ביולוגיה חישובית - דו"ח תרגיל בית 2

אריאל אשכנזי 208465096 אור בבלי 208435917

על מנת להריץ את התוכנית יש למשוך את הקוד מ-GitHub ולהריץ את GeneticAlgorithm.py:
Git clone https://github.com/ArielAsh1/Genetic-Algorithm.git
לאחר מכן, להיכנס לתיקייה שמכילה את הrepository שלנו, ולהריץ: python3 GeneticAlgorithm.py.
בפועל, התוכנית מריצה 2 קבצים- heuristics.py, GeneticAlgorithm.py. שאר הקבצים בגיט הם הקבצים הנתונים בתרגיל ובנוסף קיים הקובץ make_plots.py אשר שימש ליצירת הגרפים ואין צורך להריץ אותו.
התוכנית משתמשת בחבילות סטנדרטיות ולכן אין צורך בהתקנות נוספות.

התוכנית רצה עם פרמטרים דיפולטיביים (משתנים גלובליים) שמוגדרים מראש בתחילת התוכנית:

- ELITE_PERCENT- מהווה את אחוז הפרמוטציות ה"טובות" ביותר שאותן נשמור כמו שהן לדור הבא, מאותחל עם 0.1.
- MUTATION_PERCENT- מהווה את אחוז הפרמוטציות שיעברו מוטציות (החלפת 2 אותיות בפרמוטציה), מאותחל עם 0.3.
- DROPOUT_PERCENT- מהווה את אחוז הפרמוטציות ה"גרועות" ביותר, אותן נמחק בכל דור מסך הפרמוטציות ונחליף אותן בפרמוטציות חדשות שנוצרו ב- crossover. מאותחל עם 0.4. נציין שלאוכלוסיית ה-crossover אין פרמטר דיפולטיבי מוגדר מראש כמו לאחרות, אלא כמות האוכלוסייה שתעבור crossover היא האוכלוסייה המלאה פחות אחוזי ה-ELITE וה-DROPOUT, כלומר במקרה שלנו יעברו $1 - 0.2 - 0.4 = 0.4$. פירוט מלא מטה.
- POPULATION_SIZE- גודל האוכלוסייה, כלומר כמות הפרמוטציות השונות בכל דור. מאותחל עם 300.
- STUCK_THRESHOLD- מדד סף שמסמן כמה דורות רצופים נוכל להישאר "תקועים" עם אותו הציון. במידה ונעבור את הערך הזה, התוכנית תסתיים. מאותחל עם 15.
- ROUNDS- מספר הדורות שנריץ את התוכנית, מאותחל עם 150.
- בנוסף, ישנן משקולות שונות שמשפיעות על משקל הציון שמוסיפים הסגמנטים השונים הקוד.
  המשקולות הן: LETTER_WEIGHT, PAIR_WEIGHT, WORDS_WEIGHT, COMMON_WEIGHT, IMPORTANT_WEIGHT.
  למשל, PAIR_WEIGHT גדול מ-LETTER_WEIGHT, שכן מציאת זוג אותיות בקובץ המפוענח שלנו בעלות הסתברות הופעה דומה להסתברותם במציאות, בעלת משמעות יותר חזקה ממציאת אות בודדת בקובץ המפוענח שהסתברותה ההופעה שלה דומה למציאות.

הסבר על האלגוריתם:

האלגוריתם מתבסס על הימצאותם של קבצי השכיחויות הנתונים- Letter2_Freq.txt, Letter_Freq.txt, dict.txt אשר משמשים להשוואת שכיחויות בין הקובץ המפוענח שלנו לשכיחויות הידועות בעולם. השוואת השכיחויות הזו היא למעשה הבסיס לפונקציית ה-fitness שלנו, כך שככל שהשכיחויות שלנו יהיו קרובות יותר לעולם האמיתי כך הציון בפיטנס גבוה יותר, מה שמעיד על פיענוח טוב יותר של הקובץ. בכל סוף דור נבחרת הפרמוטציה הטובה ביותר והיא זו שמשמשת לפיענוח הקובץ בדור הנוכחי.
הבדיקה בפונקציית ה-fitness מתבצעת ב-4 שלבים:

i. חישוב שכיחות הופעת האותיות בקובץ המפוענח ביחס לשכיחות הופעת האותיות בעולם האמיתי- ראשית אנו מחשבים את השכיחות של כל אותיות ה-ABC בקובץ המפוענח שלנו. אות שלא נמצאת תקבל שכיחות 0. לאחר מכן אנחנו משווים את השכיחות שמצאנו אצלנו עם השכיחות במציאות באופן הבא- רצים בלולאה עבור כל אות, ומחשבים את ההפרש בערך מוחלט בין השכיחות שלנו עבור האות הנוכחית לבין השכיחות הידועה עבור אותה האות. את כל ההפרשים הללו עבור כל אחת מהאותיות אנחנו סוכמים ומחזירים כמשתנה - letter_freq_diff.

