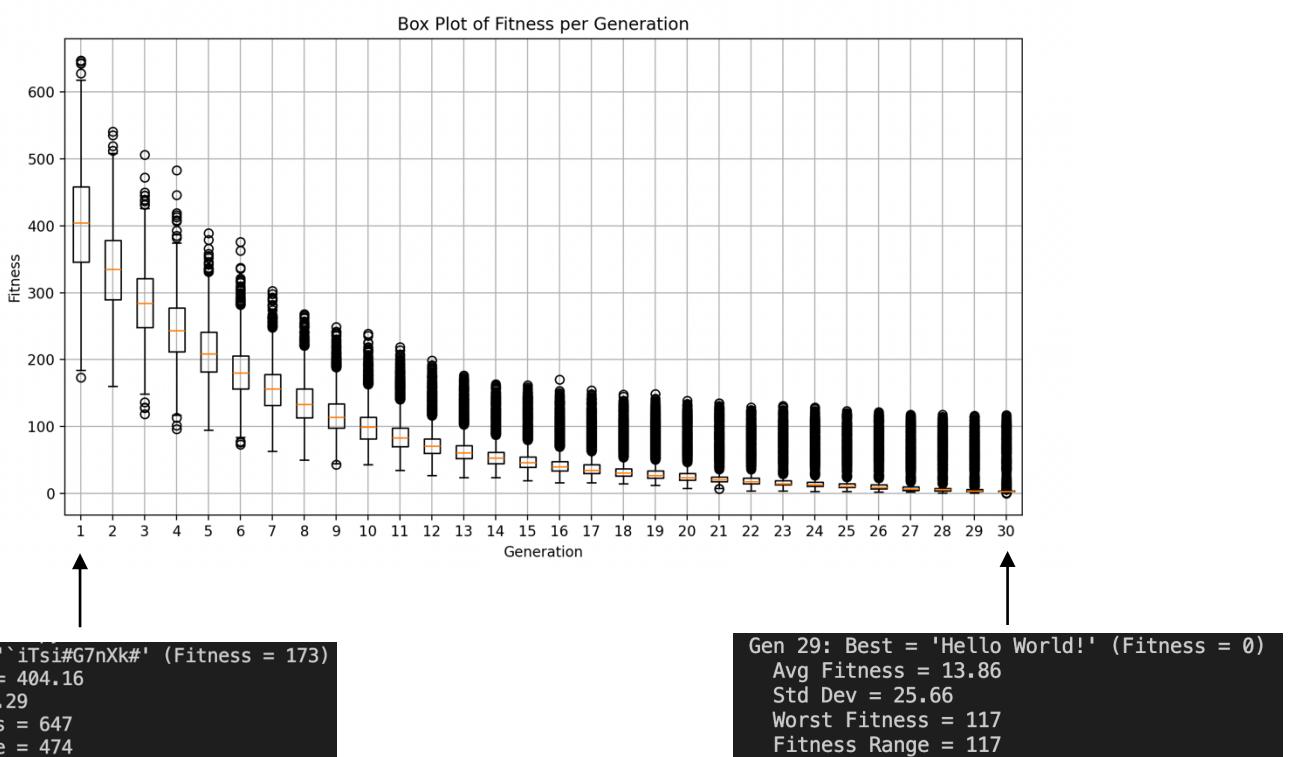
תוצאת הריצה לדוגמא, ביצירוף עם הדרישות של הדור הראשון והדור האחרון עם לוודא את נוכחות הגרף:



בסעיף 3 ב', נדרשנו להציג את התפלגות הפיטנס בכל דור באמצעות תרשימים תיבת (Box Plot). הרעיון הוא להמחיש כיצד הפיטנס של הפרטים באוכלוסייה מפוזר סביב החציון, ומהם ערכי הקיצון. בר' ניתן לראות האם ובאיזה האוכלוסייה מתבצעת לאחור מסוים (פיטנס נמוך) או עדין בשארת מפוזרת. להלן הקוד שהוספנו:

```
# ----- Task 3_B -----
plt.figure(figsize=(12, 6))
plt.boxplot(fitness_distributions, showfliers=True)
plt.xlabel('Generation')
plt.ylabel('Fitness')
plt.title('Box Plot of Fitness per Generation')
plt.grid(True)
plt.show()
```

תוצאת הרצה לדוגמא, בצירוף עם ההדפסות של הדור הראשון והדור האחרון עם לווית נתונים הגרף:



באשר אנו מדברים על שתי הצורות העיקריות של הגראפים (Box Plot ו-Line Graph), בכל אחת מהן ניתן להפיק מידע אחר על אופי הרצה של האלגוריתם הגרפי:

1. גרפ קוויי – Best, Average, Worst

- **Best Fitness** – מציאן את הפיטנס הטוב ביותר בדור הנוכחי, כלומר הפרט (אינדיבידואל) שהביใกลיה לפתרון רצוי. אם האלגוריתם מתקדם היטוב, נראה קו שיורד בהדרגה – בעוניות מינימיזציה, או עולה – בעוניות מקסימיזציה.
- **Average Fitness** – מייצג את ממוצע הפיטנס בכל האוכלוסייה באותו דור. עוזר לראות האם רוב האוכלוסייה משתפרת או רק בזדדים. אם הקו נעה באותו כיוון כמו הקו של **Best** (ירד בעוניות מינימום או עלה בעוניות מקסימום), סימן שככל האוכלוסייה שואפת לעבר פתרונות טובים יותר.
- **Worst Fitness** – מייצג את הפרט בעל הפיטנס הגרוע ביותר באותו דור. אם האלגוריתם אכן מקדם את כל האוכלוסייה, נראה שבזרות המאוחרים גם **Worst** משתפר (הפיטנס מתקרב יותר לפתרון), וכך הפעם בין **Best** לבין **Worst** קטן.

2. תרשימים תיבעה : (Box Plot)

Box Plot בכל דור מראה כיצד ערכי הפיטנס באוכלוסייה מתפלגים:

- הקו האופקי המרכז בTİBVA מייצג את החציון.
- גבולות התיבה הם הרביעון התחתון (Q1) והרביעון העליון (Q3).
- ה- Whiskers מציגים את הגבולות מחוץ לתיבעה.
- Outliers: אם יש ערכי פיטנס>KIZONIM, הם יצירום בנקודות בודדות מחוץ לטווח ה-Whiskers.

בקשר לריצה:

ניתן לראות האם בכל דור הערכים "מתכנסים" למטרך מסוות (בשתייה הולכת ונהיית צרה והחציון זו לביאן מסוים), או שעדין יש טווח רחב (תיבה רחבה) של ערכי פיטנס. אם תיבות הדורות הראשוניים וחוות 매우 וופכות צרות בדורות מאוחדים, זה מראה על ירידה בפייזור בין פרטיהם, ככלומר מקרים פתרונות טובים יותר. עדין עשויים להופיע חריגים (Outliers) בדורות המאוחדים, לרוב זה יהיה תוצר של מוטציות או פרטיהם שנתקעו "מאחור". מוכחות חריגים לעממים רציה כדי לשמר גיון גנטי (exploration), אבל אם כולם "צמודים" לתיבה ללא חריגים, ניתן שהאלגוריתם התכנס חזק מאוד (מה שעלול לגרום לתקינה מקומית).

משימה 4:

במשימה זו התקשנו:

- להוסיף ולמש שłosa אופרטורי שחילוף (Crossover) שונים באלגוריתם הגנטי –
 - Single-Point Crossover (נקודות חתך אחת)
 - Two-Point Crossover (שתי נקודות חתך)
 - Uniform Crossover (בחירה אקראית של כל גן מאחד ההורים)
- לאפשר בחירה דינמית של סוג השחלוף שරוצים להפעיל בכל ריצה של האלגוריתם (למשל באמצעות משתנה GA_CROSSOVER_OPERATOR).
- לשלב את השחלוף החדש בפונקציית mate כך שבמקרה נקודת חתך אקראית בודדת, האלגוריתם יוכל לעבוד עם שלושת הסוגים השונים שבייקשו.

איך מימושו את פתרון הבעה?

- הגדרת פונקציות השחלוף:
בקוד (בתמונה המצורפת) בתבנו שלוש פונקציות:
`crossover_single(parent1, parent2):` בוחרת אינדקס חתך יחיד (point) באופן אקראי, ומיצרת ילד (Child) מחלק הראשון של parent1 וחלק שני של parent2.
`crossover_two(parent1, parent2):` בוחרת שתי נקודות חיתוך point1 ו-point2, ומחלקת את המחרוזות לשולש מקטעים. הילד נבנה משלילוב של קטעים מהורה ראשון ומהורה שני בסדר החיתוך הנבחר.
`crossover_uniform(parent1, parent2)`: עובה על כל אינדקס בתווך אורך המחרוזות, ובכל גן בוחרת אם לחתה מ- parent1 או מ-parent2 בהסתברות 0.5 (או הסתברות אחרת לבחירת המשמש שכינתה לבחירה ע"י שינוי הקבוע בקוד), וכבר בונה ילד "מעורבב".

2. שילוב בפונקציית mate:

- בתחילת mate ממבצע Elitism (שמירה על הפרטים הטובים ביותר).
- בתווך לולאה, נבחרים שני הורים אקראיים (i1,i2) מתוך האוכלוסייה.
- בהתאם לערך של GA_CROSSOVER_OPERATOR ממבצע השחלוף הנבחר.

```
# --- New crossover operator functions ---
def crossover_single(parent1, parent2):
    tsize = len(parent1.string)
    spos = random.randint(0, tsize - 1)
    return parent1.string[:spos] + parent2.string[spos:]

def crossover_two(parent1, parent2):
    tsize = len(parent1.string)
    if tsize < 2:
        return crossover_single(parent1, parent2)
    point1 = random.randint(0, tsize - 2)
    point2 = random.randint(point1 + 1, tsize - 1)
    return parent1.string[:point1] + parent2.string[point1:point2] + parent1.string[point2:]

def crossover_uniform(parent1, parent2):
    tsize = len(parent1.string)
    child_chars = []
    for i in range(tsize):
        if random.random() < 0.5:
            child_chars.append(parent1.string[i])
        else:
            child_chars.append(parent2.string[i])
    return ''.join(child_chars)

# --- End of crossover operator functions ---
```

```
def mate(population, buffer): #Mates the population, creates a new population by mating the fittest individuals
    esize = int(GA_POPSIZE * GA_ELITRATE)
    tsize = len(GA_TARGET)

    elitism(population, buffer, esize)

    for i in range(esize, GA_POPSIZE):
        i1 = random.randint(0, GA_POPSIZE // 2)
        i2 = random.randint(0, GA_POPSIZE // 2)

        # Instead of using a single random crossover point, use the selected operator:
        if GA_CROSSOVER_OPERATOR == "SINGLE":
            child_string = crossover_single(population[i1], population[i2])
        elif GA_CROSSOVER_OPERATOR == "TWO":
            child_string = crossover_two(population[i1], population[i2])
        elif GA_CROSSOVER_OPERATOR == "UNIFORM":
            child_string = crossover_uniform(population[i1], population[i2])
        else:
            child_string = crossover_single(population[i1], population[i2])

        child = GAIIndividual(child_string)

        if random.random() < GA_MUTATIONRATE:
            child.mutate()

        buffer.append(child)
```

ע"מ לבחור שחלוף, בתחילת הרצאה המשתמש צריך לקליד את סוג השחלוף שהוא רוצה בריצת הנקחית.

```
Select crossover operator:
1 - SINGLE
2 - TWO
3 - UNIFORM
4 - TRIVIAL
Enter your choice (1/2/3/4): ■
```

באלגוריתם שלנו ניתן לבדוק בין שני מנגנוני עיקריים – מנגנון שמביא לגיון (Exploration) ומנגנון שמקוון לניצול הפתרונות הטובים (Exploitation):

מנגנון Exploration (חיפוש וגיון):

- אתחול אקראי של האוכלוסייה (population_init):

בהתחלתו, אנו יוצרים אוכלוסייה מלאה באורח אקראי. כך מתקבל מגוון רחב של פתרונות ההתחלתיים, שגם אם חלקם רוחקים מהפתרון המקורי, הם מספקים את הבסיס לחיפוש למרחב הפתרונות.

- מוטציה (mutation):

הfonקציה של המוטציה משנהתו אקראי במחוזת, ובכך מציגה שינויים בלתי צפויים ומחדשת את המגוון הגנטי באוכלוסייה. זה מאפשר לאלגוריתם לצאת מאזורים שבו הוא נתקע (локאל אופטימום) ולבחון פתרונות חדשים.

- שימוש באופרטורי שיחלוף שונים:
מעבר לאופרטורים של TWO-SINGLE, השימוש בבחירה אופרטור שונה (בהתאם לקלט המשתמש) יכול להויסף אלמנט של גיון – כל אופרטור משלב בצורה שונה את הגנים מההורם, ובכך יכול לחושף תכונות חדשות בפתרונות.

מנגנון Exploitation (ניצול הפתרונות הטובים)

- מינון ובחירה הפרטים הטובים (sort_population и elitism):

לאחר חישוב הפיטנס לכל פרט, האוכלוסייה ממיננת כך שהפרט עם הפיטנס הטוב ביותר נמצא בראש הרשימה. באמצעות אליטיזם מועברים הפרטים הטובים ביותר לשירות לדoor הבא. כך נשמרים הפתרונות האיכותיים ומעברים הלהא, ובכך נעשה ניצול של המידע הגנטי של הפרטים המוצלחים.

- בחירת הורים מתוך החצי העליון:

כאשר מtbody מtbody מtbody (crossover) בפרצתורה mate(), נבחרים הורים באקראי מתוך החצי העליון של האוכלוסייה. בחירה זו מחזקת את התכונות הטובות שנמצאו אצל הפרטים המובילים, ובכך ממקסמת את exploitation.

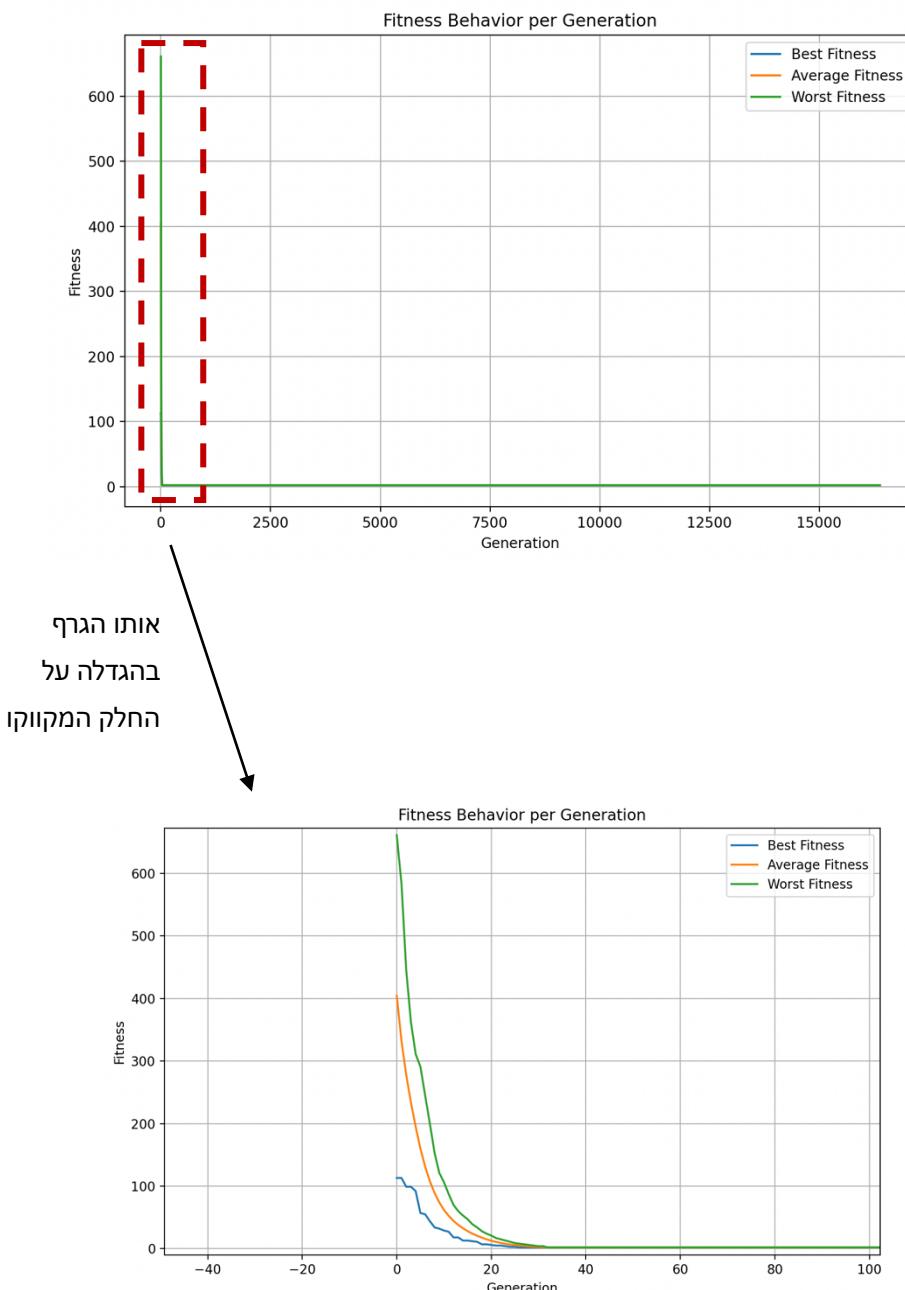
- שילוב גנים (Crossover):

אופרטורי השיחלוף (crossover) עצם, כאשר הם מtbody מtbody על פרטים שנבחרו במידה מיטבית, מהווים חלק מנגנון exploitation – הם משלבים בצורה ישירה את הגנים של ההורים הטובים לצירוף צאצאים, תוך שמירה על תכונות חייבות.

