אלגוריתם גנטי לשבירת צופן מונו-אלפביתי

מגישים:
חיים הגר **ת.ז**
עומר ארד 314096389

האלגוריתם, שכתוב ב-python, מנסה למצוא את המיפוי הנכון בין התווים המוצפנים לתווי הטקסט המקוריים.

להלן **פירוט של הקוד:**

1. הקוד מגדיר את המחלקות Solution ואת תתי המחלקות 'DarwinSolution' ו'LamarckSolution'. מחלקות אלו מייצגות גישות שונות לאבולוציה של הפתרונות.
2. המחלקה 'Solution' מגדירה שיטות לחישוב פונקציית ההערכה, המרת מילים ע"פ המיפוי הנוכחי, והערכת התאמת מילים ותדירויות, וביצוע מוטציות.
3. המחלקה 'DarwinSolution' מנסה לשפר את ביצועי האלגוריתם באמצעות אבולוציה דרוויניסטית, בעוד שהמחלקה 'LamarckSolution' מנסה לעשות זאת באמצעות אבולוציה הלמאקרית.
4. בנוסף ישנם בקוד פונקציות עזר לחישוב תדירות של אות וזוג אותיות, המרת טקסט למילון (dictionary – מבנה הנתונים בו השתמשנו לייצוג הפתרונות), בחירת פתרונות לביצוע cross-over וכן ביצוע הפעולה, התאמת זוגות, וקביעת הפתרון שימשיך הלאה.
5. הפונקציה 'gen_population' אחראית לייצר אוכלוסייה ראשונית של פתרונות פוטנציאליים תוך שימוש בתמורות של האלפאבית.
6. הפונקציה 'genetic' מיישת את לולאת האלגוריתם הגנטי הראשי, אשר מפתחת את האוכלוסייה על פני מספר מוגדר של דורות. כאן נעשים חישובי פונקציית ההערכה, נבחרים פתרונות לרבייה, מתבצעת ההצלבה (cross-over) ומתבצע המעקב אחר הפתרון הטוב ביותר שנמצא.

הוראות הרצה:

במידה ולא מסופק קלט לתכנית הקוד ירוץ על מצב 0 (אבולוציה רגילה כלומר 'solution'), עם אוכלוסייה ראשונית של 20,000 פתרונות פוטנציאליים.
אחרת, ניתן לספק לקוד מספר בין 0-2 המייצג את סוג האבולוציה (0-Solution, 1-Darwin, 2-Lamarck) ואת גודל האוכלוסייה ההתחלתי (במידה ולא יסופק הוא יוגדר באופן דיפולטיבי להיות 5000).
כמו כן התכנית מניחה קיום של הקובץ enc.txt ואותו היא מנסה לפענח.

ייצוג הפתרונות:

הפתרונות מוצגים בתור מילון שבו המפתחות הן אותיות האלפאבית לפי הסדר והערך של כל מפתח הוא האות שאליו המפתח ממופה.

למשל:
a : y, b : x, c : i, d : n, e : t, f : o, g : x, h : x, i : c, j : e, k : b, l : l, m : d, n : u, o : k, p : m, q : s, r : }
{v, s : p, t : q, u : r, v : h, w : w, x : g, y : a, z : f

הוא המיפוי המתקבל לאחר הרצת האלגוריתם שלנו על הטקסט הנתון, ללא ארגומנטים נוספים. ניתן לראות שהאות a ממופה לאות y, האות b לאות x, וכן הלאה.

פונקציית ההערכה:

נתאר את הפונקציה במספר שלבים:

1. ציון התאמת מילים:
- ציון התאמה למילה מחושב על ידי השוואת הטקסט המפוענח המתקבל מטבלת המיפוי הנוכחית עם מילון עזר בשפה האנגלית (dict.txt)
- הטקסט המפוענח מפוצל למילים בודדות, ולכל מילה, האלגוריתם בודק אם היא קיימת במילון העזר.
- אם נמצאה מילה במילון, הניקוד גדל.
- ציון התאמת המילה הסופי הוא סכום הציונים הבודדים עבור כל המילים בטקסט המפוענח.