ii. חישוב שכיחות הופעת צמדי האותיות בקובץ המפוענח ביחס לשכיחות הופעת הצמדים בעולם האמיתי- תהליך זה דומה לחישוב עבור האותיות הבודדות. ראשית נחשב את השכיחות בקובץ המפוענח שלנו עבור כל הזוגות האפשריים. לאחר מכן בדומה לתהליך שתיארנו מעלה, נרוץ בלולאה על כל צמדי האותיות ונחשב את ההפרש בערך מוחלט של השכיחויות עבור כל צמד. את ההפרשים האלו נסכום ונחזיר למשתנה pairs_freq_diff.

iii. חישוב הופעת המילים הנפוצות הנתונות בקובץ המפוענח שלנו- נרוץ על הקובץ המפוענח ונעבור על כל המילים שנמצאות בו. עבור כל מילה נבדוק האם שייכת לרשימת

המילים הידועות הנתונה. בנוסף נבדוק ספציפית האם המילה היא "חשובה" כלומר ב-{"a", "i"}, שכן למציאתן נרצה לתת חשיבות גדולה יותר. לאחר שסכמנו את מספר ההופעות של המילים הנפוצות והמילים ה"חשובות", נחזיר את הציון בעזרת החישוב הבא:

$$\frac{common\_words\_found * COMMON\_WEIGHT + important\_words\_found * IMPORTANT\_WEIGHT}{output\_word\_count}$$

כאשר $IMPORTANT\_WEIGHT\ (2.5) > COMMON\_WEIGHT(1.5)$ בכדי לתת להן יותר משקל, ואת זה נחלק בכמות המילים הכוללת בקובץ. החישוב הזה יוחזר לתוך המשתנה words_score בפונקציית הפיטנס.

iv. לאחר שבידנו שלושת התוצאות עבור 3 הבדיקות הנ"ל, את הציון הסופי שתחזיר פונקציית הפיטנס נחשב באופן הבא-

$$-(letter\_freq\_diff * LETTER\_WEIGHT) - (pairs\_freq\_diff * PAIR\_WEIGHT) + (words\_score * WORDS\_WEIGHT)$$

כלומר כל תוצאת בדיקה נכפיל במשקל המיועד לה בהתאם לגודל המשמעות שלה.
למשל מציאת הבדל שכיחויות קטן יותר עבור צמדי האותיות הוא משמעותי ונדיר יותר ממציאת הבדל שכיחויות קטן עבור האותיות הבודדות, ולכן הגדרנו $LETTER\_WEIGHT = 1, PAIR\_WEIGHT = 10$ בהתאם.
מהציון הכולל שנחזיר נחסר את ההבדלי שכיחויות (צמדים ובודדות) ונחבר אליו את ציון הופעת המילים הנפוצות. ההיגיון מאחורי כך הוא שככל שהבדל השכיחויות שנקבל יהיה קטן יותר, כך הנקודות שנחסר יהיו קטנות יותר ובכך פחות יורידו את הציון הכולל של הפיטנס ולכן עם פרמוטציות אלו נרצה להמשיך. את המדד של המילים הנפוצות נרצה לחבר לסכום שנחזיר, שכן ככל שמצאנו יותר מילים נפוצות בפיענוח שלנו, כך נרצה להעלות את הציון הסופי.

בסיום ריצת התוכנית, הפתרון ייוצג בעזרת הקבצים plain.txt שמכיל את הטקסט המפוענח שלנו, ואת הקובץ perm.txt שמכיל את הפרמוטציה אשר הביאה לפיענוח הזה. בנוסף, התוכנית מדפיסה את מספר הפעמים שנקראה פונקציית ה-fitness שהביאה אותנו לפתרון.