הפעלת האלגוריתם הגנטי ללא מוטציה, להלן הkonfiguracija שבה השתמשנו:

```
GA_POPSIZE = 2048
GA_MAXITER = 16384
GA_ELITRATE = 0.10
GA_MUTATIONRATE = 0 # Task 6A - No mutation
```

כולם, בכל דור יש רק תהליך של בחרה וחלוף (Crossover) בין הורים, ללא כל שינוי אקראי נוסף במחזורות היציאה.
להלן הגרף לאחר ריצת האלגוריתם (האלגוריתם לא מצא פתרון והופסק לאחר 16384 דורות):



ניתוח התוצאות בגרף

1. התנהוגת מהירה בתחילת הריצה:

מהגרף אפשר לראות שבדור הראשון (~ 0 Gen), ה-*Worst Fitness* (הקו הירוק) נמצא בערך 600–700, ואילו ה-*Best* וה-*Average* גובהים יחסית גם כן. כבר לאחר מספר דורות קטן, כל הקווים צונחים בצורה חדה לאוצר פיטנס מאוד נמוך.

- הסיבה לנפילה המהירה: בשלב הראשון קיים פיזור גדול באוכלוסייה (Population Size = 2048), ובכל שmbցאים שחולף בלבד בין הורים טובים (Selection + Elitism), הרבה מהפרטים "נמשכים" במהירות לפתרונות טובים יותר.
2. התכונות "רוועשת" אך לא מוציאה: כיוון שאין מוציאה, המקור היחיד לשינוי גנטי באוכלוסייה הוא השחלוף בין זוגות הורים. אם האוכלוסייה הראשונית גדולה ומכסה חלק נרחב ממוחב החיפוש, יתכן שדגימות ראשונות יתפסו פתרון מצין או קרוב מאד לפתרון אופטימלי – וזה האלגוריתם יזנק במהירות לפתרון זהה.
 3. סיכון להתקעות במינימום לוקאלי: בהיעדר מוציאה, ברגע שאיבדנו גן בלשחו (תמונה מסוימת באוכלוסייה), אין מנגנון שיכניס אותו מחדש. אם הבעה הייתה מורכבת יותר, או אם לא היה לנו מזל באוכלוסייה ההתחלתית, עלולה להתרכש התכונות מוקדמת למינימום לוקאלי – ואי אפשר להיחלץ ממנו ללא מוציאה.
 4. גודל אוכלוסייה גדול מוביל למציאת פתרון מהר: בגרף המצורף רואים שהתכונות צוז, במקרה שלנו, במקרה הגעה מהר לפתרון סביר (אולי אף אופטימלי) ואין שייפור בהמשך.
 5. חלק מרכזי בהצלחה של "Crossover Only" נבע בכך שאנו משתמשים באוכלוסייה של 2048 פרטימ. כמות גדולה של פרטימ מגבירה את הסיכוי שחלקם יהיו קרובים מאוד לפתרון כבר בשלביה, וזה האלגוריתם יתמקד בהם.

מסקנות מסעיף 6.א

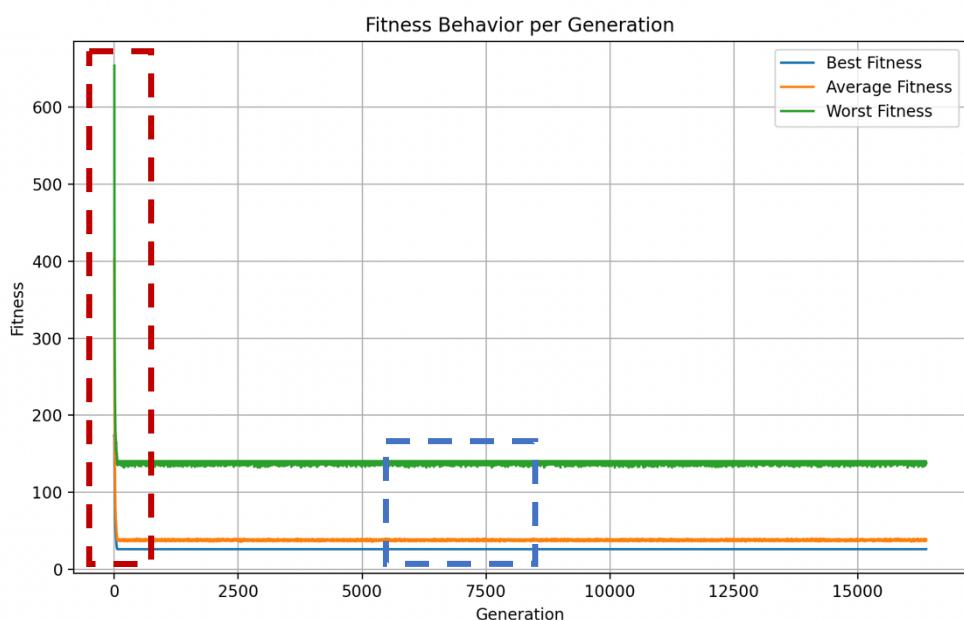
יתרונות: כאשר בעיית היעד היא יחסית "קלה" או שהאוכלוסייה ההתחלתית מספיק מגונת וגדולה, האלגוריתם עשוי להגיע מהר מאוד לפתרון גם ללא מוציאה.

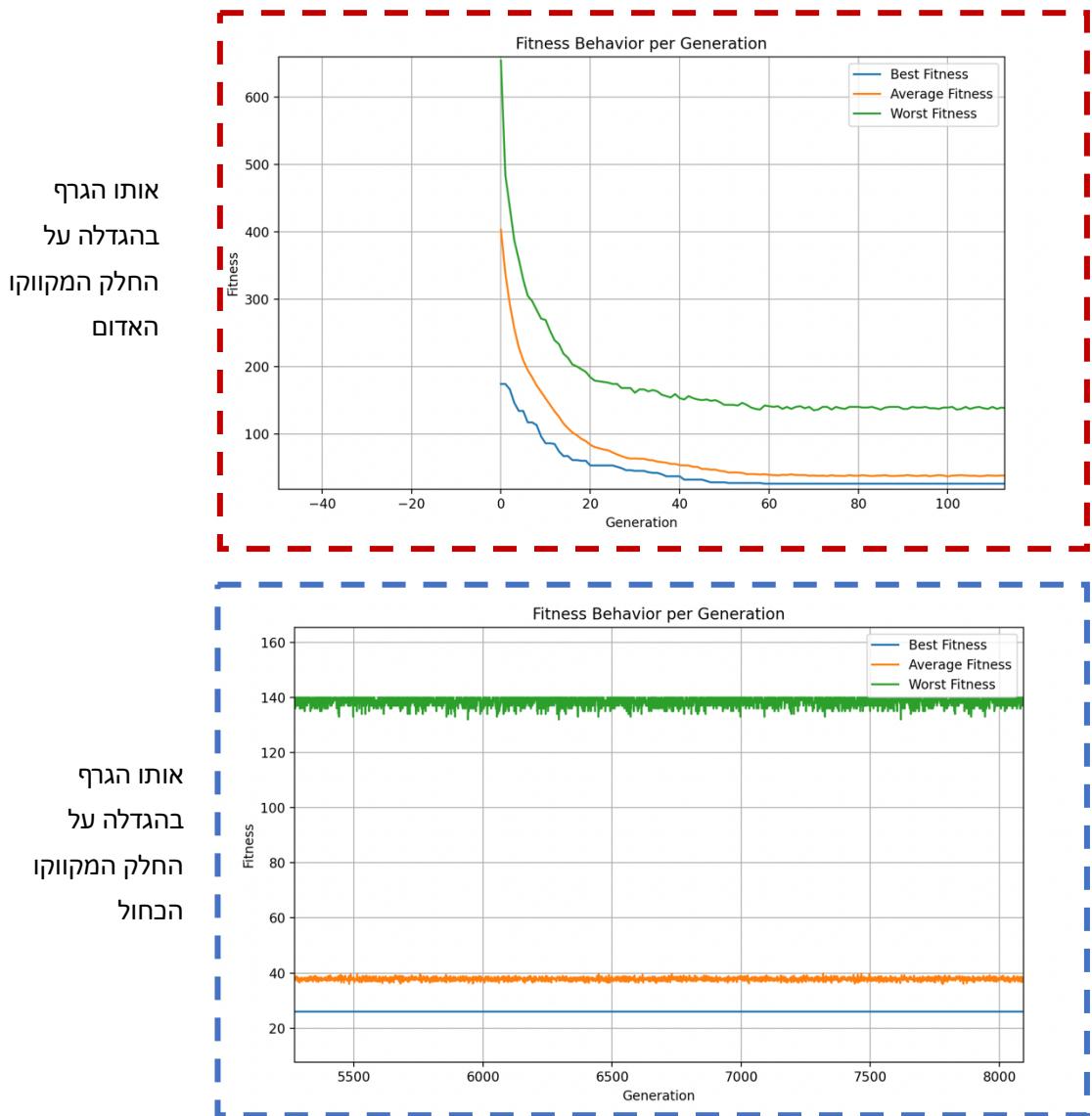
חסרונות: ללא מוציאה, האלגוריתם עלול להתכנס מהר מדי לפתרון מקומי (Local Optimum) ולא יצילח "לבrhoח" ממנו. במקרה שלנו, במקרה חישוק שחק תפקיד והאלגוריתם מצא פתרון ייחודי, אך לא מספיק כדי שהאלגוריתם יעצור והוא נתקע במינימום לוקאלי. חשיבות מוציאה: במקרים מסוימים יותר, מומלץ לכלול מוציאה כדי לשמור על גיון גנטי לאורך זמן, ולמנוע ההתקעות מוקדמת במינימום לוקאלי.

סעיף ב:

בסעיף זה, הפעלנו את האלגוריתם רק עם מוציאות ולא שחולף כלל. עם לעשות את זה, הוספנו אופציה לבצע שחולף – ללא שחולף אמיתי (אופציה 4 בתמונה מסעיפים קודמים) ע"י לקיחת הערכים במלואם מהודים בצורה אקראית.

להלן הגרף לאחר ריצת האלגוריתם (האלגוריתם לא מצא פתרון והפסיק לאחר 16384 דורות):





ניתוח התוצאות מהגרף:

1. שיפור מוגבל והתקנות לדוויה מוקדמת:

מאחר שאין שיתוף של גנים בין פרטימ שוניים, האלגוריתם מתקדם רק ע"י שינויים אקראיים (מוטציות).
בשלב הראשון יכול להיות שיפור מהיר אם אחד הפרטימ התamel מזל וקיבל מוטציה חיובית, אך לאחר זמן עלולה להופיע התיעצבות – האלגוריתם לא מנצח גנים שכבר הצליחו באינדיידואל אחד כדי "להפיץ" אותם אחרים.

2. רמת פיזור (Exploration) גבוהה, אך ללא שילוב תכונות:

מוטציות לבן יכולות להמשיך ולחזור אוזרים חדשים למרחב הפתרונות באופן אקראי.
מצד שני, אין דרך "לחבר" גנים טובים מפרטים שונים, מכיוון שאין שחלוף. יתרון שהאלגוריתם "יבזבז" זמן רב בניסוי להגעה לפתרון איקוני רק באמצעות ניסיונות מזל אקרים.

3. הקו של **Best Fitness** נשאר גבוה יחסית:

מהגרפים ניתן לראות שהפיטנס הטוב ביותר ביוטר (הקו הכחול) אמנם משתפר מהדור הראשון, אך מגיע לרמה מסוימת ומתתקשה לדרכ מעבר אליה. הסיבה היא שאין מנגנון שmaps את ה"גנים הטובים" מעבר לפרט שבו הם הופיעו.

4. ערכים יציבים לאחר אלפי דורות:

פעמים רבות רואים בגרף שה-Worst Fitness (ירוק) וה-Average Fitness (ירוק) נעים סביב ערך די קבוע, בזמן שה-Best Fitness (כחול) מעט נמוך יותר אבל לא משתפר באופן משמעותי. זה סימן קלאסי ל"תקינות" סבב פתרון בינוני – ללא עירוב גנים, יש גבול לכמה אפשר להתקדם מוטציונית בלבד.

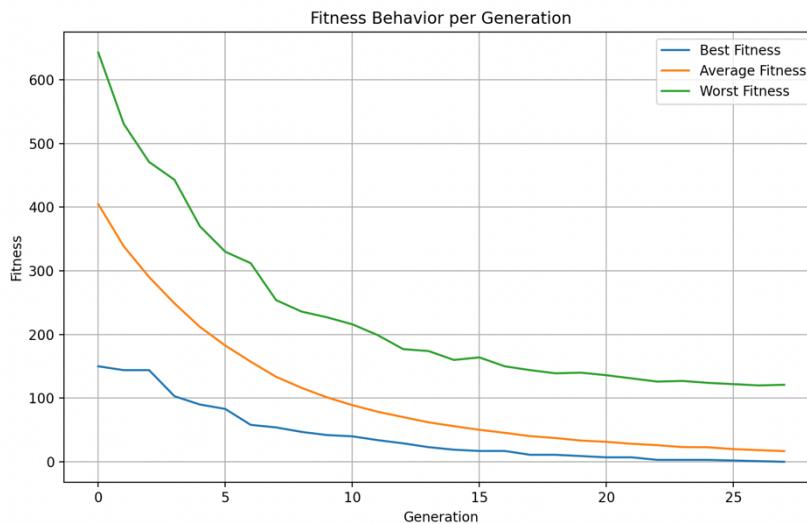
מסקנות מסעיף 6.ב

יתרונ: האלגוריתם ממישר "להציג" באופן אקראי לאזרורים שונים של המרחב, אך שתיאורטי אין סכנת "מחיקה" של גנים – תמיד אפשר שיווצרו מחדש.

חסרון: התקדמות איטית מאוד ואפשרות לחסור יציבות או "תקיעה" על פיטנס בינו אם המוטציות לא מיצירות צירופים מוצלחים.

סעיף ג

בסעיף זה שילבנו מוטציות בלבד עם שחזור. להלן הגרף שייצא לאחר הרצה זו:



הגרף הוכנס לפתרון לאחר 30~ דורות.