2. ציון התאמה בתדירות:
- ציון התאמת התדירות מעריך עד כמה התפלגות התדירות של אותיות וצמדי תווים בטקסט המפוענח תואמת את התפלגות התדירות הצפויה באנגלית.
- הטקסט הפוענח מנותח כדי לחשב את התדירות של אותיות וצמדי תווים בודדים.
- תדירות האותיות וצמדי התווים בטקסט המפוענח מושווה להתפלגות התדירות הצפויה באנגלית, המיוצגת על ידי המילונים 'Letter_Freq.txt', 'Letter2_Freq.txt'.
- עבור כל אות וזוג תווים, האלגוריתם מחשב את ההפרש המוחלט בין התדירות הנצפית בטקסט המפוענח לבין התדירות הצפויה באנגלית.
- אם ההפרש קטן מהאפסילון הנתון (עבור אות בודדת – 0.05, עובר זוג אותיות – 0.005), המונה עבור מספר ההתאמות גדל ב-1.
- ציון התאמת התדירות הוא היחס בין מספר ההתאמות לגודל קובץ התדירויות.

3. כושר כללי:
- הכושר הכולל של פתרון מחושב כסכום המשוקלל של ציון ההתאמה של המילה וציוני התאמת התדירויות.
- ניתן להתאים את המשקולות לציון ההתאמה של המילה וציון ההתאמה בתדירות בהתאם לחשיבותם היחסית. אנו קבענו את לתת משקל של 1 לכל ציון כך שהציון הגבוה ביותר הינו 3. (בפועל הציונים הגיעו לטווח בין 2.4-2.9)
- ערך הכושר הסופי הוא מדד למידת היכולת של האלגוריתם לפענח בהצלחה את הטקסט.

פעולת הcross-over:

1. בחירת פתרונות האב (Selection of Parent Solutions):
- במקרה שלנו תהליך הבחירה מתבצע באמצעות טכניקת הטורניר (tournament selection), איך ישנן טכניקות נוספות.
- ראשית, בוחרים 15% מהפתרונות באקראי ומעבירים אותם אוטומטית לדור הבא.
- לאחר מכן, מחלקים את הפתרונות הנותרים לתתי-קבוצות בגודל 2 או 3 (אם יש יותר מ10,000 פתרונות – לפני הורדת 15% מהם).
- כאשר תת-קבוצה אקראית של פתרונות נבחרת, הפתרון המתאים ביותר (כלומר בעל הכושר הגבוה ביותר) מתוך תת-הקבוצה נבחר כהורה.
- מצוותים את כל פתרונות האב שנבחרו לזוגות (לא כולל את ה-15% שהוצאנו מהתהליך) על מנת לבצע ביניהם תהליך של הצלבה.
2. קביעת נקודת ההצלבה (Cross-over point Determination):
- נקודת ההצלבה היא המיקום בו ישולב החומר הגנטי (מיפוי האותיות) של פתרונות האב.
- ניתן לבחור את נקודת ההצלבה בשיטות שונות. באלגוריתם שלנו נקודת ההצלבה נבחרת באקראי.

3. דור הצאצאים:

- פתרונות הצאצאים נוצרים על ידי שילוב טבלאות המיפוי של פתרונות האב.
- כל שני פתרונות יכולים לייצר מספר משתנה של צאצאים, אך תחילה נסביר כיצד תהליך יצירת הצאצא מתבצע.
- טבלת המיפוי מפוצלת בנקודת ההצלבה, והחלק הראשון של החומר הגנטי מהורה אחד משולב עם החלק השני של החומר הגנטי מההורה השני ליצירת הצאצא.
- תהליך זה מבטיח שכל צאצא יורש חומר גנטי משני ההורים.
- במידה ונקודת ההצלבה אינה באמצע הטבלה, מייצרים צאצא נוסף משני החלקים של החומרים הגנטיים של ההורים שלא נוצלו בעת יצירת הצאצא הראשון (כלומר החלק השני של החומר הגנטי מההורה הראשון משולב עם החלק הראשון של החומר הגנטי מההורה השני).
- בוחרים באקראי האם לייצר צאצא אחד או שניים. אם בחרנו לייצר שני צאצאים, התהליך שתיארנו יחזור על עצמו (כלומר יתכן וייוצרו 4 צאצאים לכל זוג הורים, לא כולל מוטציות).
- בשלב זה ייתכן ותתבצע מוטציה על הפתרון. במידה ואכן מתבצעת מוטציה, היא מתווספת אל מרחב הפתרונות.
- לבסוף, אנו מחליטים האם להשאיר את פתרונות האב לדור הבא. כל פתרון אב נשאר בהסתברות של 35%.
- נציין כי עלול להיוולד צאצא שטבלת המיפוי שלו אינה חד-חד ערכית. בעיה זו נפתרת מעצמה כיוון שהציון שניתן למיפוי שכזה לא יהיה מספיק גבוה כדי להיבחר הלאה ולכן במרוצת הדורות הוא יודח.