<u>סעיף א:</u>
ראשית האלגוריתם קורא את הקבצים הנתונים ושומר את המידע שלהם על מנת שנוכל להשתמש בו להשוואות לפיענוח שלנו בהמשך. לאחר מכן מיוצרות הפרמוטציות ההתחלתיות אשר מועברות לפונקציה הראשית run_round שמופעלת בלולאה כמספר הדורות שהגדרנו, ומריצה דור חדש ומעודכן בכל איטרציה. זהו למעשה החלק העיקרי באלגוריתם, שבסופו נגיע להתכנסות ופיענוח הצופן.
נפרט על שלבי הפעולות שמבוצעות ב-run_round:

1. עבורה כל אחת מהפרמוטציות שלנו (כגודל האוכלוסייה) נחשב את הפיטנס עבורה ונשמור את הציון ברשימה.
2. ניצור רשימה חדשה המכילה את האינדקסים של רשימת הציונים, כאשר הן מסודרים באופן יורד לפי הציון שהתקבל.
3. נחלץ מרשימת האינדקסים הממוינת את ה-ELITE שהם 10% הפרמוטציות הטובות ביותר בדור הנוכחי ונשמור אותן כמו שהן לשימוש בדור הבא. נחלץ את ה- DROPOUT שהם 40% הגרועות ביותר אותן נמחק מהרשימה ונשאר עם 40% פרמוטציות נוספות.
4. על הנותרות האלו נבצע crossover שמשלב הורה אחד מאוכלוסיית ה-ELITE והורה אחד מהנותרת, ליצירת 40% פרמוטציות חדשות על חשבון אלו שנמחקו ב-DROPOUT .
5. מתוך אוכלוסיית ה-crossover החדשה, על 30% מתוכה נבצע מוטציה (בהתאם לערך MUTATION_PERCENT). כל מוטציה תחליף לפרמוטציה 2 אותיות באותיות אחרות, ותוודא שאין חזרות במיפוי האותיות החדש ושכל האותיות האפשריות עדין מיוצגות.
כל השינויים האלו מטרתם לוודא שהפרמוטציות ממשיכות להשתנות באופן קבוע ובדרכים מגוונות בכדי למנוע התכנסות מוקדמת של האלגוריתם.
6. לסיום, נשלב את אוכלוסיית ה-ELITE ואוכלוסיית ה-crossover לקבלת האוכלוסייה החדשה והמעודכנת שלנו אותה נרצה להעביר לדור הבא.
7. על מנת שהאוכלוסייה החדשה אכן תמשיך לדור נוסף, נבדוק בסוף שהיא לא הגיעה להתכנסות בעזרת 3 התנאים הבאים:
   a. is_max_round: פונק' שמוודאת שלא הגענו לדור האחרון כפי שהוגדר במשתנה ROUNDS. במידה וכן היא תסיים את הריצה.
   b. intersection_percent_with_common_words: הפונקציה מחשבת אחוזי חיתוך בין המילים בקובץ המפוענח הנוכחי לבין רשימת המילים הנפוצות הידועות שקיבלנו בקובץ dict.txt. אם אחוזי החיתוך הינם 100, כלומר כל המילים בפיענוח שלנו נמצאות גם בקובץ מילים הנפוצות, משמע הגענו להתכנסות ונסיים את הריצה.

c. is_stuck : פונקציה שמטרתה לזהות אם נשארנו "תקועים" על אותו הציון הטוב ביותר מעל לסף דורות מסוים (STUCK_THRESHOLD). הפונקציה עוקבת מתי היה הדור האחרון בו התרחש שינוי בציון הפיטנס, ובמידה ולא השתנה הציון כבר 15 דורות (הגדרה שלנו) אז היא תכריז על התכנסות ותסיים את הריצה.

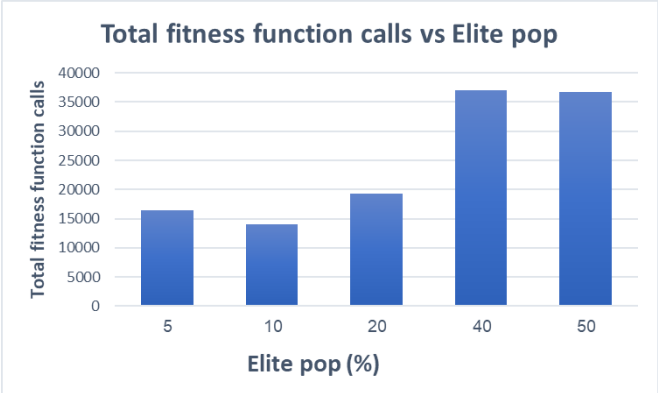
8. במידה ואף אחד משלושת התנאים הללו לא התקיים, נשלח את רשימת הפרמוטציות החדשה לדור הבא, שם תעבור שוב את אותו התהליך מחדש וחוזר חלילה.