מסקנות מסעיף 6.ג

שילוב של Crossover + Mutation מניב את האלגוריתם הגנטי הקלאס. רואים גרף שבו הוא קצב השיפור (בדוגמת סעיף 6.א) והן הגיון לאורך זמן (בדוגמת סעיף 6.ב) באים לידי ביטוי, ולכן האלגוריתם מגיע לפתרונות משופרים ביחס למקורה של "רק שחזור" או "רק מוטציה".

מה היה עליו לעשות במשימה?

1. להוסף היריסטיות LCS:

הגדנו קритריון פיטנס חדש, המבוסס על חישוב Longest Common Subsequence בין המחרוזת הנוכחית (הפתרון הנוכחי) לבין מחרוזת היעד (Target String). ככלומר, יש לבדוק בכל פרט עד כמה תתי-סדרה הארוכה ביותר של תואמת לזו שבסחרוזת היעד. בכל שה-LCS ארוך יותר, בר היפיטנס טוב יותר (או נמוך יותר – תלוי אם זו בעיה מינימיזציה או מקסימיזציה).

בנוסף, היינו אמורים לתת בonus עבור אותיות הממוקמות במקומות המדויק שלן בחרוזת היעד (כלומר, לא רק שהאות מופיעה ברצף, אלא גם במקומם המקורי).

2. להשוות את היריסטיות החדשה מול היריסטיות המקורי:

להריץ את האלגוריתם המקורי בשני המקרים: פעם אחת עם היריסטיות המקורי, ופעם שנייה עם LCS. בעת היינו צריכים לבדוק האם היריסטיות החדשה משפרת את קצב ההתכנסות/איכות הפתרונות.

3. לבצע טיב פרמטרים:

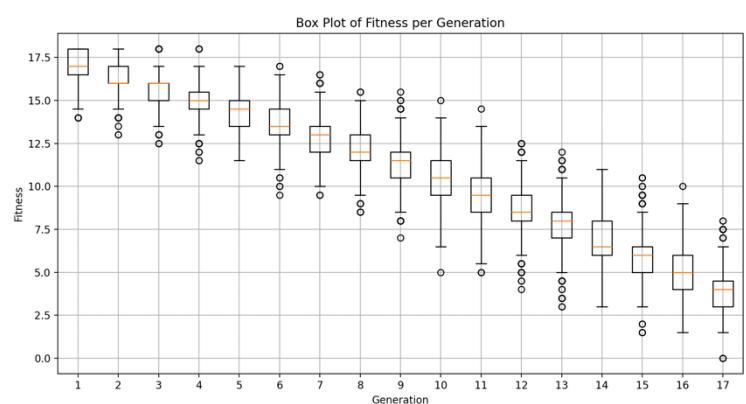
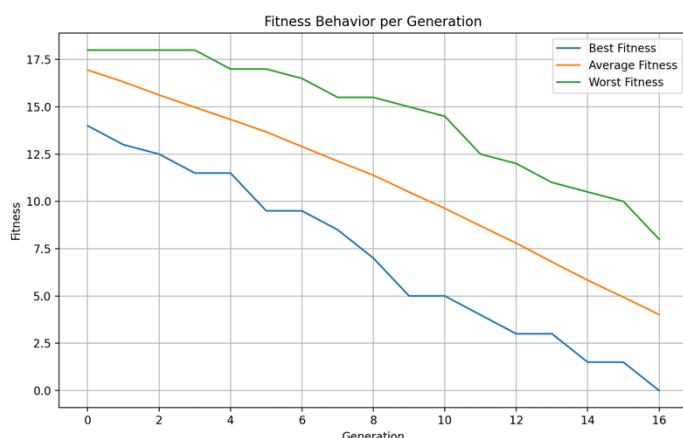
לבחור פרמטרים (כמו גודל אוכלוסייה, קצב מוטציה, גודל bonus למקומות מדויק וכו') ברשייניקו ביצועים מיטביים. להשוות את האיכות (Fitness סופי) ואת מהירות ההגעה לפתרון.

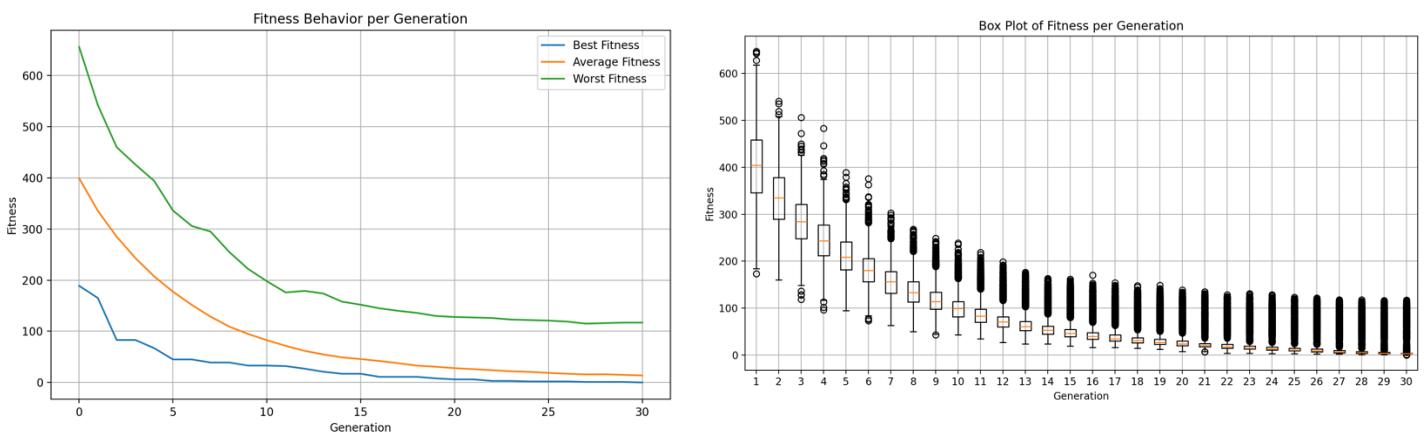
להלן חלק מהקוד שהוספנו על מנת לחשב את הפיטנס ואת ה-LCS:

```
# ----- Task 7 -----
def calculate_fitness_lcs(self):
    """New fitness based on LCS with offset adjustment:
       fitness = (len(GA_TARGET) - LCS_length) - (GA_BONUS_FACTOR * number of exact matches)
                  + (GA_BONUS_FACTOR * len(GA_TARGET))
    """
    lcs = lcs_length(self.string, GA_TARGET)
    bonus = sum(1 for i in range(len(GA_TARGET)) if self.string[i] == GA_TARGET[i])
    offset = GA_BONUS_FACTOR * len(GA_TARGET)
    self.fitness = (len(GA_TARGET) - lcs) - (GA_BONUS_FACTOR * bonus) + offset
```

```
def lcs_length(s, t):
    """Compute the length of the Longest Common Subsequence between s and t."""
    m, n = len(s), len(t)
    dp = [[0]*(n+1) for _ in range(m+1)]
    for i in range(m):
        for j in range(n):
            if s[i] == t[j]:
                dp[i+1][j+1] = dp[i][j] + 1
            else:
                dp[i+1][j+1] = max(dp[i+1][j], dp[i][j+1])
    return dp[m][n]
```

והגרפים לאחר הריצת ה-LCS:





ניתוח התוצאות שהתקבלו:

1. שיפור בקצב ההתקננות:
המגרף רואים שהשימוש ב-LCS גורם לירידה מהירה יותר של הפיטנס בהשוואה לגרסה המקורית. בשלבים הראשונים, האלגוריתם צובר "נקודות" מהר יותר כי עצם הופעתן של אותיות בסדר הנכון (גם אם לא בדיק באותו מיקום) מזכה את הפרט בиндקס חיובי גבוה יחסית.
2. הגעה למינרעה במספר דורות קטן יותר:
נראה שב相较ויה ליריסטייקה הקודמת, עם LCS האלגוריתם מגע לפתרון אופטימלי או מתקדם אליו אחרי פחות דורות (או בזמן ריצה קצר יותר). בגרף רואים בבירור שעם LCS יש "קפיצות" מהירות יותר ברמת הפיטנס מאשר עם היריסטייקה המקורית.
3. בניית מיקום מדויק:
בגרף של LCS עשוי להופיע פתאום "שיפור חד" בשפרטים מצחילים לייצר יותר אותיות בדיק למיקום הנכון (למשל בין דורות 9-6). הדבר מוגרש במיוחד כשהם המוציאה וגם השחלוף מתחילה ליציר יותר אותיות במקומן האמתי, והפיטנס משתפר בקצבים גדולים.

מסקנות מהמשימה

1. היריסטייקה LCS יכולה לשפר ביצועים:
שימוש בה מאפשר "לבון" את האלגוריתם כך שיזהה נכון נוכן רצפים משותפים ארוכים יותר. באשר מוסיפים בונוס למיקום מדויק, מקבלים למעשה שקלולiesel ויתר בין "האם האות קיימת לפני הסדר" לבין "האם היא גם נמצאת במקום הנכון". הדבר מקדם פתרונות טובים מהר יותר, בהשוואה לשיטת חישוב פיטנס מקורית (למשל).
2. קשר לשחלופים (Crossover):
על פה מטה, נתבונן גם להסביר כיצד השחלוף מgeb ל-LCS: מכיוון ששחלוף משלב רצפי אותיות מהWORDS שונים, היריסטייקה של LCS "מגיבה" היטוב לרצפים (Subsequences) שיופיעו במהלך אחרת. אם להוור אחד יש רצף נכון ולהוור השני רצף אחר נכון, השחלוף עשוי לייצר צאצא שמכיל יותר אותיות תואמות עד ברצף הנכון.
3. טיב פרמטרים:
הចורך בבחירה גדול בונוס מדויק למיקום, שיעור מוטציה, גודל האוכלוסייה וכו', בא לידי ביטוי: בגרפים ניתן לראות הבדלים בקצב ההתקננות וברמת הפיטנס הסופית.

במשימה זו התקשנו:

1. למדוד את עצמת לחץ הבחירה בתחום האוכלוסייה בכל דוח, כלומר עד כמה האלגוריתם "معدיף" את הפרטים הטובים ומדכו את הפרטים הגרועים.
2. לעשות זאת באמצעות שני מדרדים עיקריים:
 - Fitness Variance
 - Top-Average Selection Probability Ratio

להלן הקוד שלנו, ולאחר מכן נסביר מה עושים בו:

```
# ----- Task 8: Selection Pressure Metrics -----
adjusted = [worst.fitness - ind.fitness for ind in population]
mean_adjusted = sum(adjusted) / len(adjusted)
std_adjusted = statistics.stdev(adjusted)
selection_variance = std_adjusted / mean_adjusted if mean_adjusted != 0 else 0
total_adjusted = sum(adjusted)
if total_adjusted == 0:
    probabilities = [1.0 / len(population)] * len(population)
else:
    probabilities = [val / total_adjusted for val in adjusted]
top_k = max(1, int(0.1 * len(population)))
top_avg = sum(probabilities[:top_k]) / top_k
overall_avg = 1.0 / len(population)
top_avg_ratio = top_avg / overall_avg
print(f" Selection Variance = {selection_variance:.6f}")
print(f" Top-Average Selection Probability Ratio = {top_avg_ratio:.2f}")
```

1. חישוב "Adjusted Fitness"

היעון הוא ללקח את הפיטנס הגראן באוכלוסייה ולהחסיר מכל פרט את הפיטנס שלו. כך, לפחות הטוב ביותר יהיה הערך הגבוה ביותר ב-`adjusted`, ואילו הגראן ביותר יקבל ערך 0 (כי $0 = \text{worst.fitness} - \text{worst.fitness}$). בעת, הערכים ב-`adjusted` יכולים ≤ 0, ואפשר להשתמש בהם לחישוב הסתבריות.

```
adjusted = [worst.fitness - ind.fitness for ind in population]
```

2. Fitness Variance

מחשבים ממוצע וסטיית תקן של `adjusted`. היחס (Std/Mean) משקף את פיזור הפיטנס היחסי בין פרטים. אם היחס גדול, יש שונות גבוהה ולחץ בחירה חזק.

```
mean_adjusted = sum(adjusted) / len(adjusted)
std_adjusted = statistics.stdev(adjusted)
selection_variance = std_adjusted / mean_adjusted if mean_adjusted != 0 else 0
```

3. חישוב לחץ הבחירה:

מוגדים שהסכום לא 0. אם הוא 0 (כל האוכלוסייה שווה לערך הגראן), אז פשוט נתונים הסתברות שוות לכלם. אחרת, כל פרט ≠ מקבל הסתברות פרופורציונית לערכו ב-`adjusted`.

```
total_adjusted = sum(adjusted)
if total_adjusted == 0:
    probabilities = [1.0 / len(population)] * len(population)
else:
    probabilities = [val / total_adjusted for val in adjusted]
```

מניחים שהאוכלוסייה ממוקמת כך שהפרטים הטוביים ביותר נמצאים בהתחלה (או שסידרנו את probabilities לפי סדר הפרטים הći טובים).

ובחרים את 10% המובילים (top_k).

מחשבים את הממוצע שלהם (top_avg), ומשווים לממוצע הכלל.

יחס גבוה (>1) פירשו שהטופ 10% מקבלים סיכוי גדול בהרבה מהסבירו "הגיל", זה מצביע על לחץ סלקטיבי חזק.

```
top_k = max(1, int(0.1 * len(population)))
top_avg = sum(probabilities[:top_k]) / top_k
overall_avg = 1.0 / len(population)
top_avg_ratio = top_avg / overall_avg
```

.5 הדפסת התוצאות:

```
print(f" Selection Variance = {selection_variance:.6f}")
print(f" Top-Average Selection Probability Ratio = {top_avg_ratio:.2f}")
```

להלן דוגמה להדפסות בדור 0 ובדור אחרון (31) של ריצה:

```
Gen 0: Best = '0del`2CZuJb$' (Fitness = 148)
Avg Fitness = 400.19
Std Dev = 77.04
Worst Fitness = 662
Fitness Range = 514
Tick Duration (sec) = 0.0031
Total Elapsed Time (sec) = 0.0158
Selection Variance = 0.294263
Top-Average Selection Probability Ratio = 1.52
```

```
Gen 31: Best = 'Hello World!' (Fitness = 0)
Avg Fitness = 15.89
Std Dev = 27.08
Worst Fitness = 114
Fitness Range = 114
Tick Duration (sec) = 0.0032
Total Elapsed Time (sec) = 0.4304
Selection Variance = 0.276055
Top-Average Selection Probability Ratio = 1.14
```

במשימה זו נתבקשנו:

1. להגדיר ולחשב ממדים מסוימים שיעזרו לעירק את הגיון הgentiy באוכלוסייה בכלל זה.
2. לדוח ערכים אלה לצד פרמטרי האלגוריתם הגנטי הרגילים (Best Fitness, Average Fitness וכו'), כדי לעקוב אחריו אופן התפתחות ה-Exploration (גיון) והאם האוכלוסייה מתכנסת או עדין שומרת על שונות מספקת בין הפרטאים.