מוטציות:

- בוחרים באקראי זוג אותיות מתוך טבלת המיפוי ומחליפים ביניהן (כלומר בין האותיות שהן ממופות אליהן).
- המוטציה מתבצעת בהסתברות של 1% על כל צאצא.
- במידה והמוטציה אכן מתבצעת, הפתרון החדש מתווסף אל מאגר הפתרונות בנוסף לצאצא שעליו התבצעה המוטציה.

בעיית ההתכנסות המוקדמת:

האלגוריתם מטפל בבעיית ההתכנסות המוקדמות באמצעות **גיוון גנטי** (מנגנונים שעוזרים לשמור על אוכלוסייה מגוונת):

- בחירת 15% מהפתרונות באקראי לפני ביצוע פעולת ההצלבה.
- יצירת מספר משתנה של צאצאים עבור כל זוג הורים.
- מוטציה על הצאצאים בהסתברות של 1%.
- השארת ההורים לדור הבא בהסתברות של 35%.

עצירת הריצה:

ישנם מספר גורמים האחראיים על עצירת האלגוריתם:

- **התכנסות**: התכנסות האוכלוסייה נקבעת על ידי מעקב אחר ערכי הכושר של הפרטים לאורך הדורות. אם ערך הכושר המקסימלי לא משתפר במשך 20 דורות, התוכנית תיעצר והפתרון הטוב ביותר עד כה יודפס למסך.
- **מספר דורות מקסימלי**: האלגוריתם יפעל לכל היותר 800 דורות. ברגע שהגבול הזה יושג, האלגוריתם ייפסק גם אם ההתכנסות לא הושגה.
- **צמצום האוכלוסייה**: בסבירות מאוד נמוכה, ייתכן כי האוכלוסייה ההתחלתית תהיה מספיק קטנה כך שבמהלך הריצה היא תצטמצם להיות פחות מ-2 או 3 פרטים. במקרה זה האלגוריתם ייפסק גם אם ההתכנסות לא הושגה.

האלגוריתם הדארויני והלמארקי:

- לכל אלגוריתם יש פונקציית 'improve' בה מתבצע שינוי איטרטיבי של הטבלה בניסיון לצמצם את ההבדל בין הטקסט שנוצר לטקסט הקלט.
- האלגוריתמים הללו מנסים לשפר את ריצת האלגוריתם ע"י בחירת **שני** זוגות של אותיות, החלפה בין ערכיהם וקבלת ההחלפה אם היא משפרת את הfitness.
- הניסיון השיפור מתבצע פעם אחת עבור כל פתרון.
- האלגוריתם **הדארויני** מעביר לדור הבא את ה"גנום" של הפתרון **לפני** האופטימיזציה בעוד **שהלאמרקי** מעביר לדור הבא את הגנום **אחרי** האופטימיזציה.

תוצאות:

הרצנו מספר ריצות של האלגוריתמים השונים כאשר בכל פעם שינינו את גודל האוכלוסייה ההתחלתית.
נציין כי הרצת האלגוריתם עם אותם פרמטרים שוב עלולה להניב תוצאות שונות. אנחנו בחרנו להציג את הפתרון הטוב ביותר שקיבלנו עבור הפרמטרים הללו - דוגמאות ההרצה ותוצאותיהן מופיעות בנספח.
להלן סיכום התוצאות בטבלה (**ערך מקסימלי** (טוב), **ערך מינימלי** (גרוע)):