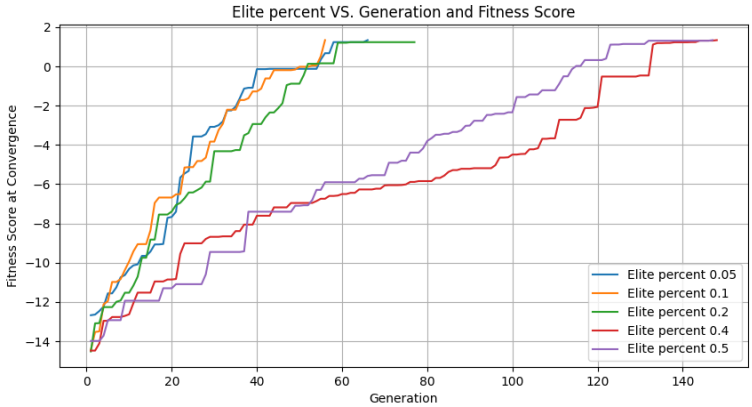
<u>מציאת הפרמטרים</u>

כעת, נסביר כיצד בחרנו את הפרמטרים האופטימליים לריצת התוכנית שלנו. לשם כך, נעזרנו בסקריפט שכתבנו, אליו אנו מכניסים את תוצאות ההרצה השונות, והוא יוצר לנו את הגרפים הבאים. את ה-bar chart יצרנו בעזרת המידע של total fitness calls. בסוג הגרף השני, ציר ה-X הוא מספר הדורות וציר הY מכיל את ערכי המקסימום של ה-fitness score בכל דור.

חשוב לציין, שבחלק א' של התרגיל, בכל איטרציה של האלגוריתם, קראנו גודל האוכלוסייה פעמים לפונ' הfitness שלנו. ועל כן, מספר הקריאות הכולל הוא פחות או יותר מספר האיטרציות כפול גודל האוכלוסייה.

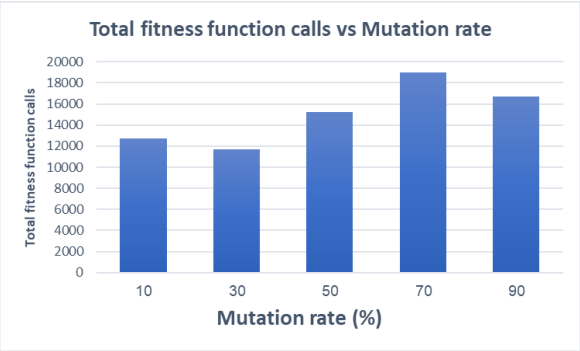
גרף 1- השוואה של אחוז Elite population:

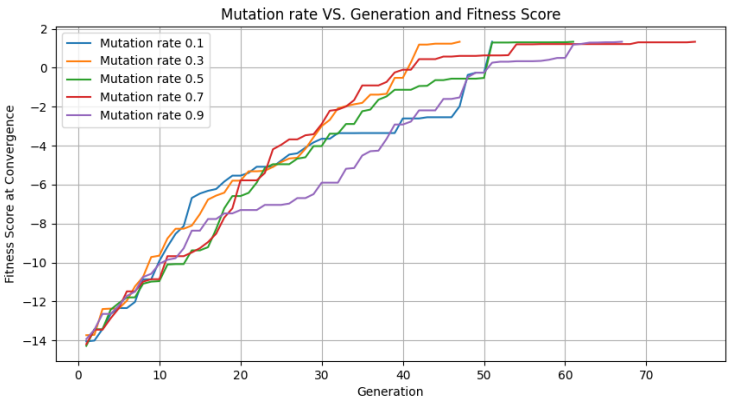




ראשית, רצינו לדעת מהי אחוז האוכלוסייה המתאים אותו נשמר בין ריצה לריצה, בלי להפעיל עליו קרוסאובר/מוטציות וכו'. אוכלוסייה זו היא האחוז העליון (המפורט בגרף). שיערנו כי עדיף לשמור על אחוז קטן מהאוכלוסייה, כדי לאפשר וורביליות ושינויים באוכלוסייה שיובילו להתכנסות מהירה יותר, ואכן ניתן לראות כי האלגוריתם מתכנס מהר יותר כאשר אחוזי הtop population נמוכים יותר. לכן, בחרנו ב10%. ניתן לראות שאחוז זה מוביל להתכנסות מהירה יותר וגם לפחות קריאות לפונ' ה-fitness.