בקוד המצורף, ניתן לראות את הקריאה לפונקציה compute_diversity_metrics

```
# ----- Task 9: Genetic Diversity Metrics (Factor Exploration) -----
def compute_diversity_metrics(population):

    L = len(GA_TARGET)
    N = len(population)
    total_hamming = 0.0
    total_distinct = 0
    total_entropy = 0.0

    # For each gene position, compute frequencies
    for j in range(L):
        freq = {}
        for ind in population:
            allele = ind.string[j]
            freq[allele] = freq.get(allele, 0) + 1
        # (a) Average pairwise difference at this position: 1 - sum(p^2)
        pos_entropy_component = 0.0
        pos_p2_sum = 0.0
        for count in freq.values():
            p = count / N
            pos_p2_sum += p * p
            if p > 0:
                pos_entropy_component += -p * math.log2(p)
        avg_diff = 1 - pos_p2_sum # probability two individuals differ at this gene
        total_hamming += avg_diff
        # (b) Number of distinct alleles at this position:
        total_distinct += len(freq)
        # (c) Entropy at this position:
        total_entropy += pos_entropy_component

    # Multiply avg_diff by L to get average Hamming distance per pair (over entire string)
    avg_hamming_distance = total_hamming * L
    avg_distinct = total_distinct / L # average number of distinct alleles per position
    avg_entropy = total_entropy / L # average entropy per position (in bits)

    return avg_hamming_distance, avg_distinct, avg_entropy
```

אשר עשו את השלבים הבאים:

1. איסוף תדריות (Frequencies) לכל מקום גן

שובר כל מקום גן בחרזות (Gene), אנו בודקים אילו אללים (אותיות, ספורות, וכו') הופיעו וכמה פעמים. לדוגמה, אם בבדיקה גן מופיעות אותיות 'c', 'b', 'a' בערכיהם השונים, אז נקלוט במילון freq בכמה פרטאים שונים משתמשים בכל אות.

```
for j in range(L):
    freq = {}
    for ind in population:
        allele = ind.string[j]
        freq[allele] = freq.get(allele, 0) + 1
```

2. Average Pairwise Hamming Distance

מוחשב כ- $\sum_{N}^{count} p^2 - 1$ למיקום גן יחיד, כאשר k הוא ההסתברות לאלל מסויים ($\frac{count}{N}$).

הסכום הזה הוא ההסתברות שני פרטאים שנבחרו אקראית שווים זה לזה באותו מקום גן.

מצטבריםם לכל המיקומים כדי לקבל בסופו "ממוצע Hamming" על פניו כל הגנים באוכלוסייה.

:Average Number of Distinct Alleles per Gene .3

לכל מיקום j , סופרים כמה אללים ייחודיים יש ($len(freq)$).
 $.total_distinct += len(freq)$
 מוסיפים למשתנה סוכם (Gene).
 בסוף עושים ממוצע "לכל מיקום" כך שיודעים במדויק כמה אללים שונים קיימים בכל

$$\text{avg_distinct} = \text{total_distinct} / L$$

:Average Shannon Entropy .4

מחשבים את סכום ($p \cdot -p * \log_2(p)$) על כל האללים באותו מיקום.
 סוכמים בכל המיקומים, ומחלקים ב-L לקבל תאנטרופיה ממוצעת כללית.

```
if p > 0:
    pos_entropy_component += -p * math.log2(p)
total_entropy += pos_entropy_component
avg_entropy = total_entropy / L
```

ערך אנטרופיה גבוה מצביע על התפלגות אחידה למדי (כלומר, הרבה אללים שונים באותו גן בפרופורציות דומות).
 ערך נמוך מצביע על כך שרוב הפרטימים חולקים אלל זהה באותו מיקום זה.

.5. החזרת התוצאות:

בסוף מחשבים ומחזירים:

```
return avg_hamming_distance, avg_distinct, avg_entropy
```

פלט לדוגמא:

```
Gen 0: Best = '0de1`2CZuJb$' (Fitness = 148)
Avg Fitness = 400.19
Std Dev = 77.04
Worst Fitness = 662
Fitness Range = 514
Tick Duration (sec) = 0.0031
Total Elapsed Time (sec) = 0.0158
Selection Variance = 0.294263
Top-Average Selection Probability Ratio = 1.52
Avg Pairwise Hamming Distance = 142.35
Avg Number of Distinct Alleles per Gene = 91.00
Avg Shannon Entropy per Gene (bits) = 6.48
```

```
Gen 31: Best = 'Hello World!' (Fitness = 0)
Avg Fitness = 15.89
Std Dev = 27.08
Worst Fitness = 114
Fitness Range = 114
Tick Duration (sec) = 0.0032
Total Elapsed Time (sec) = 0.4304
Selection Variance = 0.276055
Top-Average Selection Probability Ratio = 1.14
Avg Pairwise Hamming Distance = 40.03
Avg Number of Distinct Alleles per Gene = 34.58
Avg Shannon Entropy per Gene (bits) = 0.83
```

:בהתחלת (Gen 0)

Hamming Distance גבוהה יחסית, מספר אללים ייחודיים גדול מאוד, ואנטרופיה גבוהה (6.48 במדויק).
 משמע שהאוכלוסייה הראשונית מגוונת מאוד: אין יתרון לאללים מסוימים והכל מפוזר באקרואיות.

:בדור 31

Hamming Distance יורד משמעותית, גם מספר האללים הייחודיים בכל מיקום יורד, וכן גם האנטרופיה.
 המשמעות היא שרוב הפרטימים באוכלוסייה קרובים זה לזה (Hamming קטן), יש הרבה פחות אללים מגוונים (34.58 במדויק במקום 91),
 וההתפלגות לא אחידה (Entropy רק 0.83). כי האלגוריתם הוכנס להעדפת אללים מסוימים שהובילו לפתרון.
 זה סימן מובהק לאובדן גיוון גנטי: האוכלוסייה מתכנסת באופן הדוק סביב הפתרון הנכון.

במשימה זו ביקשו מאייתנו להוכיח את האלגוריתם הגנטי ולתמוך במגוון שיטות לבחירת הורים (Parent Selection). בכל תתי-סעיף היה עלינו למשש שיטה אחרת, להריץ את האלגוריתם בכמה קונפיגורציות, ולנתח את התוצאות.

להלן המימוש (שלל הסעיפים), המשתמש יוכל לבחור את השיטה על ידי הזנת מספר בקונסול:

```
# ----- Task 10: Parent Selection Methods -----
def select_parent_RWS(population):
    # Roulette Wheel Selection with linear scaling using adjusted fitness
    worst = max(ind.fitness for ind in population)
    adjusted = [worst - ind.fitness for ind in population]
    total = sum(adjusted)
    if total == 0:
        return random.choice(population)
    r = random.uniform(0, total)
    cum = 0
    for ind, val in zip(population, adjusted):
        cum += val
        if cum >= r:
            return ind
    return population[-1]

def select_parent_TournamentDet(population):
    # Deterministic tournament: choose best among K randomly sampled individuals
    candidates = random.sample(population, GA_TOURNAMENT_K)
    return min(candidates, key=lambda ind: ind.fitness)

def select_parent_TournamentStoch(population):
    # Stochastic tournament: sample K individuals, sort by fitness, and select
    candidates = random.sample(population, GA_TOURNAMENT_K)
    candidates.sort(key=lambda ind: ind.fitness)
    for candidate in candidates:
        if random.random() < GA_TOURNAMENT_P:
            return candidate
    return candidates[-1]

def select_parents_SUS(population, num_parents):
    worst = max(ind.fitness for ind in population)
    adjusted = [worst - ind.fitness for ind in population]
    total = sum(adjusted)
    if total == 0:
        return [random.choice(population) for _ in range(num_parents)]
    step = total / num_parents
    start = random.uniform(0, step)
    pointers = [start + i * step for i in range(num_parents)]
    parents = []
    for p in pointers:
        cum = 0
        for ind, val in zip(population, adjusted):
            cum += val
            if cum >= p:
                parents.append(ind)
                break
    return parents

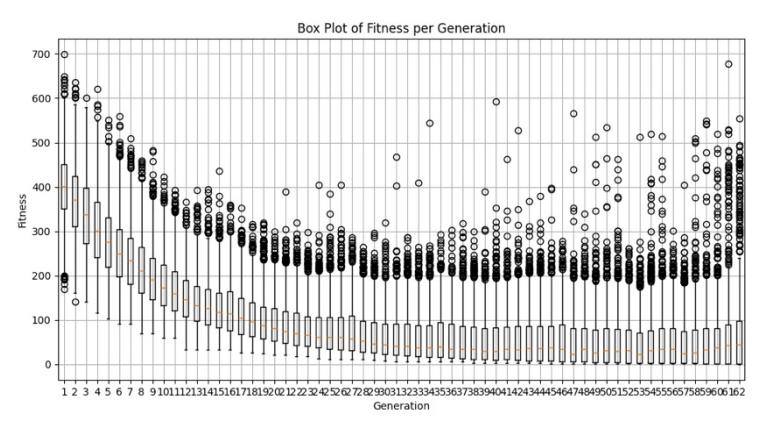
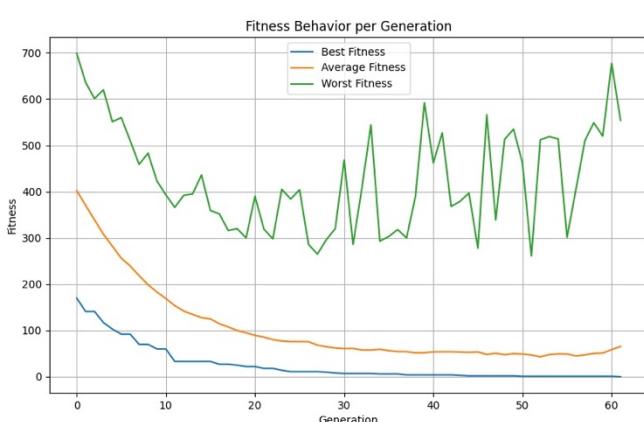
# ----- Task 10: Mating Function with Various Parent Selection Methods -----
def mate(population, buffer):
    esize = int(GA_POPSIZE * GA_ELITRATE)
    elitism(population, buffer, esize)
    num_offspring = GA_POPSIZE - esize
    # For SUS, pre-select all needed parents
    sus_parents = []
    if GA_PARENT_SELECTION_METHOD == "SUS":
        sus_parents = select_parents_SUS(population, num_offspring * 2)
    for i in range(esize, GA_POPSIZE):
        if GA_PARENT_SELECTION_METHOD == "RWS":
            parent1 = select_parent_RWS(population)
            parent2 = select_parent_RWS(population)
        elif GA_PARENT_SELECTION_METHOD == "TournamentDet":
            parent1 = select_parent_TournamentDet(population)
            parent2 = select_parent_TournamentDet(population)
        elif GA_PARENT_SELECTION_METHOD == "TournamentStoch":
            parent1 = select_parent_TournamentStoch(population)
            parent2 = select_parent_TournamentStoch(population)
        elif GA_PARENT_SELECTION_METHOD == "SUS":
            parent1 = sus_parents.pop(0)
            parent2 = sus_parents.pop(0)
        else:
            parent1 = random.choice(population)
            parent2 = random.choice(population)
    # Crossover
    if GA_CROSSOVER_OPERATOR == "SINGLE":
        child_string = crossover_single(parent1, parent2)
    elif GA_CROSSOVER_OPERATOR == "TWO":
        child_string = crossover_two(parent1, parent2)
    elif GA_CROSSOVER_OPERATOR == "UNIFORM":
        child_string = crossover_uniform(parent1, parent2)
    elif GA_CROSSOVER_OPERATOR == "TRIVIAL":
        child_string = crossover_trivial(parent1, parent2)
    else:
        child_string = crossover_single(parent1, parent2)
    child = GAIIndividual(child_string)
    if random.random() < GA_MUTATIONRATE:
        child.mutate()
    buffer.append(child)

# ----- Task 10: Aging Survivor Selection -----
def apply_aging(population):
    survivors = []
    for ind in population:
        ind.age += 1
        if ind.age < GA_MAX_AGE:
            survivors.append(ind)
    # Fill population with new random individuals if needed
    while len(survivors) < GA_POPSIZE:
        new_ind = GAIIndividual()
        new_ind.age = 0
        survivors.append(new_ind)
    return survivors
```

RWS + Linear Scaling (א)

חלקה פרופורציונית (Fitness-Proportional) של "גלאג" לבחירת הורים. ככל שהפיטנס גבוה יותר, כך גדול סיכוי הפרט להיבחר. טרנספורמציה ליניארית על ערכי הפיטנס הגלומיים (Raw Fitness) כדי למנוע מצב שפרט אחד עם פיטנס גבוה במיוחד "משתלט" על הגלג. לדוגמה, $f'(x) = a * f(x) + b$

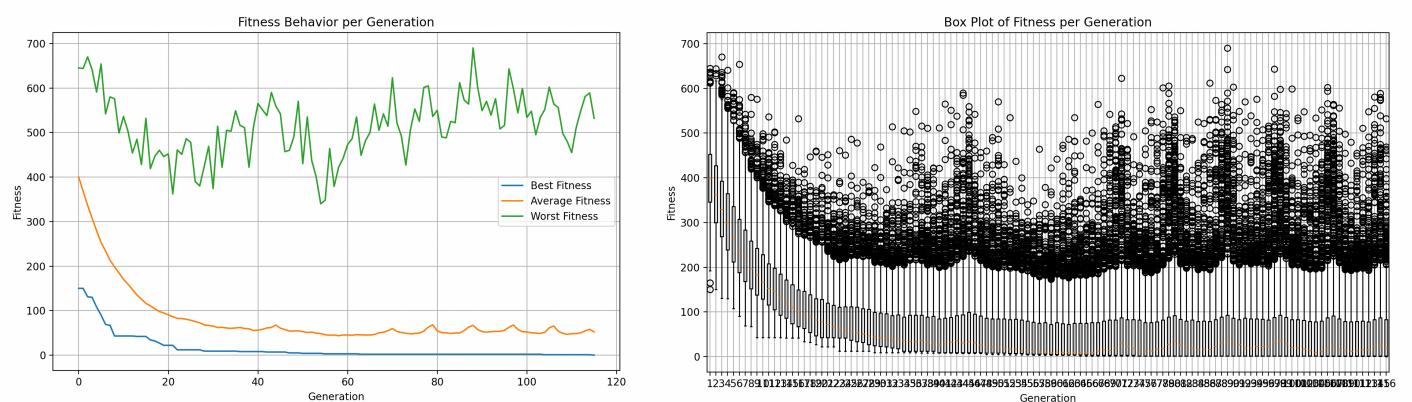
הרעין הוא לשלווט בערכי הפיטנס כך שההפרשים יהיו סבירים, ולשמור על Selection Pressure מסוון.



בשיטת הזו, מנגנון האלטיזם שומר על הפתרון הטוב לאורך כל הריצה, אך במקביל לשמור גיוון רחב – מה שמתבטא בשונות גובהה בערכי הפיטנס הממצאים והגראויים. כאשר האלגוריתם נתקל במצב של "לכידה" או "נטקעות" (כמו בדור 25 ~ למשל), מנגנונים כמו הגדלת שיעור המוציאה או הכנסת פרטאים אקרים מעודדים חיפוש נרחב יותר למרחב הפתרונות, ובכך מאפשרים גילוי של פתרונות חדשים ושיפור נוספים. באופן זה, האלגוריתם מצליח לשמר על הפתרון הטוב הקיים תוך כדי שמירה על חקירה מתמשכת, שmbiah בסופה של דבר לשיפור בתוצאות.

(ב) SUS + Linear Scaling

SUS: שיטה דומה ל-RWS, אבל במקום "להגריל" כמה פעמים בנפרד, מסמנים כמה פוינטרים (Pointers) על "גלגלי" הסיביות במרקחים שווים זה מזה. **Linear Scaling:** כמו בסעיף א', מפעלים סקלילינג ליניארי לפיטנס כדי למנוע פערים קיצוניים. **SUS נוטה להעניק "מדגם הוגן"** יותר של האוכולוסיה – פחות סטיות מקריות מאשר RWS וgilah.