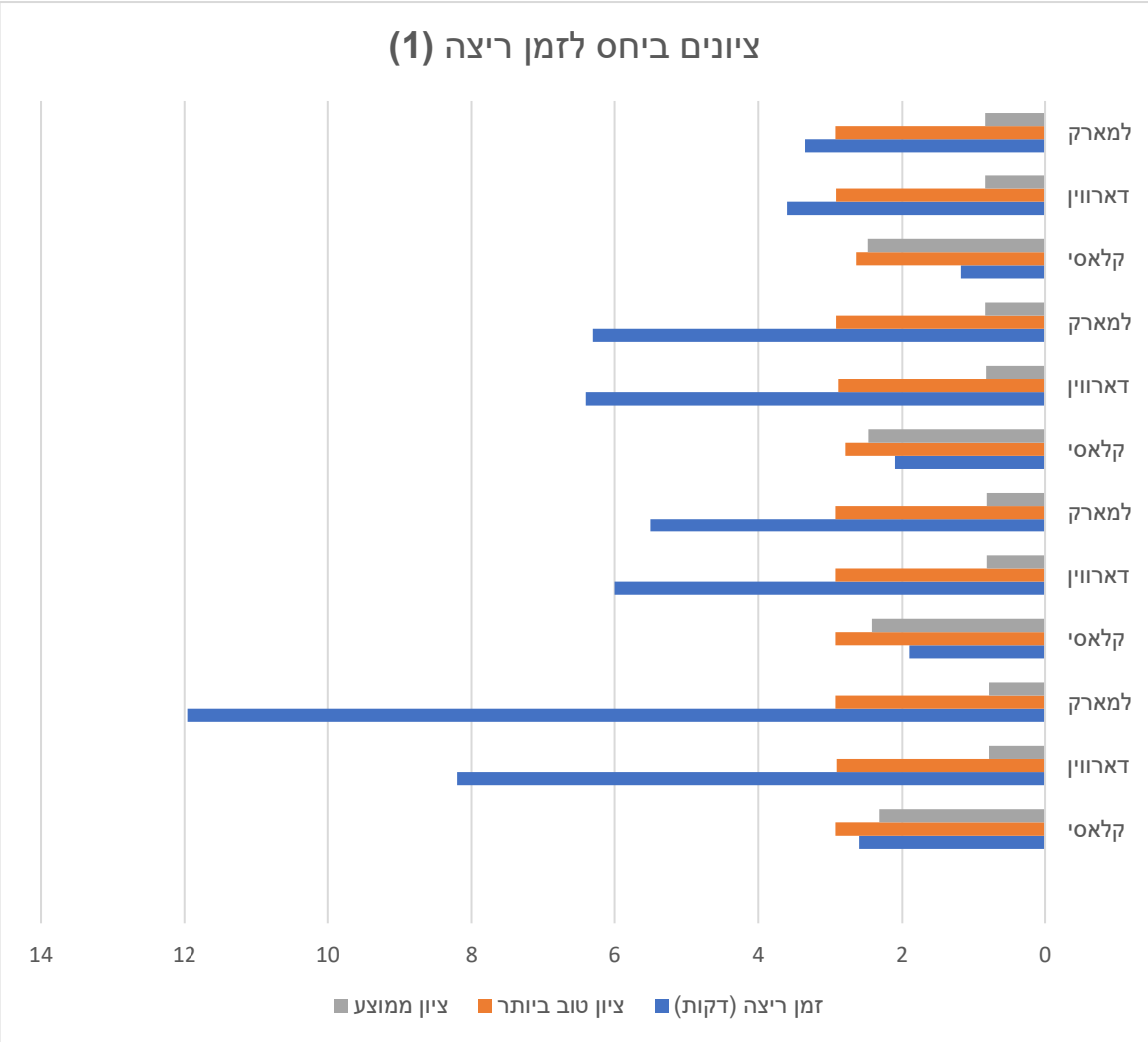
פתרון	**גודל אוכלוסייה התחלתי**	מספר דורות בריצה	מונה פונקציית הערכה	זמן ריצה (דקות)	אחוז התאמה לטקסט	אחוז התאמה לצופן	**ציון טוב ביותר**	ציון ממוצע
קלאסי	20000	75	114054	2.6	99.96%	96.1%	**2.93**	2.32
דארווין	20000	80	346343	8.2	99.8%	96.1%	2.91	**0.78**
למארק	20000	73	350550	**11.96**	**100%**	**100%**	**2.93**	**0.78**
קלאסי*	11000	77	81786	1.9	**100%**	**100%**	**2.93** (2.88 בממוצע)	2.42
דארווין	11000	**90**	250500	6.0	99.8%	92.3%	**2.93**	0.81
למארק	11000	82	238269	5.5	**100%**	**100%**	**2.93**	0.81
קלאסי	9000	86	89106	2.1	98.4%	96.1%	2.79	2.47
דארווין	9000	81	266736	6.4	99.6%	92.3%	2.89	0.82
למארק	9000	**71**	276630	6.3	99.9%	92.3%	2.92	0.83
קלאסי	5000	78	**50789**	**1.17**	**96.8%**	**84.6%**	**2.64**	**2.48**
דארווין	5000	78	150459	3.6	99.7%	88.5%	2.92	0.83
למארק	5000	75	145620	3.35	99.96%	96.1%	**2.93**	0.83