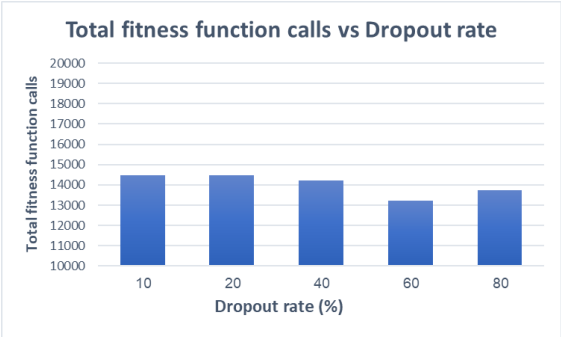
גרף 2- השוואה של אחוז מוטציות:

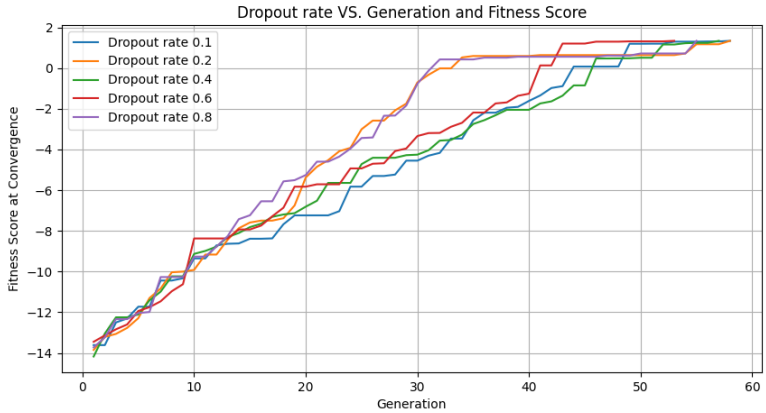




כעת, רצינו לבדוק מהו אחוז המוטציות הנכון. בקוד שלנו, לאחר שאנו מעיפים את האחוזון התחתון שהגדרנו ומבצעים קרוסאובר על האוכלוסייה, אנו מבצעים מוטציית החלפה אחת עבור פריטים רנדומליים מתוך האוכלוסייה החדשה (ללא הelite popularton). כאשר אנו מסתכלים על הגרפים, ניתן לראות כי 30% מוטציות מוביל להתכנסות המהירה ביותר, ועל כן גם למספר קריאות המערכת הנמוך יותר. אחוז זה מאפשר מספיק שונות באוכלוסייה כדי להוביל להתכנסות, ועם זאת לא גורם ליותר מדי שינויים שעלולים להשפיע, כפי שקורה באחוזי מוטציות גבוהים יותר ל100.

גרף 3- השוואה של אחוז הDropout:

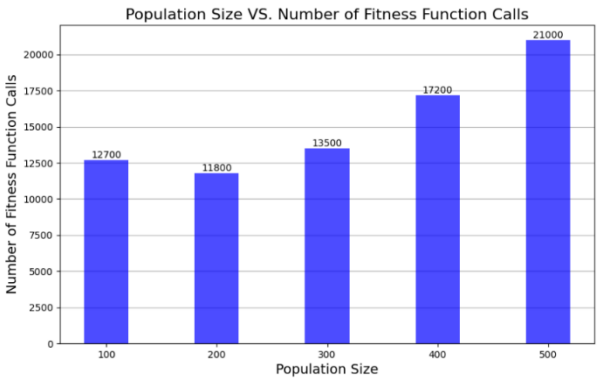


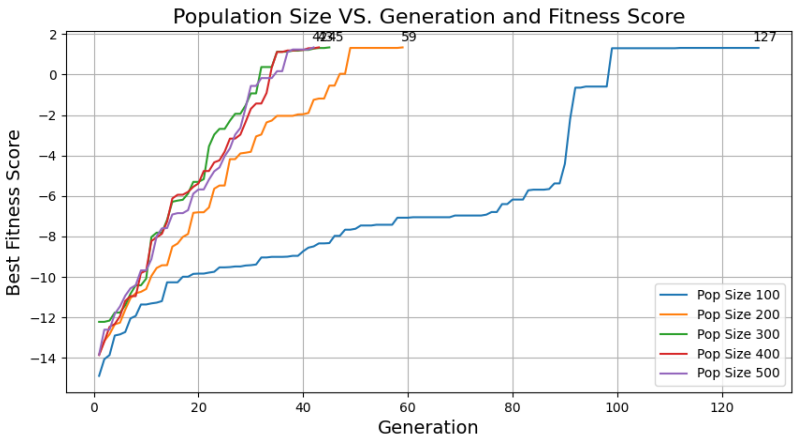


הנתון הבא אותו רצינו לבדוק הוא אחוז הDropout - כלומר, מאיזה גודל של אוכלוסייה, מהאחוזון התחתון של ערכי הfitness, אנו רוצים לזרוק?

כפי שרואים בגרף בצד שמאל, לאחוזים השונים אין השפעה קריטית על התכנסותה האלגוריתם - כל הריצות התכנסו בממוצע לאחר 55 דורות, ובעלות מספר דומה של קריאות לfitness. אם זאת, שמנו לב ש40% dropout מוצלח מעט יותר מהיתר כאשר ניסינו צפנים אחרים, ועל כן בחרנו אותו.

גרף 4- השוואה של גודל אוכלוסייה:





הפרמטר האחרון אותו רצינו לבדוק בחלק הוא גודל האוכלוסייה. אנו שיערנו שככל שהאוכלוסייה גדלה, כך נגיע להתכנסות מהירה יותר, כיוון שתהיה יותר ורביליות שתוביל לפרמוטציות שונות יותר, וכך להתכנסות מהירה יותר.
להפתעתנו, נוכחנו לדעת כי מעבר לסף מסוים, גודל האוכלוסייה לא משפיע מאד על ההתכנסות, והתכנסות האלגוריתם דומה כשגודל האוכלוסייה הוא 300-500. החלטנו לבחור בגודל אוכלוסייה של 300, כיוון שלגודל האוכלוסייה יש השפעה ישירה על הזמן שלוקח לריצת האלגוריתם, כמו גם מספר נמוך יותר של קריאות לפונ' fitness.

לאחר השוואת הפרמטרים הללו, החלטנו כי ערכי הפרמטרים האופטימליים, שהביאו יחדיו להתכנסות המהירה ביותר, הינם:

- Elite Population של 10%.
- אחוז מוטציות של 30%.
- Dropout rate של 40%.
- גודל אוכלוסייה – 300 פרמוטציות.

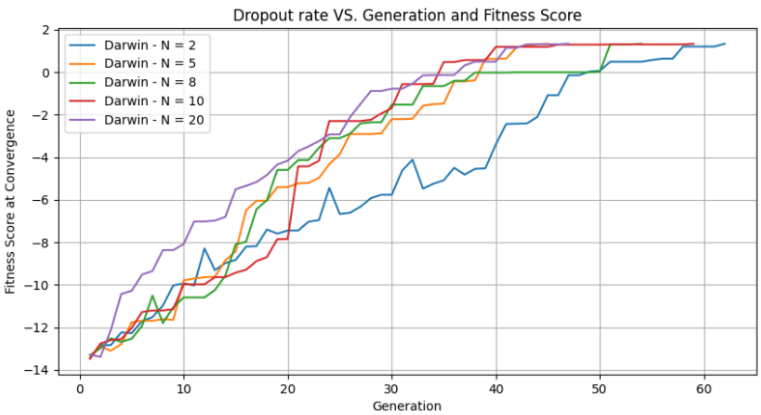
חלק ב':