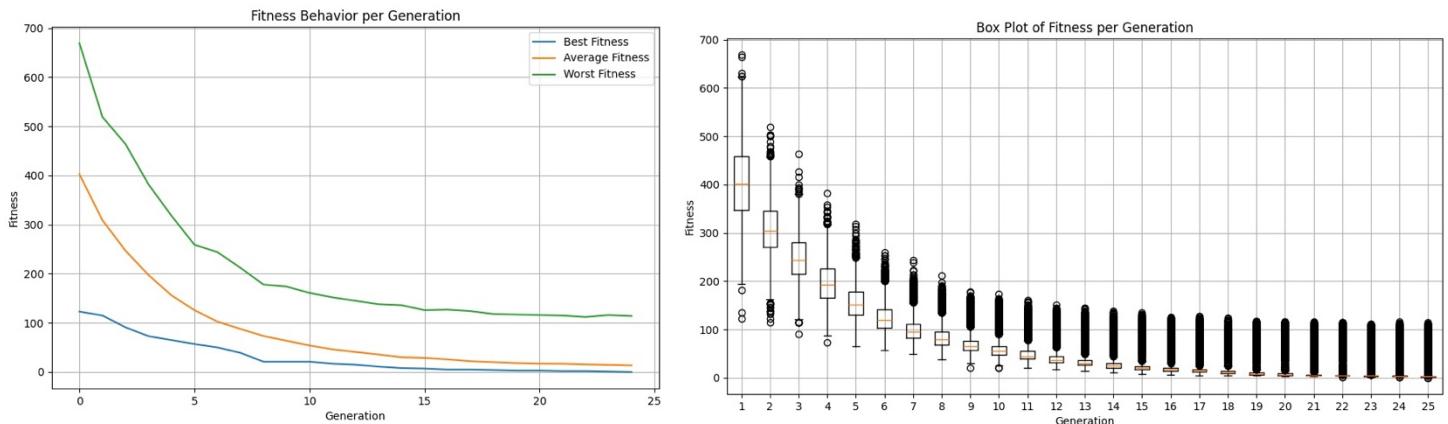
לאורך כ-120 דורות ניתן לראות ירידת עקבית בערך ה-Best Fitness (הקו הכהול), מה שמצוין על שיפור הדרגתית ומהיר באיכות הפתרון עד להתקנות סביר ערך טוב. ה-Average Fitness (הקו הכתום) הולך ומתקרב ל-Best, מה שמעיד על כך שרוב הפרטאים באוכולוסיה מתקרבים לפתרון מיטבי. במקביל, ה-Worst Fitness (הקו הירוק) עדין "רועל" יחסית, ומצוין על הימצאותם של פרטאים בעלי ערכים גורועים יותר – נראה עקב מוטציה או החדרת פרטאים אקרים. באיזור דור 70, ניתן לראות את ה-Average Fitness מתנהга בצורה "גלית" – עקב ה-exploration הגדל.

ב-Box Plot רואים כי חלק גדול מהאוכולוסיה נמצא בטוחה נמוך (מעיד על פתרונות טובים), אך עדין ישנים ערכיהם ("Outliers") גבוהים. משמעות הדבר היא שהאלגוריתם ממשיר לשמר על Exploration גם בשלב מאוחר, אף על פי שהתכנס לפתרון מוצלח כבר בסביבות דור 50.

(המשך בדף הבא)

(ג) טורניר דטרמיניסטי עם פרמטר K –Fitness Ranking

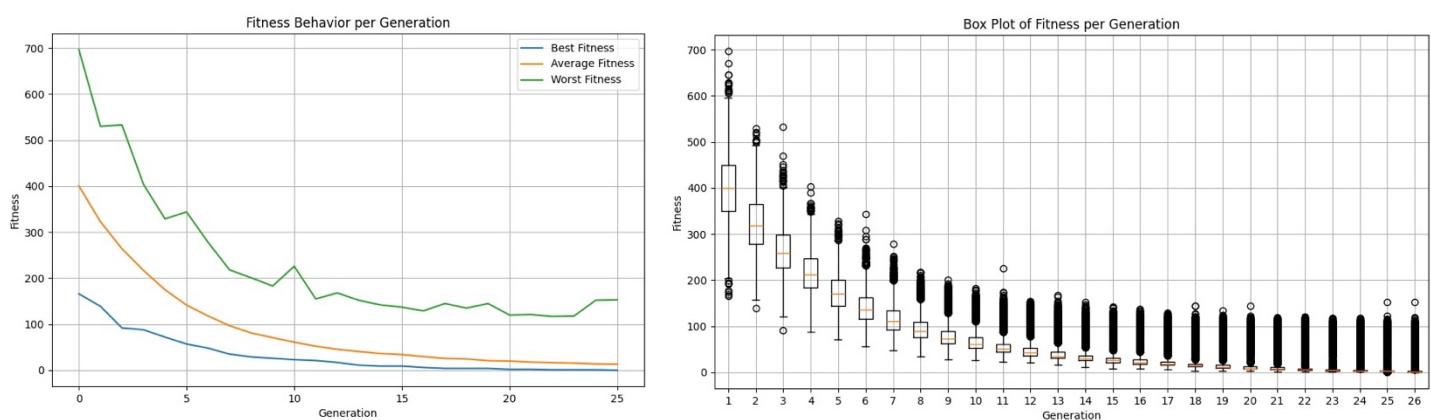
Deterministic Tournament: בוחרים אקרואית K פרטיטים ומתוכם לוקחים תמיד את הפרט בעל הפיטנס הטוב ביותר להוּהה. Fitness Ranking: במקומם להשתמש בפיטנס גלמי, מדורגים את האוכלוסייה וממירים את הדירוג לערכי פיטנס. (למשל, הטוב ביותר = Rank 1) – דבר המונע התפוצצות ערכבים או הבדלי פיטנס דרמטיים. Selection Pressure (סיכוי גבוה יותר שהטוב ביותר בכל טורניר ינצח). ככל ש-K גדול יותר, גדל ה-Selection Pressure.



הגרף הליניארי מראה שבזרות הראשוניים יש ירידה חזקה בכל שלושת הערכים, מה שמעיד על שיפור מהיר באיכות הפתרונות. בהמשך, הفور בין הפתרון הטוב לגרוע מציגים באופן ממשוני, כך שלקראת דור 20–25 כולם כבר קרובים לערך נמוך. Box Plot ניכרת בהתחלת התפלגות רחבה מאוד, עם Outliers רבים, ואילו לקראת הדורות המאוחרים התיבת הופכת צרה והדגם נע סביב ערך נמוך ואחד. המשמעות היא שהאוכלוסייה נעשתה הומוגנית יותר – כולם קרובים לפתרון טוב מאוד. התוצאה הסופית היא התכונות מהירה תוך כמה עשרות דורות, כאשר הפרט הטוב והאוכלוסייה כולה מגיעים לערכי פיטנס דומים ונמוכים, המעידים על פתרון מיטבי או קרוב אליו.

(ד) טורניר לא דטרמיניסטי עם פרמטר P –Fitness Ranking

עדין בוחרים K פרטיטים באקראי, אבל לא תמיד לוקחים את הטוב ביותר. במקום זאת, הפרט הטוב עשוי להיות הבחירה בנסיבות k , ופרט חלש יותר עשוי להיות הבחירה בנסיבות $k - 1$. שומר על קצת יותר Exploration, מכיוון שגם הפרטיטים הפחות טובים יכולים להיות נבחרים לעיתים.



הגרף הליניארי מראה ירידה חזקה בשזרות המודדים בזרות הראשוניים, מה שמעיד על התכונות מהירה להתחלת של פתרון טוב. בהמשך, קצב השיפור אمنם מואט, אך ה-Best ממשיר לרדת והאוכלוסייה בכללתה מתקרבת אליו. כן ניתן לראות שהאלג' מבצע exploration גם בזרות מתקדמים.

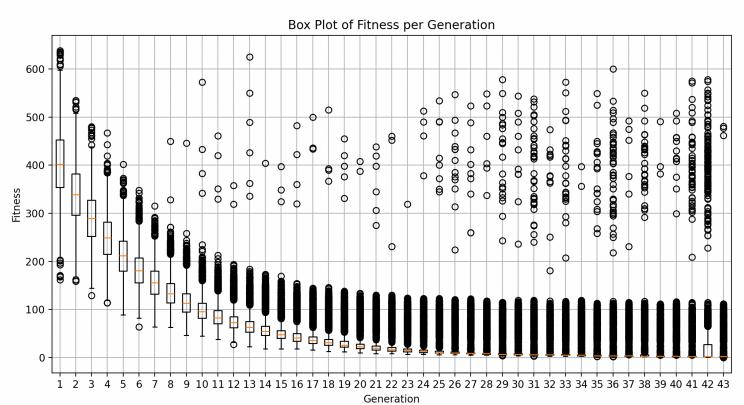
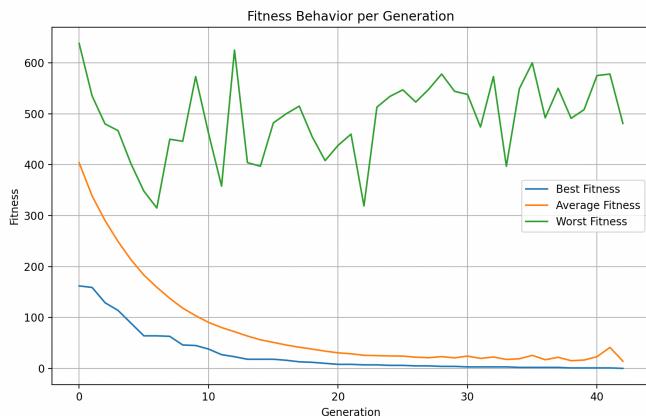
Box Plot ממחיש שההתפלגות הפיטנס בתחילת רחבה מאוד (רבים מהפרטיטים גרוועים), אך תוך מספר דורות התיבת הופכת אופן דרמטי. עד דור 25 לפחות, האוכלוסייה כבר כמעט הומוגנית סביב ערך פיטנס נמוך, מה שמעיד על פתרון יציב וטוב.

(ה) Aging / Survivor Selection

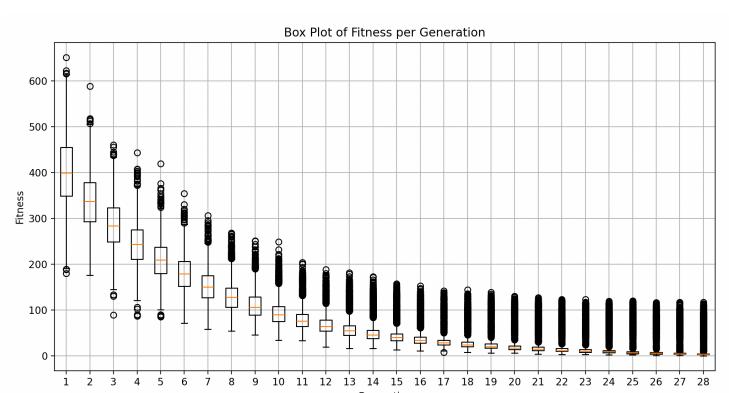
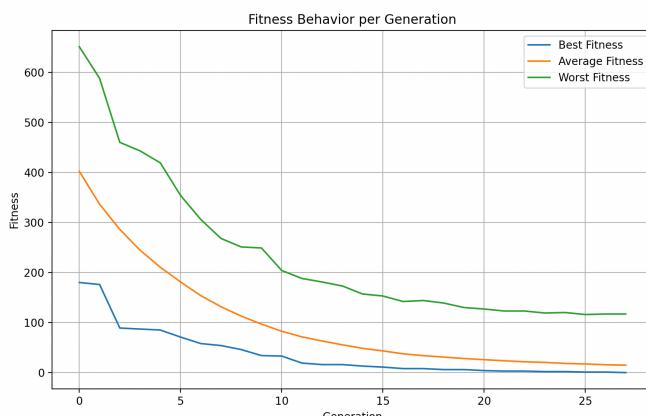
לכל פרט מצמידים "גיל" (מספר דורות שהיה באוכלוסייה).

מגדירים Max Age – בשחרprt חורג ממנו, הוא נדחה אוטומטי מהתבנית, גם אם הפיטנס שלו מעולם. שיטה זו מוסיפה למנגנון "חילופי דורות" המבטיח תחלופה מתמדת ומונעת מפרטים ותיקים, חזקים ככל שהוא, להישאר לנצח ולהקפיא את האוכלוסייה.

להלן תוצאות של aging שערכו 5:



להלן תוצאות של aging שערכו 15:



מההשוואה בין שני הגרפים (aging=15 לעומת aging=5) עולה שהגדלת ערך ה-*aging* מובילה לייצוב ממשמעותי יותר של האוכלוסייה ושימור פתרונות טובים למשך זמן. ב-*aging*=5, פרטים מוחלפים בתדרות גבוהה יותר, ולכן האלגוריתם מציג חקירה (Exploration) מוגברת אך גם "רעש" גבוה יותר: התכנית Worst Fitness נוטרת גבולה יחסית, והאוכלוסייה מתקשה להתכנס במהירות. לעומת זאת, ב-*aging*=15, הפרטים מוחלפים בתדרות נמוכות באוכלוסייה זמן רב יותר, וכך שהפרטים הטובים מספקים לתהבות ולשמור תכונות חייבות. בתוצאה מכון, ה-*Best Fitness* יורד (או עולה, בהתאם לסוג הבעה) מהר יותר, והאוכלוסייה מתיצבת סביב פתרון איבוטי יותר, עם שונות (Variance) קטנה יותר בכל דור. מבון שלא משתמש מכך שככל שה-*aging* גדול יותר, כך התוצאה טובה יותר.

בתרגיל זה יישמו שתי גישות לפתרון בעיית Bin Packing:

First Fit: אלגוריתם גריידי פשוט שמקם כל פריט במיכל הראשון שבו הוא מתאים, לרוב בסדר יורד – פעולה שהיא מהירה מאוד. אלגוריתם גנני: גישה אבולוציונית שבה אנו מפעילים אוכלוסייה של פתרונות, מבצעים בחירות הורים, חיתוכים ומוטציות כדי לשפר את הפתרונות לאורך דורות רבים, עם מנגנון אליטיזם והזדקנות.

לאחר הריצה על 5 בעיות שונות מהקובץ BinPack1.txt, יצא לנו שה First Fit עבד הבהה יותר מהר. נראה שזה נובע מכיוון שהאלגוריתם הגריידי מבצע פעולה ישירה שמקמת כל פריט בהתאם על תנאים פשוטים, ללא חיפוש רחב ועמוק – וכן הוא מהיר מאוד. לעומת זאת, האלגוריתם הגנני מחשב מספר עצום של פתרונות (אוכלוסייה), דורש ביצוע חישובי פיטנס, חיתוכים, מוטציות ועוד, מה שמנגדיל את במתה החישובים לאורך הבהה דורות.