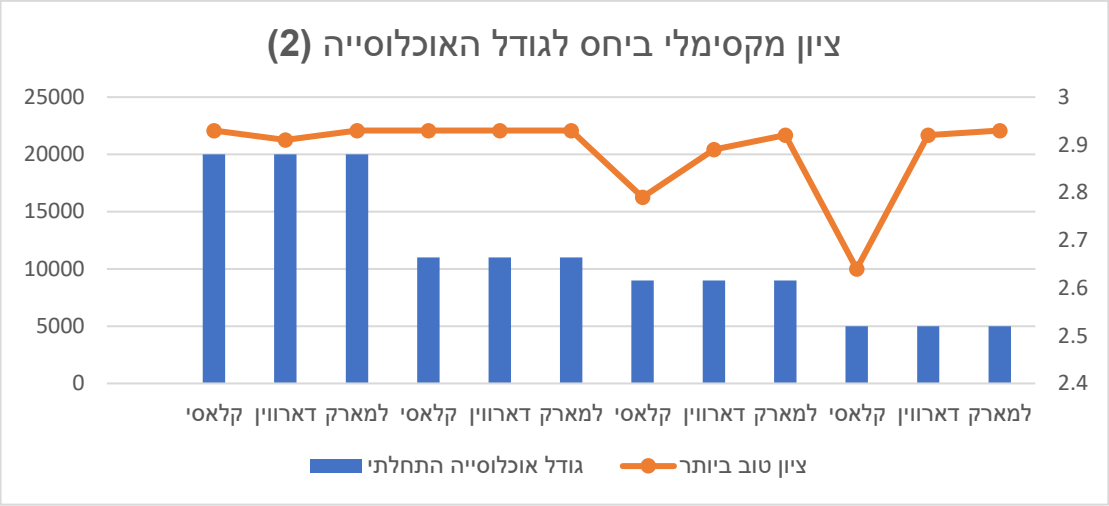
מסקנות:

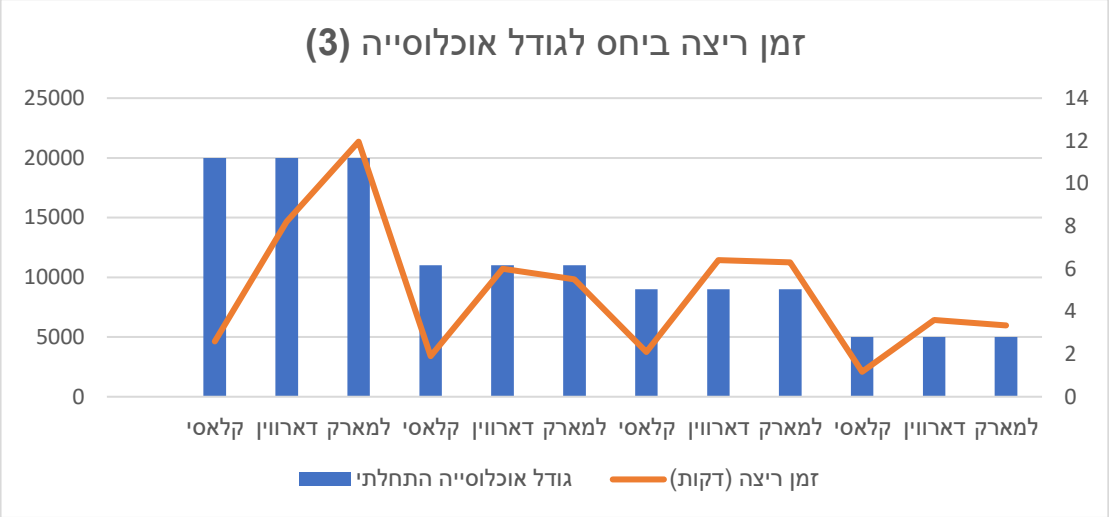
- נראה שגודל האוכלוסייה יכול להכריע איזה אלגוריתם יגיע לביצועים טובים יותר – כלומר אחוזי התאמת טקסט, אחוזי התאמת צופן וציוני fitness מקסימליים גבוהים יותר.
- עבור האלגוריתם הקלאסי - גודל האוכלוסייה של 11,000 יכול לייצר ציונים ואחוזי התאמה גבוהים יותר בהשוואה לגודל האוכלוסייה של 9,000, 20,000, ו5,000. עם זאת, אוכלוסייה בגודל של 20,000 תניב תוצאות טובות באופן יותר עקבי. לשם כך, צירפנו לנספח דוגמאות הרצה של האלגוריתם הקלאסי עם גודל אוכלוסייה של 11,000 שהניבו תוצאות נמוכות יותר (ציון ממוצע 2.88).