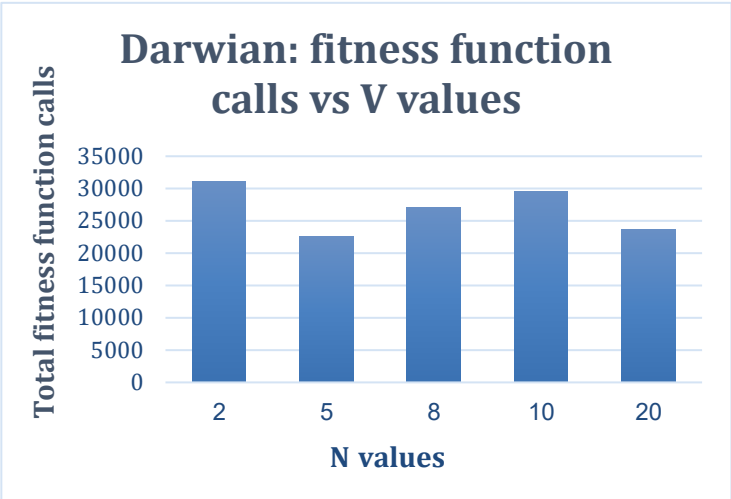
כעת, בחלק ב' של התרגיל, בנינו 2 פונקציות חדשות:

- run_round_darwin(permutations, curr_round, N, fitness_scores, typeFlag)
- check_local_optimum(permutation, old_fitness, N, typeFlag)

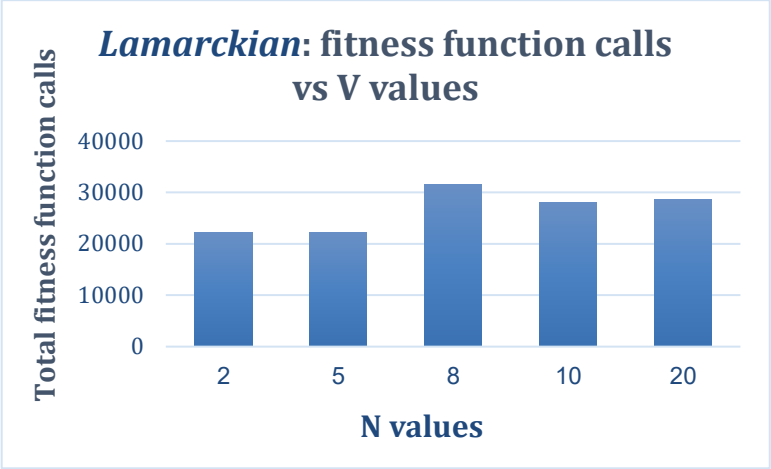
הפונקציה הראשונה דומה לפונקציית הריצה בחלק א', אך הפעם, אנו מחשבים fitness scores לאוכלוסייה בתוך check_local_optimum, ולא בתחילת כל איטרציה בrun_round.
כמו בסעיף א', אנחנו עדיין שומרים את הפרמוטציות הכי טובות, בהתאם לאחוזים שהגדרנו בתחילת הריצה, מורידים את האחוז התחתון, ומבצעים קרוסאובר ומוטציות על היתר.
ואז, עבור כל פרמוטציה באוכלוסייה שלנו, אנו שולחים אותה לcheck_local_optimum. שם, אנו מחשבים מהו fitness עבור כל פרמוטציה, ומעבירים אותה סדרה של N מוטציות, ומחשבים את הציון שלה שוב. במידה והציון החדש טוב מהקודם, היא שומרת את הציון החדש והפרמוטציה החדשה.
במידה ומדובר בשיטה הדארווינית, אנו מחזירים את הפרמוטציה לפני המוטציות, ואת הציון החדש שלה.
במידה ומדובר בשיטה הלאמרקנית, אנו מחזירים את הפרמוטציה והציון החדשים.
כעת, אנו יוצרים רשימה חדשה של פרמוטציות וציוני fitness, המתבססים על הפרמוטציה והציון שחזרנו מפונ' check_local_optimum.
שאר האלגוריתם שכתבנו דומה לסעיף א', עם תנאי התכנסות דומים. הפרמטרים זהים לפרמטרים בסעיף א'.