להלן דוגמאות הריצה של ה- FF (מצד שמאל) לעומת הגנני (מצד ימין):

```
Results for u120_00:
Bins used: 49
Theoretical minimum: 48
Deviation from optimal: 1 bins
Runtime: 0.0006 seconds

Bin details:
Bin 1: [98, 50] (Total: 148/150)
Bin 2: [98, 49] (Total: 147/150)
Bin 3: [98, 49] (Total: 147/150)
Bin 4: [96, 49] (Total: 145/150)
Bin 5: [96, 47] (Total: 143/150)
Bin 6: [94, 55] (Total: 149/150)
Bin 7: [93, 57] (Total: 150/150)
Bin 8: [93, 57] (Total: 150/150)
Bin 9: [92, 58] (Total: 150/150)
Bin 10: [91, 59] (Total: 150/150)
Bin 11: [91, 58] (Total: 149/150)
Bin 12: [90, 60] (Total: 150/150)
Bin 13: [87, 62] (Total: 149/150)
Bin 14: [86, 64] (Total: 150/150)
Bin 15: [85, 62] (Total: 147/150)
Bin 16: [85, 60] (Total: 145/150)
Bin 17: [84, 66] (Total: 150/150)
Bin 18: [84, 58] (Total: 142/150)
Bin 19: [84, 57] (Total: 141/150)
Bin 20: [84, 57] (Total: 141/150)
Bin 21: [84, 55] (Total: 139/150)
Bin 22: [83, 67] (Total: 150/150)
Bin 23: [83, 55] (Total: 138/150)
Bin 24: [82, 46, 28] (Total: 148/150)
Bin 25: [82, 46] (Total: 128/150)
Bin 26: [81, 69] (Total: 150/150)
Bin 27: [80, 70] (Total: 150/150)
Bin 28: [80, 70] (Total: 150/150)
Bin 29: [80, 70] (Total: 150/150)
Bin 30: [79, 71] (Total: 150/150)
Bin 31: [79, 69] (Total: 148/150)
Bin 32: [78, 72] (Total: 150/150)
Bin 33: [78, 69] (Total: 147/150)
Bin 34: [78, 45, 27] (Total: 150/150)
Bin 35: [78, 45, 27] (Total: 150/150)
Bin 36: [76, 74] (Total: 150/150)
Bin 37: [74, 73] (Total: 147/150)
Bin 38: [73, 73] (Total: 146/150)
Bin 39: [73, 44, 33] (Total: 150/150)
Bin 40: [44, 43, 43] (Total: 130/150)
Bin 41: [43, 43, 42] (Total: 128/150)
Bin 42: [42, 42, 42, 24] (Total: 150/150)
Bin 43: [42, 41, 41, 26] (Total: 150/150)
Bin 44: [41, 39, 39, 30] (Total: 149/150)
Bin 45: [38, 38, 38, 36] (Total: 150/150)
Bin 46: [37, 36, 36, 35] (Total: 144/150)
Bin 47: [33, 33, 32, 32] (Total: 130/150)
Bin 48: [30, 30, 29, 28, 25] (Total: 142/150)
Bin 49: [25, 23, 23] (Total: 71/150)
```

```
Results for u120_00:
Bins used: 49
Theoretical minimum: 48
Deviation from optimal: 1 bins
Runtime: 35.1651s

Bin details:
Bin 1: [70, 41, 33] (Total: 144/150)
Bin 2: [42, 38, 38, 32] (Total: 150/150)
Bin 3: [98, 49] (Total: 147/150)
Bin 4: [71, 43, 36] (Total: 150/150)
Bin 5: [87, 62] (Total: 149/150)
Bin 6: [70, 43, 37] (Total: 150/150)
Bin 7: [79, 41, 30] (Total: 150/150)
Bin 8: [86, 57] (Total: 143/150)
Bin 9: [50, 49, 49] (Total: 148/150)
Bin 10: [57, 84] (Total: 141/150)
Bin 11: [98, 26, 28] (Total: 144/150)
Bin 12: [78, 67] (Total: 145/150)
Bin 13: [91, 55] (Total: 146/150)
Bin 14: [83, 60] (Total: 143/150)
Bin 15: [73, 70] (Total: 143/150)
Bin 16: [91, 46] (Total: 137/150)
Bin 17: [82, 57] (Total: 143/150)
Bin 18: [94, 46] (Total: 140/150)
Bin 19: [84, 66] (Total: 150/150)
Bin 20: [76, 69] (Total: 145/150)
Bin 21: [93, 38] (Total: 131/150)
Bin 22: [85, 60] (Total: 145/150)
Bin 23: [80, 57] (Total: 137/150)
Bin 24: [78, 29, 42] (Total: 149/150)
Bin 25: [93, 27, 25] (Total: 145/150)
Bin 26: [74, 69] (Total: 143/150)
Bin 27: [96, 47] (Total: 143/150)
Bin 28: [92, 58] (Total: 150/150)
Bin 29: [81, 69] (Total: 150/150)
Bin 30: [78, 42, 24] (Total: 144/150)
Bin 31: [82, 58] (Total: 140/150)
Bin 32: [80, 35, 33] (Total: 148/150)
Bin 33: [84, 41, 23] (Total: 148/150)
Bin 34: [73, 36, 33] (Total: 142/150)
Bin 35: [80, 62] (Total: 142/150)
Bin 36: [84, 64] (Total: 146/150)
Bin 37: [98, 28, 23] (Total: 149/150)
Bin 38: [96, 42] (Total: 138/150)
Bin 39: [73, 43, 27] (Total: 143/150)
Bin 40: [83, 36, 30] (Total: 149/150)
Bin 41: [78, 72] (Total: 150/150)
Bin 42: [79, 58] (Total: 137/150)
Bin 43: [85, 39] (Total: 124/150)
Bin 44: [45, 73, 30] (Total: 148/150)
Bin 45: [55, 44, 43] (Total: 142/150)
Bin 46: [57, 84] (Total: 141/150)
Bin 47: [74, 39, 32] (Total: 145/150)
Bin 48: [44, 55, 45] (Total: 144/150)
Bin 49: [90, 59] (Total: 149/150)
```

אלגוריתם FF לבעה 00: U120_00

מצא פתרון במרקח 1 מואופטימלי,
לקח לו 0.0006 שניות.

```
Results for u120_01:
Bins used: 49
Theoretical minimum: 49
Deviation from optimal: 0 bins
Runtime: 0.0007 seconds

Bin details:
Bin 1: [100, 50] (Total: 150/150)
Bin 2: [100, 49] (Total: 149/150)
Bin 3: [99, 49] (Total: 148/150)
Bin 4: [99, 48] (Total: 147/150)
Bin 5: [98, 52] (Total: 150/150)
Bin 6: [98, 52] (Total: 150/150)
Bin 7: [98, 48] (Total: 146/150)
Bin 8: [99, 47] (Total: 145/150)
Bin 9: [99, 47] (Total: 145/150)
Bin 10: [97, 53] (Total: 150/150)
Bin 11: [97, 53] (Total: 150/150)
Bin 12: [97, 53] (Total: 150/150)
Bin 13: [95, 55] (Total: 150/150)
Bin 14: [95, 55] (Total: 150/150)
Bin 15: [95, 53] (Total: 148/150)
Bin 16: [94, 53] (Total: 147/150)
Bin 17: [92, 58] (Total: 150/150)
Bin 18: [90, 60] (Total: 150/150)
Bin 19: [90, 60] (Total: 150/150)
Bin 20: [88, 62] (Total: 150/150)
Bin 21: [88, 61] (Total: 149/150)
Bin 22: [85, 65] (Total: 150/150)
Bin 23: [82, 68] (Total: 150/150)
Bin 24: [81, 67] (Total: 148/150)
Bin 25: [81, 66] (Total: 148/150)
Bin 26: [81, 67] (Total: 148/150)
Bin 27: [80, 70] (Total: 150/150)
Bin 28: [80, 70] (Total: 150/150)
Bin 29: [80, 70] (Total: 150/150)
Bin 30: [79, 71] (Total: 150/150)
Bin 31: [79, 67] (Total: 146/150)
Bin 32: [78, 72] (Total: 150/150)
Bin 33: [78, 72] (Total: 150/150)
Bin 34: [76, 74] (Total: 150/150)
Bin 35: [75, 75] (Total: 150/150)
Bin 36: [66, 66] (Total: 132/150)
Bin 37: [65, 64, 20] (Total: 149/150)
Bin 38: [61, 60, 29] (Total: 150/150)
Bin 39: [59, 57, 33] (Total: 149/150)
Bin 40: [57, 57, 36] (Total: 150/150)
Bin 41: [53, 47, 46] (Total: 146/150)
Bin 42: [46, 45, 45] (Total: 136/150)
Bin 43: [45, 44, 43] (Total: 132/150)
Bin 44: [43, 43, 41, 23] (Total: 150/150)
Bin 45: [39, 39, 39, 35] (Total: 145/150)
Bin 46: [38, 38, 37, 36] (Total: 140/150)
Bin 47: [36, 36, 30, 30] (Total: 131/150)
Bin 48: [29, 27, 27, 27, 25] (Total: 135/150)
Bin 49: [24, 23, 22, 22, 20] (Total: 133/150)
```

אלגוריתם FF לבעה 01: U120_01

מצא פתרון אופטימלי, لكח לו
0.0007 שניות.

```
Results for u120_01:
Bins used: 50
Theoretical minimum: 49
Deviation from optimal: 1 bins
Runtime: 17.5616s

Bin details:
Bin 1: [30, 41, 67] (Total: 138/150)
Bin 2: [81, 61] (Total: 142/150)
Bin 3: [97, 32, 20] (Total: 149/150)
Bin 4: [92, 33, 25] (Total: 150/150)
Bin 5: [90, 52] (Total: 150/150)
Bin 6: [65, 74] (Total: 147/150)
Bin 7: [65, 65] (Total: 144/150)
Bin 8: [55, 65, 29] (Total: 150/150)
Bin 9: [70, 76] (Total: 146/150)
Bin 10: [39, 64, 36] (Total: 139/150)
Bin 11: [99, 38] (Total: 137/150)
Bin 12: [58, 78, 29] (Total: 150/150)
Bin 13: [53, 72, 23] (Total: 148/150)
Bin 14: [79, 67] (Total: 146/150)
Bin 15: [98, 35, 24] (Total: 149/150)
Bin 16: [68, 45, 30] (Total: 143/150)
Bin 17: [53, 59, 38] (Total: 150/150)
Bin 18: [100, 49] (Total: 149/150)
Bin 19: [97, 49] (Total: 146/150)
Bin 20: [58, 55, 36] (Total: 149/150)
Bin 21: [44, 70, 29] (Total: 143/150)
Bin 22: [48, 94] (Total: 142/150)
Bin 23: [43, 69, 39] (Total: 142/150)
Bin 24: [80, 60, 37] (Total: 143/150)
Bin 25: [58, 52] (Total: 150/150)
Bin 26: [97, 33] (Total: 150/150)
Bin 27: [77, 75] (Total: 146/150)
Bin 28: [88, 53] (Total: 141/150)
Bin 29: [69, 80] (Total: 150/150)
Bin 30: [37, 46, 33, 27] (Total: 143/150)
Bin 31: [98, 39] (Total: 137/150)
Bin 32: [72, 75] (Total: 147/150)
Bin 33: [65, 38] (Total: 145/150)
Bin 34: [81, 47, 22] (Total: 150/150)
Bin 35: [98, 48] (Total: 146/150)
Bin 36: [81, 62] (Total: 143/150)
Bin 37: [58, 45, 27, 22] (Total: 144/150)
Bin 38: [23, 57, 47, 22] (Total: 149/150)
Bin 39: [86, 67] (Total: 147/150)
Bin 40: [79, 47] (Total: 126/150)
Bin 41: [82, 67] (Total: 149/150)
Bin 42: [198, 45] (Total: 144/150)
Bin 43: [100, 45, 57] (Total: 145/150)
Bin 44: [195, 53] (Total: 146/150)
Bin 45: [195, 46] (Total: 141/150)
Bin 46: [85, 61] (Total: 146/150)
Bin 47: [80, 45] (Total: 125/150)
Bin 48: [99, 27] (Total: 126/150)
Bin 49: [78, 57] (Total: 135/150)
Bin 50: [53, 95] (Total: 148/150)
```

אלגוריתם גנני לבעה 01: U120_01

מצא פתרון במרקח 1 מואופטימלי,
לקח לו 17.5615 שניות.

algorthim geneti lbeua u120_02:

Results for u120_02:
Bins used: 47
Theoretical minimum: 46
Deviation from optimal: 1 bins
Runtime: 0.0006 seconds
Bin details:
Bin 1: [100, 50] (Total: 150/150)
Bin 2: [100, 50] (Total: 150/150)
Bin 3: [100, 51] (Total: 149/150)
Bin 4: [99, 52] (Total: 150/150)
Bin 5: [97, 53] (Total: 150/150)
Bin 6: [96, 51] (Total: 147/150)
Bin 7: [94, 48] (Total: 142/150)
Bin 8: [92, 57] (Total: 149/150)
Bin 9: [92, 48] (Total: 140/150)
Bin 10: [91, 59] (Total: 150/150)
Bin 11: [91, 59] (Total: 150/150)
Bin 12: [90, 60] (Total: 150/150)
Bin 13: [90, 48] (Total: 138/150)
Bin 14: [90, 47] (Total: 137/150)
Bin 15: [88, 51] (Total: 140/150)
Bin 16: [86, 60] (Total: 150/150)
Bin 17: [84, 66] (Total: 150/150)
Bin 18: [84, 66] (Total: 150/150)
Bin 19: [84, 66] (Total: 150/150)
Bin 20: [83, 67] (Total: 150/150)
Bin 21: [81, 69] (Total: 150/150)
Bin 22: [81, 69] (Total: 150/150)
Bin 23: [80, 70] (Total: 150/150)
Bin 24: [80, 68] (Total: 148/150)
Bin 25: [80, 68] (Total: 148/150)
Bin 26: [80, 67] (Total: 147/150)
Bin 27: [79, 67] (Total: 146/150)
Bin 28: [79, 67] (Total: 146/150)
Bin 29: [76, 64] (Total: 143/150)
Bin 30: [76, 64] (Total: 150/150)
Bin 31: [76, 73] (Total: 149/150)
Bin 32: [75, 75] (Total: 150/150)
Bin 33: [64, 64, 22] (Total: 150/150)
Bin 34: [64, 64, 21] (Total: 149/150)
Bin 35: [62, 61, 27] (Total: 150/150)
Bin 36: [61, 46, 42] (Total: 149/150)
Bin 37: [46, 46, 45] (Total: 137/150)
Bin 38: [45, 44, 42] (Total: 131/150)
Bin 39: [41, 41, 40, 28] (Total: 150/150)
Bin 40: [38, 38, 38, 36] (Total: 150/150)
Bin 41: [37, 37, 37, 37] (Total: 148/150)
Bin 42: [36, 36, 37, 41] (Total: 149/150)
Bin 43: [34, 33, 32, 32] (Total: 131/150)
Bin 44: [32, 31, 31, 30, 26] (Total: 150/150)
Bin 45: [29, 29, 29, 29, 28] (Total: 144/150)
Bin 46: [26, 25, 24, 24, 23, 23] (Total: 145/150)
Bin 47: [21, 20] (Total: 41/150)

Results for u120_02:
Bins used: 47
Theoretical minimum: 46
Deviation from optimal: 1 bins
Runtime: 27.0077s
Bin details:
Bin 1: [51, 75, 24] (Total: 150/150)
Bin 2: [37, 51, 34, 26] (Total: 148/150)
Bin 3: [64, 46, 38] (Total: 148/150)
Bin 4: [88, 38, 29] (Total: 146/150)
Bin 5: [73, 76] (Total: 149/150)
Bin 6: [91, 32, 25] (Total: 148/150)
Bin 7: [36, 67, 41] (Total: 144/150)
Bin 8: [92, 45] (Total: 137/150)
Bin 9: [64, 75] (Total: 139/150)
Bin 10: [67, 68] (Total: 135/150)
Bin 11: [97, 41] (Total: 138/150)
Bin 12: [62, 37, 48] (Total: 147/150)
Bin 13: [29, 85, 35] (Total: 149/150)
Bin 14: [99, 37, 23] (Total: 150/150)
Bin 15: [84, 65] (Total: 149/150)
Bin 16: [70, 53, 27] (Total: 150/150)
Bin 17: [57, 42, 44] (Total: 143/150)
Bin 18: [61, 38, 32] (Total: 131/150)
Bin 19: [69, 58, 29] (Total: 148/150)
Bin 20: [98, 29, 23] (Total: 150/150)
Bin 21: [100, 48] (Total: 148/150)
Bin 22: [81, 66] (Total: 147/150)
Bin 23: [79, 26, 32] (Total: 137/150)
Bin 24: [92, 50] (Total: 142/150)
Bin 25: [66, 83] (Total: 149/150)
Bin 26: [45, 97] (Total: 142/150)
Bin 27: [100, 29, 21] (Total: 150/150)
Bin 28: [67, 79] (Total: 146/150)
Bin 29: [94, 31, 21] (Total: 146/150)
Bin 30: [80, 60] (Total: 140/150)
Bin 31: [42, 31, 64] (Total: 137/150)
Bin 32: [81, 68] (Total: 149/150)
Bin 33: [96, 48] (Total: 144/150)
Bin 34: [80, 69] (Total: 149/150)
Bin 35: [80, 59] (Total: 139/150)
Bin 36: [35, 96, 24] (Total: 149/150)
Bin 37: [91, 33, 22] (Total: 146/150)
Bin 38: [90, 36] (Total: 126/150)
Bin 39: [66, 53, 28] (Total: 147/150)
Bin 40: [46, 76, 28] (Total: 150/150)
Bin 41: [74, 34, 30] (Total: 138/150)
Bin 42: [84, 64] (Total: 148/150)
Bin 43: [86, 61] (Total: 141/150)
Bin 44: [46, 62, 37] (Total: 145/150)
Bin 45: [84, 64] (Total: 148/150)
Bin 46: [79, 67] (Total: 146/150)
Bin 47: [59, 48, 47] (Total: 146/150)

algorthim FF lbeua u120_03:

Results for u120_03:
Bins used: 50
Theoretical minimum: 49
Deviation from optimal: 1 bins
Runtime: 0.0005 seconds
Bin details:
Bin 1: [100, 49] (Total: 149/150)
Bin 2: [100, 48] (Total: 148/150)
Bin 3: [99, 47] (Total: 146/150)
Bin 4: [97, 53] (Total: 150/150)
Bin 5: [97, 52] (Total: 149/150)
Bin 6: [97, 47] (Total: 144/150)
Bin 7: [96, 54] (Total: 150/150)
Bin 8: [96, 54] (Total: 150/150)
Bin 9: [95, 54] (Total: 149/150)
Bin 10: [95, 46] (Total: 141/150)
Bin 11: [95, 46] (Total: 141/150)
Bin 12: [95, 46] (Total: 141/150)
Bin 13: [94, 56] (Total: 150/150)
Bin 14: [92, 58] (Total: 150/150)
Bin 15: [92, 58] (Total: 150/150)
Bin 16: [91, 59] (Total: 150/150)
Bin 17: [91, 57] (Total: 148/150)
Bin 18: [90, 60] (Total: 150/150)
Bin 19: [90, 60] (Total: 150/150)
Bin 20: [90, 56] (Total: 146/150)
Bin 21: [89, 61] (Total: 150/150)
Bin 22: [88, 62] (Total: 150/150)
Bin 23: [87, 63] (Total: 150/150)
Bin 24: [87, 63] (Total: 150/150)
Bin 25: [86, 63] (Total: 149/150)
Bin 26: [86, 45] (Total: 131/150)
Bin 27: [85, 65] (Total: 150/150)
Bin 28: [84, 66] (Total: 150/150)
Bin 29: [84, 45, 21] (Total: 150/150)
Bin 30: [84, 45, 20] (Total: 149/150)
Bin 31: [83, 67] (Total: 150/150)
Bin 32: [82, 68] (Total: 150/150)
Bin 33: [82, 44, 24] (Total: 150/150)
Bin 34: [81, 43, 26] (Total: 150/150)
Bin 35: [80, 70] (Total: 150/150)
Bin 36: [80, 70] (Total: 150/150)
Bin 37: [80, 43, 27] (Total: 150/150)
Bin 38: [79, 71] (Total: 150/150)
Bin 39: [78, 71] (Total: 149/150)
Bin 40: [76, 74] (Total: 150/150)
Bin 41: [75, 74] (Total: 149/150)
Bin 42: [73, 73] (Total: 146/150)
Bin 43: [73, 42, 35] (Total: 150/150)
Bin 44: [42, 42, 40, 26] (Total: 150/150)
Bin 45: [40, 40, 39, 31] (Total: 150/150)
Bin 46: [37, 37, 35, 35] (Total: 144/150)
Bin 47: [35, 34, 34, 33] (Total: 136/150)
Bin 48: [32, 32, 30, 29, 27] (Total: 150/150)
Bin 49: [29, 28, 26, 25, 25] (Total: 133/150)
Bin 50: [25, 22] (Total: 47/150)

Results for u120_03:
Bins used: 50
Theoretical minimum: 49
Deviation from optimal: 1 bins
Runtime: 171.4093s
Bin details:
Bin 1: [79, 21, 26, 20] (Total: 146/150)
Bin 2: [91, 58] (Total: 149/150)
Bin 3: [96, 27, 32] (Total: 149/150)
Bin 4: [95, 54] (Total: 149/150)
Bin 5: [71, 40, 31] (Total: 142/150)
Bin 6: [97, 45] (Total: 142/150)
Bin 7: [94, 54] (Total: 148/150)
Bin 8: [95, 32, 22] (Total: 149/150)
Bin 9: [80, 68] (Total: 148/150)
Bin 10: [90, 35, 25] (Total: 150/150)
Bin 11: [84, 65] (Total: 149/150)
Bin 12: [96, 42] (Total: 138/150)
Bin 13: [85, 34, 28] (Total: 147/150)
Bin 14: [86, 37, 26] (Total: 149/150)
Bin 15: [87, 37, 24] (Total: 148/150)
Bin 16: [74, 66] (Total: 149/150)
Bin 17: [83, 29, 33] (Total: 145/150)
Bin 18: [95, 44] (Total: 139/150)
Bin 19: [88, 58] (Total: 146/150)
Bin 20: [92, 56] (Total: 146/150)
Bin 21: [99, 43] (Total: 142/150)
Bin 22: [96, 45] (Total: 141/150)
Bin 23: [97, 39] (Total: 136/150)
Bin 24: [82, 59] (Total: 141/150)
Bin 25: [100, 46] (Total: 146/150)
Bin 26: [97, 52] (Total: 149/150)
Bin 27: [91, 46] (Total: 137/150)
Bin 28: [75, 74] (Total: 149/150)
Bin 29: [95, 48] (Total: 143/150)
Bin 30: [60, 87] (Total: 147/150)
Bin 31: [71, 42, 27] (Total: 140/150)
Bin 32: [78, 47, 25] (Total: 150/150)
Bin 33: [89, 57] (Total: 146/150)
Bin 34: [73, 76] (Total: 149/150)
Bin 35: [82, 67] (Total: 149/150)
Bin 36: [86, 29, 35] (Total: 150/150)
Bin 37: [100, 42] (Total: 142/150)
Bin 38: [63, 81] (Total: 144/150)
Bin 39: [49, 63, 35] (Total: 147/150)
Bin 40: [90, 49, 25] (Total: 145/150)
Bin 41: [47, 35, 60] (Total: 142/150)
Bin 42: [92, 54] (Total: 146/150)
Bin 43: [84, 40, 26] (Total: 150/150)
Bin 44: [70, 45, 34] (Total: 149/150)
Bin 45: [90, 53] (Total: 143/150)
Bin 46: [43, 61, 46] (Total: 150/150)
Bin 47: [73, 73] (Total: 146/150)
Bin 48: [62, 84] (Total: 146/150)
Bin 49: [80, 70] (Total: 150/150)
Bin 50: [56, 30, 63] (Total: 149/150)

mcfa pturon bmrhck 1 maoptimali,

lkch lo 0.0005 snciot.

algorthim geneti lbeua u120_03:

mcfa pturon bmrhck 1 maoptimali,

lkch lo 171.4093 snciot.

אלגוריתם FF לבעה U120_04

מצא פתרון במרחב 0 מאופטימלי,

לקח לו 0.0002 שניות.

```
Results for u120_04:  
Bins used: 50  
Theoretical minimum: 50  
Deviation from optimal: 0 bins  
Runtime: 0.0002 seconds  
  
Bin details:  
Bin 1: [99, 50] (Total: 149/150)  
Bin 2: [99, 50] (Total: 149/150)  
Bin 3: [99, 52] (Total: 150/150)  
Bin 4: [98, 52] (Total: 150/150)  
Bin 5: [97, 53] (Total: 150/150)  
Bin 6: [97, 49] (Total: 146/150)  
Bin 7: [96, 48] (Total: 144/150)  
Bin 8: [95, 55] (Total: 150/150)  
Bin 9: [92, 57] (Total: 149/150)  
Bin 10: [92, 57] (Total: 149/150)  
Bin 11: [92, 57] (Total: 149/150)  
Bin 12: [92, 57] (Total: 149/150)  
Bin 13: [91, 59] (Total: 150/150)  
Bin 14: [91, 57] (Total: 148/150)  
Bin 15: [91, 56] (Total: 147/150)  
Bin 16: [90, 60] (Total: 150/150)  
Bin 17: [89, 61] (Total: 150/150)  
Bin 18: [89, 60] (Total: 149/150)  
Bin 19: [88, 45] (Total: 133/150)  
Bin 20: [87, 45] (Total: 132/150)  
Bin 21: [87, 43, 20] (Total: 150/150)  
Bin 22: [87, 43, 20] (Total: 150/150)  
Bin 23: [86, 42, 21] (Total: 149/150)  
Bin 24: [85, 65] (Total: 150/150)  
Bin 25: [84, 66] (Total: 150/150)  
Bin 26: [84, 65] (Total: 149/150)  
Bin 27: [84, 42, 24] (Total: 150/150)  
Bin 28: [84, 42, 24] (Total: 150/150)  
Bin 29: [82, 68] (Total: 150/150)  
Bin 30: [82, 68] (Total: 150/150)  
Bin 31: [81, 69] (Total: 150/150)  
Bin 32: [79, 71] (Total: 150/150)  
Bin 33: [78, 72] (Total: 150/150)  
Bin 34: [78, 71] (Total: 149/150)  
Bin 35: [77, 73] (Total: 150/150)  
Bin 36: [77, 73] (Total: 150/150)  
Bin 37: [76, 74] (Total: 150/150)  
Bin 38: [76, 73] (Total: 149/150)  
Bin 39: [75, 75] (Total: 150/150)  
Bin 40: [75, 73] (Total: 148/150)  
Bin 41: [71, 71] (Total: 142/150)  
Bin 42: [70, 69] (Total: 139/150)  
Bin 43: [69, 69] (Total: 138/150)  
Bin 44: [69, 67] (Total: 136/150)  
Bin 45: [42, 42, 41, 25] (Total: 150/150)  
Bin 46: [40, 40, 39, 31] (Total: 150/150)  
Bin 47: [39, 37, 37, 37] (Total: 150/150)  
Bin 48: [36, 35, 34, 32] (Total: 137/150)  
Bin 49: [32, 31, 30, 28, 27] (Total: 148/150)  
Bin 50: [23, 21, 21, 21, 20] (Total: 127/150)
```

```
Results for u120_04:  
Bins used: 51  
Theoretical minimum: 50  
Deviation from optimal: 1 bins  
Runtime: 17.9432s  
Bin details:  
Bin 1: [69, 81] (Total: 150/150)  
Bin 2: [98, 48] (Total: 146/150)  
Bin 3: [37, 61, 21, 20] (Total: 139/150)  
Bin 4: [57, 89] (Total: 146/150)  
Bin 5: [69, 37, 43] (Total: 149/150)  
Bin 6: [92, 42] (Total: 134/150)  
Bin 7: [87, 31, 28] (Total: 146/150)  
Bin 8: [97, 39] (Total: 136/150)  
Bin 9: [98, 55] (Total: 145/150)  
Bin 10: [77, 71] (Total: 148/150)  
Bin 11: [87, 39, 28] (Total: 146/150)  
Bin 12: [92, 52] (Total: 144/150)  
Bin 13: [99, 32] (Total: 131/150)  
Bin 14: [91, 57] (Total: 148/150)  
Bin 15: [91, 56] (Total: 147/150)  
Bin 16: [96, 40] (Total: 136/150)  
Bin 17: [69, 71] (Total: 140/150)  
Bin 18: [75, 71] (Total: 148/150)  
Bin 19: [98, 45] (Total: 143/150)  
Bin 20: [78, 69] (Total: 147/150)  
Bin 21: [92, 21, 24] (Total: 137/150)  
Bin 22: [84, 57] (Total: 141/150)  
Bin 23: [91, 36, 21] (Total: 148/150)  
Bin 24: [92, 38, 25] (Total: 147/150)  
Bin 25: [89, 23, 37] (Total: 149/150)  
Bin 26: [75, 71] (Total: 146/150)  
Bin 27: [76, 70] (Total: 146/150)  
Bin 28: [88, 53] (Total: 141/150)  
Bin 29: [84, 42, 21] (Total: 147/150)  
Bin 30: [82, 58] (Total: 132/150)  
Bin 31: [69, 78] (Total: 147/150)  
Bin 32: [42, 65, 41] (Total: 148/150)  
Bin 33: [75, 73] (Total: 148/150)  
Bin 34: [85, 50] (Total: 135/150)  
Bin 35: [71, 74] (Total: 145/150)  
Bin 36: [68, 32, 42] (Total: 142/150)  
Bin 37: [95, 43] (Total: 138/150)  
Bin 38: [57, 73, 29] (Total: 150/150)  
Bin 39: [73, 75] (Total: 148/150)  
Bin 40: [59, 84] (Total: 143/150)  
Bin 41: [68, 79] (Total: 139/150)  
Bin 42: [72, 76] (Total: 148/150)  
Bin 43: [86, 60] (Total: 146/150)  
Bin 44: [73, 77] (Total: 150/150)  
Bin 45: [35, 82, 31] (Total: 148/150)  
Bin 46: [84, 65] (Total: 149/150)  
Bin 47: [87, 57] (Total: 144/150)  
Bin 48: [67, 27, 24, 21] (Total: 139/150)  
Bin 49: [68, 34, 45] (Total: 147/150)  
Bin 50: [66, 40, 42] (Total: 148/150)  
Bin 51: [52, 97] (Total: 149/150)
```

אלגוריתם גנוי לבעה U120_04

מצא פתרון במרחב 1 מאופטימלי,

לקח לו 17.9432 שניות.

- במשימה זו נתבקשנו לתמוך בפתרון של מספר חידות Arc Puzzle של קאגל. על מנת לפתור את בעיות אלו עשינו את השלבים הבאים:
1. בחירת מצב - הוסףנו טפריט לבחירה בין מצב מחרוזת ל-Arc Puzzle.
 2. מדידת התאמה - פונקציית הפיטנס סופרת תאים תואמים בין המטריצה הנוכחית למטרה.
 3. אופרטורים גנטיים: מוציעה: שני ערקי אקראי, crossover: חלוקה אופקית/אנכית של המטריצה, ויזואליזציה - הצגה גרפית של 3 מטריצות במקביל: קלט, מטרה, פתרון.
 4. טעינתקובץ JSON - אפשרות לבחור תאיים ספציפי מקובץ המכיל מספר דוגמאות
 5. שמירת פונקציונליות - כל התוכנות המקוריות (בחירה הורים, מדדי גיוון, גרפים) נשמרו וועבדות בשני המצבים, האלגוריתם משתמש בהםים עוקרונות גנטיים גם למטריצות וגם למטריצות, עם התאמות ספציפיות לסוג הבעיה.

```
Select mode:
1 - String evolution
2 - ARC puzzle
Enter your choice (1/2): 2
Enter path to ARC JSON file: ARC_Puzzles/913fb3ed.json

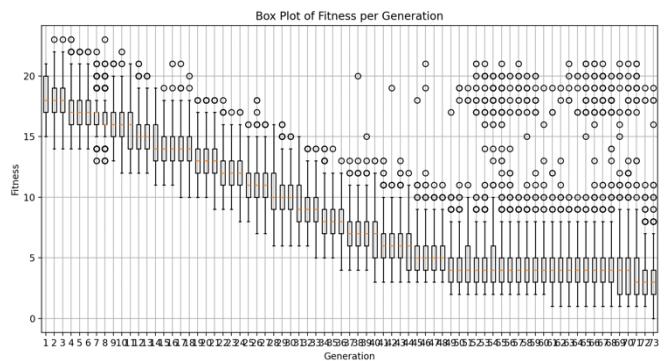
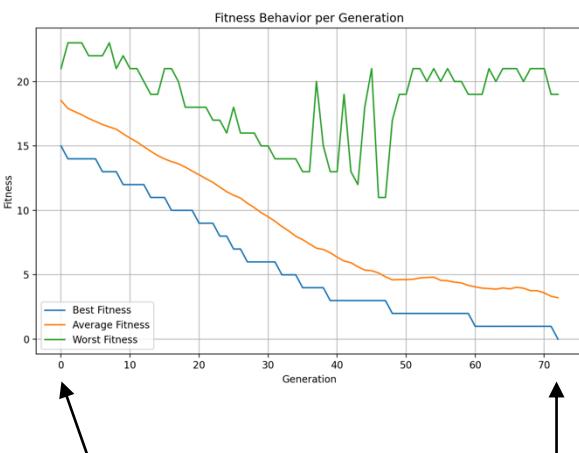
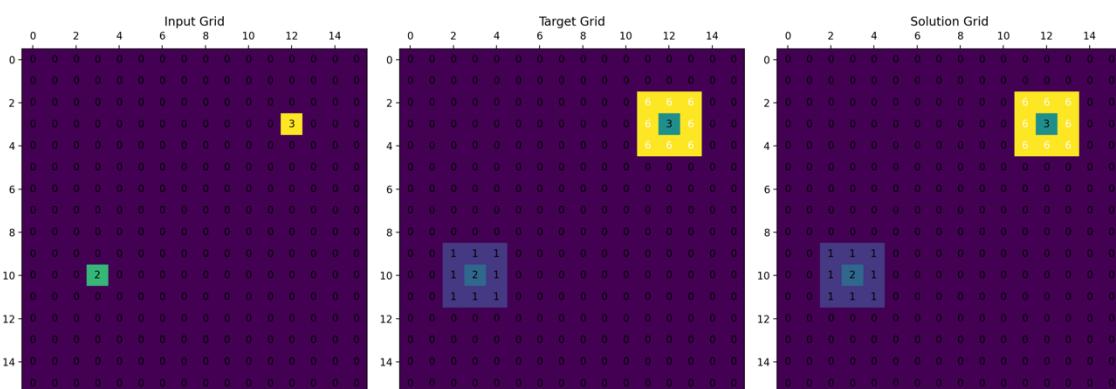
Found 4 training examples:
1. Input size: 12x12
2. Input size: 6x6
3. Input size: 16x16
4. Input size: 6x6

Select example (1-4): 3
```

זוגמא לבחירת הזנת פאל בטעינת הראיי ולאחר מכן בחירת הפאל בבחירה מתווך

קובץ ה-Json.