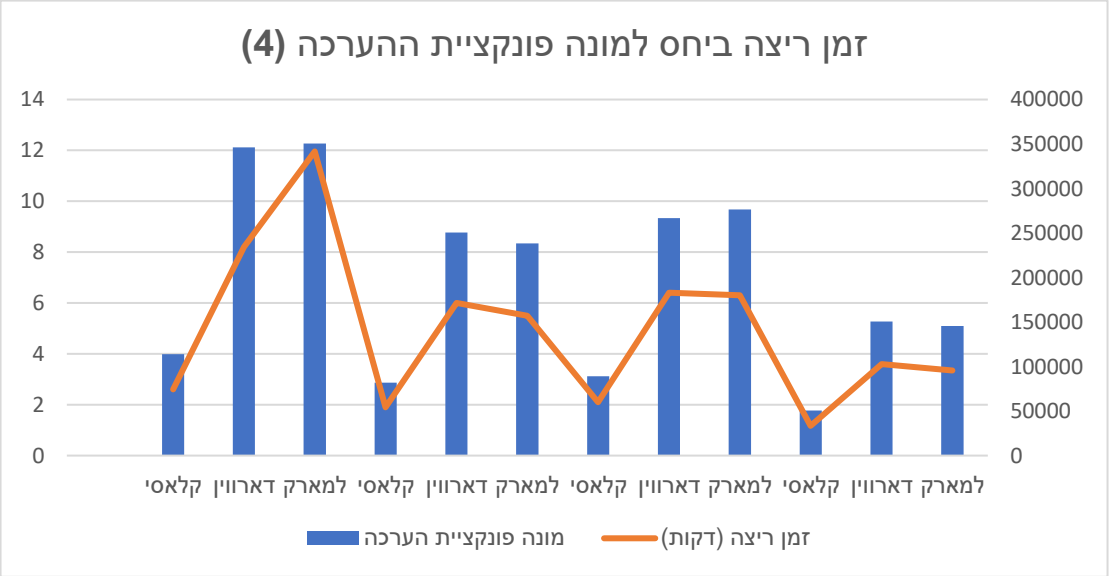
- האלגוריתמים של למארק ודארווין מגיעים לרוב לביצועים די טובים ללא קשר לגודל האוכלוסייה, אך במחיר של זמן ריצה ארוך פי 3 ולפעמים אף יותר, מזה של האלגוריתם הקלאסי **(1)** (באופן דומה אפשר לשים לב למספר הפעמים שפונקציית ההערכה נקראה **(4)**).
- זמן ריצה קצר מאוד עלול להניב תוצאה סופית נמוכה. **(1)**
- ציון ממוצע גבוה אינו מעיד על ציון סופי גבוה. **(1)**
- ביצועי האלגוריתמים אינם בהכרח משתפרים כאשר מגדילים את גודל אוכלוסייה (למשל מ11,000 ל-20,000). **(2)**
- זמן הריצה של האלגוריתם הקלאסי לא מושפע בצורה משמעותית כמו שאר האלגוריתמים מגודל האוכלוסייה ההתחלתית. **(3)**
- עבור גודל אוכלוסייה קטן יותר מ-11,000, ניתן לראות שביצועיו של האלגוריתם הקלאסי, היינו ציון fitness, ואחוזי התאמה לטקסט ולצפון, יורדים ברמתם לעומת ביצועי שני האלגוריתמים האחרים, ובפרט, האלגוריתם הלמארקי מגיע לתוצאות הכי טובות עבור פרמטרים הללו. **(2)**
- האלגוריתם הלמארקי תמיד מגיע לביצועים טובים יותר מאלו של האלגוריתם הדארוויני. **(2)**
- זמני הריצה, והציונים הממוצעים של האלגוריתמים הדארויני והלמארקי תמיד פחות טובים מאלו של האלגוריתם הקלאסי. **(1)**
- מספר הדורות נע בין 70 ל-90, כלומר הפתרונות מתכנסים תוך 50-70 דורות.
- במקרה של הטקסט שקיבלנו ציון fitness של 2.93 לרוב מעיד על פיתוח הצופן בצורה מושלמת, הציון 3 לא מתקבל בגלל פערים בין הטקסט לקבצי המילון והתדירויות. בטקסטים אחרים ייתכן והציון עבורו פיצוח הצופן יעשה בצורה המושלמת יהיה שונה. **(5)**