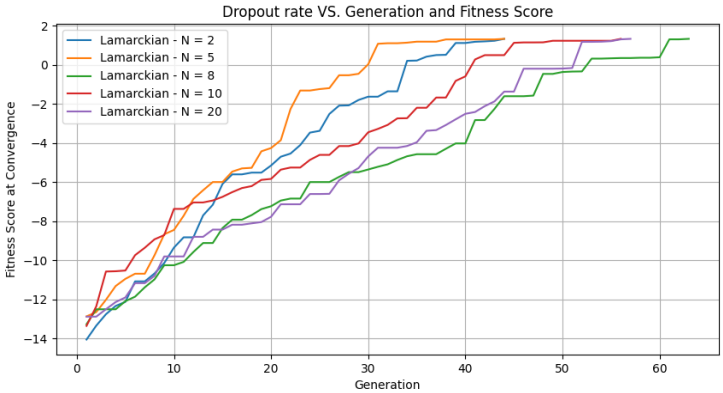
גרף 6- בחירת ערך N – דארווין:





גרף 7 – בחירת ערך N – למארק:

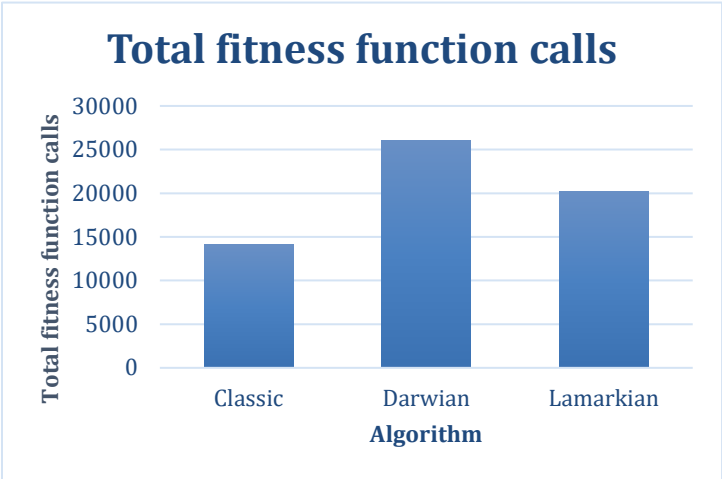


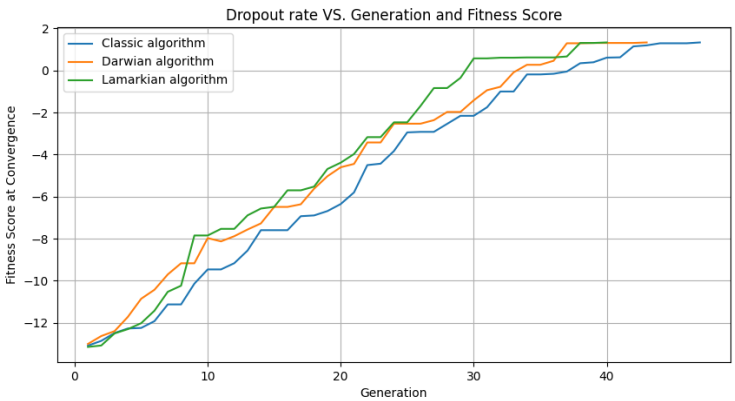


ניתן לראות שבשני האלגוריתמים, N=5 היה הערך שהוביל לתוצאות הטובות ביותר, גם מבחינת התכנסות וגם מבחינת כמות קריאות לפונ' fitness (כמו גם N=2 עבור לאמרק).
חשוב לציין, כי כאן מספר הקריאות כפול עבור כל פרמוטציה בכל דור, ועל כן אנו רואים פי 2 יותר קריאות לפונ' fitness מאשר באלגוריתם הקלאסי.

<u>השוואה בין האלגוריתמים השונים:</u>

לאחר שביצענו את כל ההשוואות והצגנו את המסקנות שלנו לגבי כל סוג אלגוריתם (קלאסי, דארוויני,, לאמרקי) נרצה להשוות את הביצועים שלהם עם הפרמטרים האופטימליים שמצאנו.
לשם כך, הרצנו כל אלגוריתם, וייצרנו גרפים מתוצאות ההרצה בצורה דומה לשאר הגרפים





ניתן לראות כי בסה"כ, ביצועי האלגוריתמים השונים דומים, כאשר האלגוריתם הקלאסי מתכנס מעט יותר לאט לעומת השניים האחרים. ההבדל המשמעותי בקריאות לפונ' fitness נובע מקריאה כפולה עבור כל פרמוטציה בכל דור באלגוריתמים של דארווין ולאמרק. ועל כן, כאשר נרצה לפענח צופן חדש, נרצה להריץ את האלגוריתם הקלאסי – לו יש פחות קריאות לפונ' הפיטנס, ביחד עם הלאמרקני, שנראה שנוטה להתכנס יותר מהר, בגלל המוטציות התדירות שעובר בכל סיבוב.