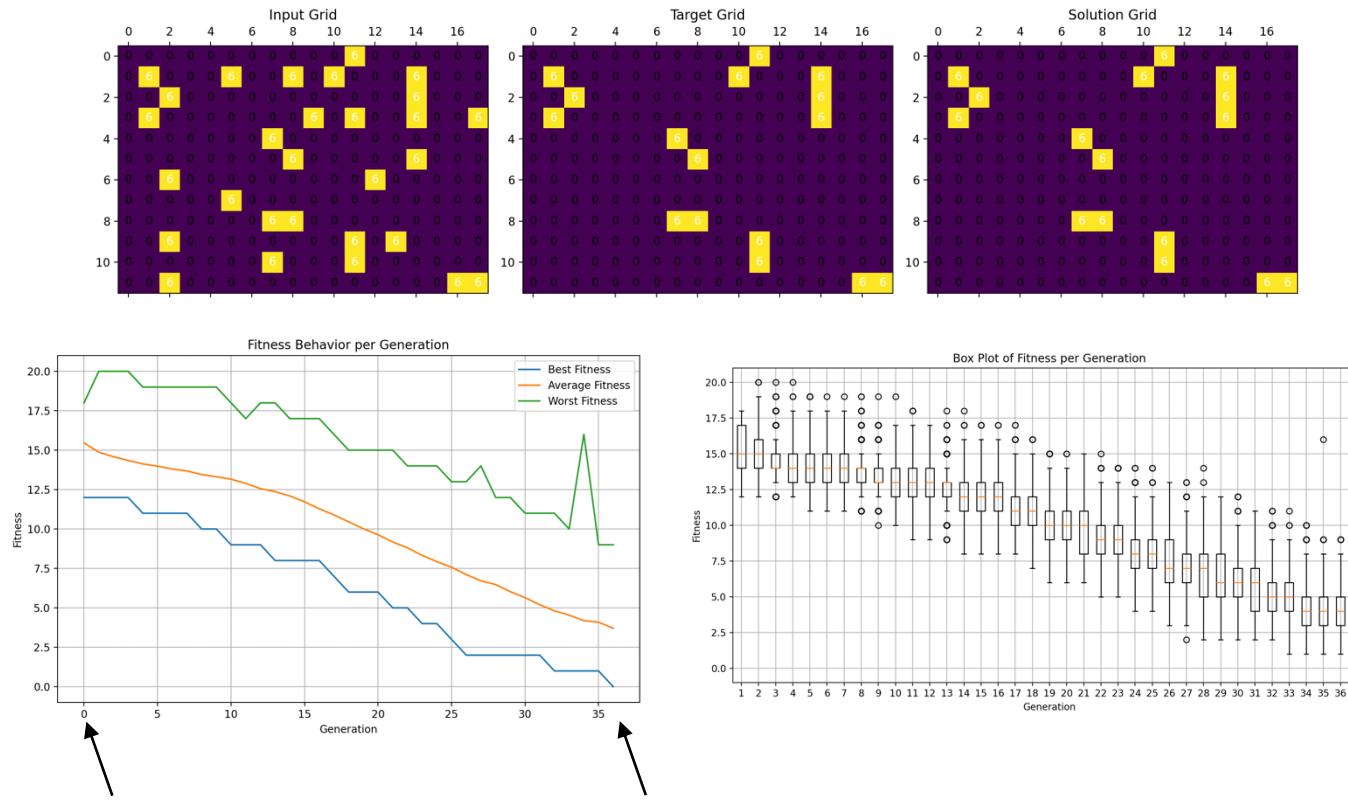
דוגמא 1 – פאל 913fb3ed



```
Gen 0: Best = Grid [0, 0, 0]... (Fitness = 15)
Avg Fitness = 18.53
Std Dev = 1.35
Worst Fitness = 21
Fitness Range = 6
Tick Duration (sec) = 0.0375
Total Elapsed Time (sec) = 0.0702
Selection Variance = 0.544346
Top-Average Selection Probability Ratio = 1.79
Avg Pairwise Hamming Distance = 0.00
Avg Number of Distinct Alleles per Gene = 9.15
Avg Shannon Entropy per Gene (bits) = 0.11
```

```
Gen 72: Best = Grid [0, 0, 0]... (Fitness = 0)
Avg Fitness = 3.22
Std Dev = 1.79
Worst Fitness = 19
Fitness Range = 19
Tick Duration (sec) = 0.0241
Total Elapsed Time (sec) = 44.0479
Selection Variance = 0.113730
Top-Average Selection Probability Ratio = 1.14
Avg Pairwise Hamming Distance = 0.00
Avg Number of Distinct Alleles per Gene = 5.25
Avg Shannon Entropy per Gene (bits) = 0.07
```

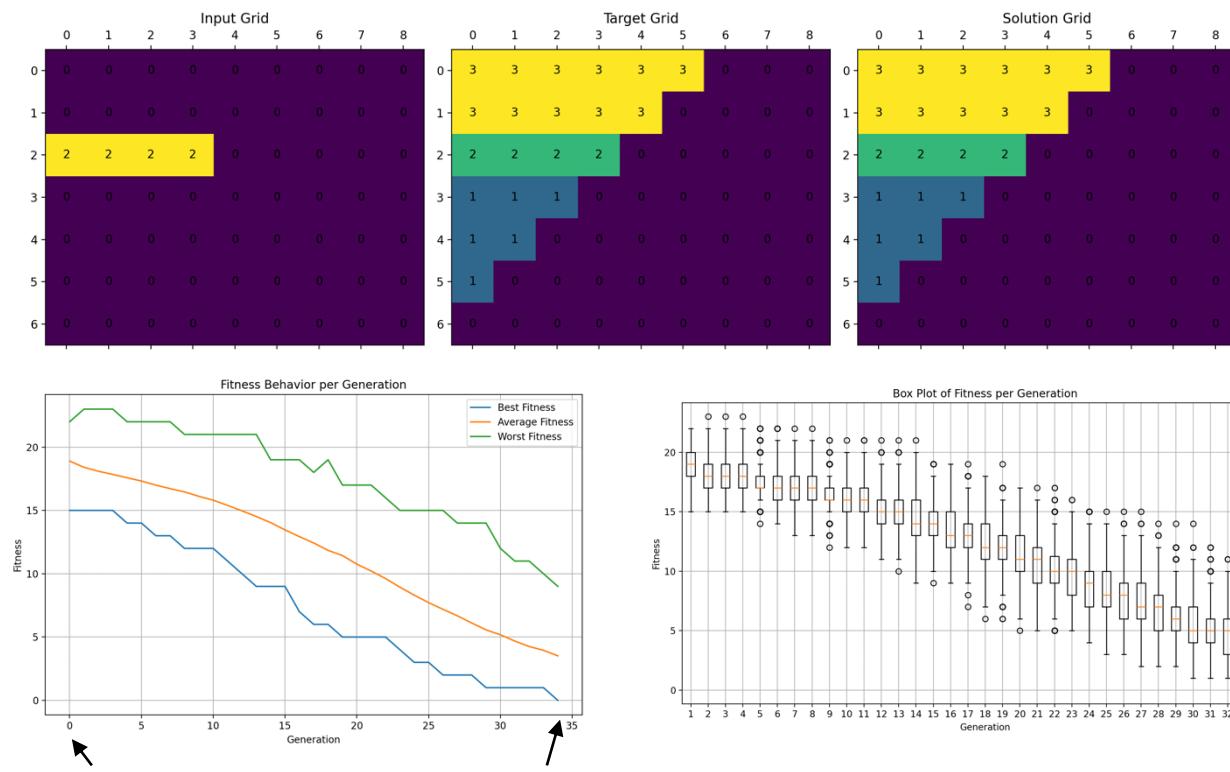
תגמא 2 – פאל



```
Gen 0: Best = Grid [0, 0, 0]... (Fitness = 12)
Avg Fitness = 15.46
Std Dev = 1.37
Worst Fitness = 18
Fitness Range = 6
Tick Duration (sec) = 0.0341
Total Elapsed Time (sec) = 0.0652
Selection Variance = 0.539645
Top-Average Selection Probability Ratio = 1.78
Avg Pairwise Hamming Distance = 0.00
Avg Number of Distinct Alleles per Gene = 9.46
Avg Shannon Entropy per Gene (bits) = 0.13
```

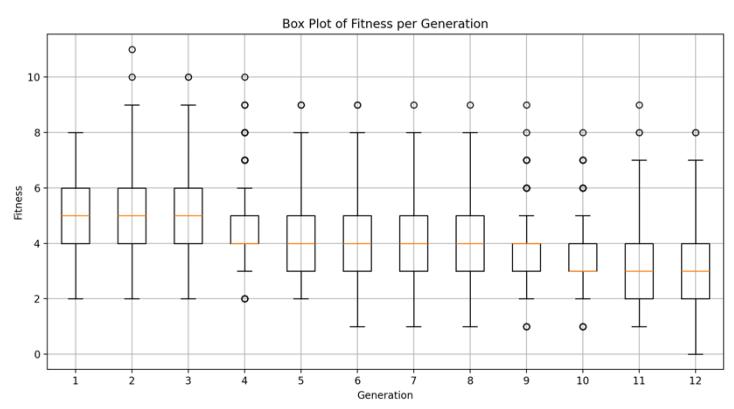
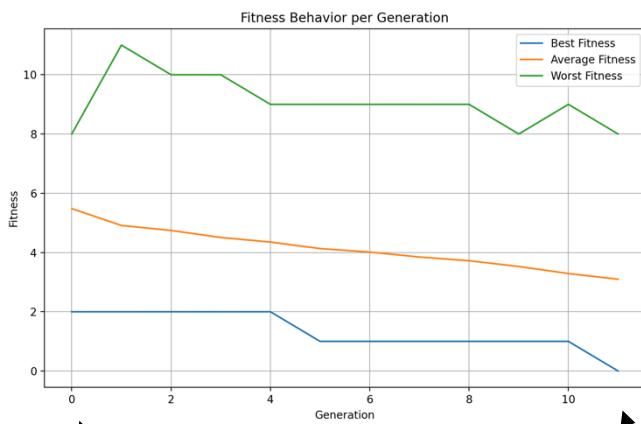
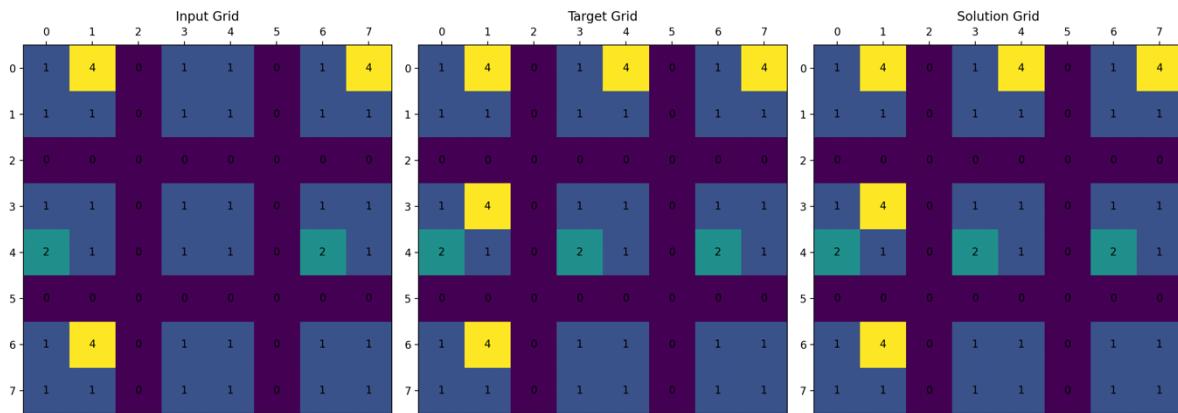
```
Gen 36: Best = Grid [0, 0, 0]... (Fitness = 0)
Avg Fitness = 3.70
Std Dev = 1.49
Worst Fitness = 9
Fitness Range = 9
Tick Duration (sec) = 0.0196
Total Elapsed Time (sec) = 20.7820
Selection Variance = 0.280486
Top-Average Selection Probability Ratio = 1.38
Avg Pairwise Hamming Distance = 0.00
Avg Number of Distinct Alleles per Gene = 5.16
Avg Shannon Entropy per Gene (bits) = 0.08
```

תגמא 3 – פאל



```
Gen 0: Best = Grid [0, 0, 0]... (Fitness = 15)
Avg Fitness = 18.90
Std Dev = 1.29
Worst Fitness = 22
Fitness Range = 7
Tick Duration (sec) = 0.0150
Total Elapsed Time (sec) = 0.0399
Selection Variance = 0.414414
Top-Average Selection Probability Ratio = 1.66
Avg Pairwise Hamming Distance = 0.00
Avg Number of Distinct Alleles per Gene = 10.00
Avg Shannon Entropy per Gene (bits) = 0.39
```

```
Gen 34: Best = Grid [3, 3, 3]... (Fitness = 0)
Avg Fitness = 3.51
Std Dev = 1.58
Worst Fitness = 9
Fitness Range = 9
Tick Duration (sec) = 0.0070
Total Elapsed Time (sec) = 17.9846
Selection Variance = 0.287247
Top-Average Selection Probability Ratio = 1.43
Avg Pairwise Hamming Distance = 0.00
Avg Number of Distinct Alleles per Gene = 9.02
Avg Shannon Entropy per Gene (bits) = 0.27
```



```
Gen 0: Best = Grid [1, 4, 0]... (Fitness = 2)
Avg Fitness = 5.48
Std Dev = 1.33
Worst Fitness = 8
Fitness Range = 6
Tick Duration (sec) = 0.0144
Total Elapsed Time (sec) = 0.0397
Selection Variance = 0.529603
Top-Average Selection Probability Ratio = 1.75
Avg Pairwise Hamming Distance = 0.00
Avg Number of Distinct Alleles per Gene = 9.98
Avg Shannon Entropy per Gene (bits) = 0.37
```

```
Gen 11: Best = Grid [1, 4, 0]... (Fitness = 0)
Avg Fitness = 3.10
Std Dev = 1.17
Worst Fitness = 8
Fitness Range = 8
Tick Duration (sec) = 0.0072
Total Elapsed Time (sec) = 5.8972
Selection Variance = 0.238164
Top-Average Selection Probability Ratio = 1.32
Avg Pairwise Hamming Distance = 0.00
Avg Number of Distinct Alleles per Gene = 8.44
Avg Shannon Entropy per Gene (bits) = 0.19
```