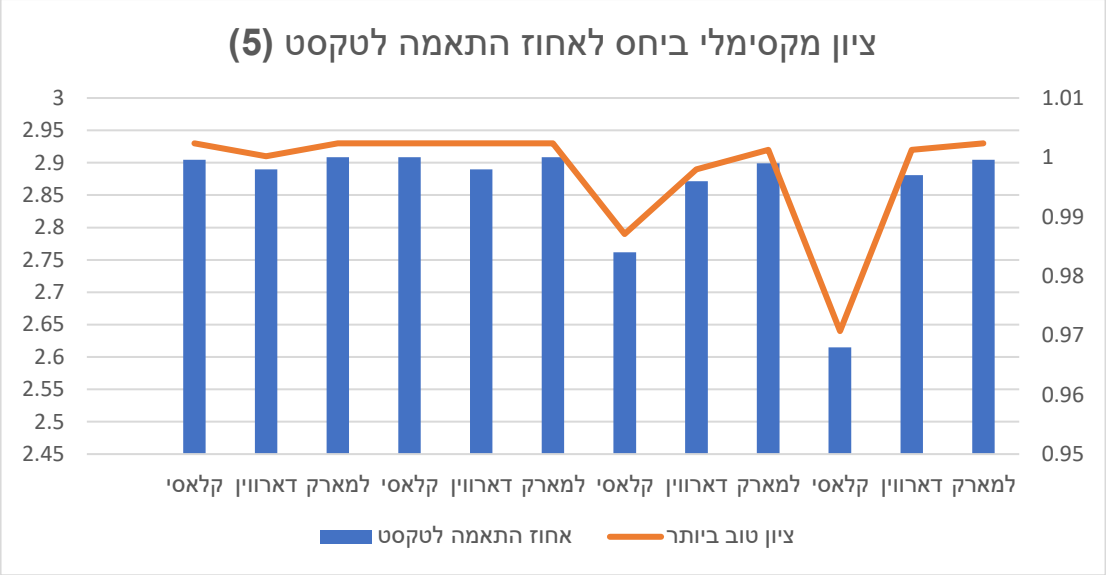
נסכם באמצעות גרפים:




ציון מקסימלי ביחס לגודל האוכלוסייה (2)
25000
20000
15000
10000
5000
0
3
2.9
2.8
2.7
2.6
2.5
2.4
קלאסי דארווין למארק קלאסי דארווין למארק קלאסי דארווין למארק קלאסי דארווין למארק
גודל אוכלוסייה התחלתי
ציון טוב ביותר


זמן ריצה ביחס לגודל אוכלוסייה (3)
25000
20000
15000
10000
5000
0
14
12
10
8
6
4
2
0
קלאסי דארווין למארק קלאסי דארווין למארק קלאסי דארווין למארק קלאסי דארווין למארק
גודל אוכלוסייה התחלתי
זמן ריצה (דקות)


זמן ריצה ביחס למונה פונקציית ההערכה (4)
14
12
10
8
6
4
2
0
400000
350000
300000
250000
200000
150000
100000
50000
0
קלאסי דארווין למארק קלאסי דארווין למארק קלאסי דארווין למארק קלאסי דארווין למארק
מונה פונקציית הערכה
זמן ריצה (דקות)


ציון מקסימלי ביחס לאחוז התאמה לטקסט (5)
3
2.95
2.9
2.85
2.8
2.75
2.7
2.65
2.6
2.55
2.5
2.45
1.01
1
0.99
0.98
0.97
0.96
0.95
קלאסי
דארווין
למארק
קלאסי
דארווין
למארק
קלאסי
דארווין
למארק
קלאסי
דארווין
למארק
אחוז התאמה לטקסט
ציון טוב ביותר

נספח:

גודל אוכלוסייה: 5,000

```
PS S:\code\Computational Biology\GeneticAlgorithm\GeneticAlgorithm\src> python3 geneticAlgorithm.py 0 5000
Best fit score: 2.6422041420118343
Avg fit score:2.482442144614784
Generation: 78
Fitness counter: 50789
Time:70.17947816848755 seconds
```

```
PS S:\code\Computational Biology\GeneticAlgorithm\GeneticAlgorithm\src> python3 geneticAlgorithm.py 2 5000
Best fit score: 2.928224852071006
Avg fit score:0.8308045370580316
Generation: 75
Fitness counter: 145620
Time:201.14749789237976 seconds
```

```
PS S:\code\Computational Biology\GeneticAlgorithm\GeneticAlgorithm\src> python3 geneticAlgorithm.py 1 5000
Best fit score: 2.920266272189349
Avg fit score:0.8290545655147323
Generation: 78
Fitness counter: 150459
Time:216.0130066871643 seconds
```

גודל אוכלוסייה: 9,000

```
PS S:\code\Computational Biology\GeneticAlgorithm\GeneticAlgorithm\src> python3 geneticAlgorithm.py 0 9000
Best fit score: 2.7907248520710057
Avg fit score:2.47142397685517
Generation: 86
Fitness counter: 89106
Time:124.44172310829163 seconds
```

```
PS S:\code\Computational Biology\GeneticAlgorithm\GeneticAlgorithm\src> python3 geneticAlgorithm.py 1 9000
Best fit score: 2.888224852071006
Avg fit score:0.8249999910711089
Generation: 81
Fitness counter: 266736
Time:382.8073899745941 seconds
```

```
PS S:\code\Computational Biology\GeneticAlgorithm\GeneticAlgorithm\src> python3 geneticAlgorithm.py 2 9000
Best fit score: 2.923224852071006
Avg fit score:0.8346242317456919
Generation: 71
Fitness counter: 276630
Time:379.9925274848938 seconds
```

תוצאות שונות עבור האלגוריתם הקלאסי עם גודל אוכלוסייה: 11,000

```
PS S:\code\Computational Biology\GeneticAlgorithm\GeneticAlgorithm\src> python3 geneticAlgorithm.py 0 11000
Best fit score: 2.933224852071006
Avg fit score:2.4234134429989505
Generation: 77
Fitness counter: 81786
Time:116.26139903068542 seconds
```

```
PS S:\code\Computational Biology\GeneticAlgorithm\GeneticAlgorithm\src> python3 geneticAlgorithm.py 0 11000
Best fit score: 2.7797041420118345
Avg fit score:2.419972951247152
Generation: 86
Fitness counter: 82995
Time:178.69510769844055 seconds
```

```
PS S:\code\Computational Biology\GeneticAlgorithm\GeneticAlgorithm\src> python3 geneticAlgorithm.py 0 11000
Best fit score: 2.8892455621301774
Avg fit score:2.4080145993362847
Generation: 76
Fitness counter: 81728
Time:114.75606966018677 seconds
```

```
PS S:\code\Computational Biology\GeneticAlgorithm\GeneticAlgorithm\src> python3 geneticAlgorithm.py 0 11000
Best fit score: 2.908224852071006
Avg fit score:2.424787328688018
Generation: 79
Fitness counter: 83666
Time:117.3335907459259 seconds
```

גודל אוכלוסייה: 11,000 (דארווין ולמארק)

```
PS S:\code\Computational Biology\GeneticAlgorithm\GeneticAlgorithm\src> python3 geneticAlgorithm.py 1 11000
Best fit score: 2.9267455621301774
Avg fit score:0.8136728647438841
Generation: 90
Fitness counter: 250500
Time:361.5360767841339 seconds
```

```
PS S:\code\Computational Biology\GeneticAlgorithm\GeneticAlgorithm\src> python3 geneticAlgorithm.py 2 11000
Best fit score: 2.933224852071006
Avg fit score:0.8065015219112032
Generation: 82
Fitness counter: 238269
Time:329.09780621528625 seconds
```

גודל אוכלוסייה: 20,000

```
PS S:\code\Computational Biology\GeneticAlgorithm\GeneticAlgorithm\src> python3 geneticAlgorithm.py
Best fit score: 2.928224852071006
Avg fit score:2.316533036282096
Generation: 75
Fitness counter: 114054
Time:156.3733720779419 seconds
```

```
PS S:\code\Computational Biology\GeneticAlgorithm\GeneticAlgorithm\src> python3 geneticAlgorithm.py 1 20000
Best fit score: 2.9142455621301773
Avg fit score:0.7799159092967253
Generation: 80
Fitness counter: 346434
Time:494.1617224216461 seconds
```

```
PS S:\code\Computational Biology\GeneticAlgorithm\GeneticAlgorithm\src> python3 geneticAlgorithm.py 2 20000
Best fit score: 2.933224852071006
Avg fit score:0.7826808018928832
Generation: 73
Fitness counter: 350550
Time:717.4961605072021 seconds
```

הרצה על טקסט אחר (ציון מקסימלי 2.5):

```
PS S:\code\Computational Biology\GeneticAlgorithm\GeneticAlgorithm\src> python3 geneticAlgorithm.py 0 30000
Best fit score: 2.4971372014814186
Avg fit score:2.1697478777065315
Generation: 86
Fitness counter: 136575
Time:190.30846166610718 seconds
```

```
PS S:\code\Computational Biology\GeneticAlgorithm\GeneticAlgorithm\src> python3 geneticAlgorithm.py 0 20000
Best fit score: 2.4971372014814186
Avg fit score:2.293253270361216
Generation: 83
Fitness counter: 115623
Time:145.4735677242279 seconds
```

```
PS S:\code\Computational Biology\GeneticAlgorithm\GeneticAlgorithm\src> python3 geneticAlgorithm.py 0 11000
Best fit score: 2.4224383806564216
Avg fit score:2.3762316904098215
Generation: 83
Fitness counter: 80833
Time:109.36550378799438 seconds
```

```
PS S:\code\Computational Biology\GeneticAlgorithm\GeneticAlgorithm\src> python3 geneticAlgorithm.py 2 5000
Best fit score: 2.3763143331488656
Avg fit score:0.8272623923147335
Generation: 81
Fitness counter: 151842
Time:128.3097686767578 seconds
```