אלגוריתמים אבולוציוניים לפיצוח צופן החלפה בטקסט

נעה שחר דלטיצקי 209134881 (ביולוגיה חישובית)
יונתן פייגל 208296210 (מדעי המחשב)

(תרגיל משולב)

מבנה הקוד, ואיך פתרונות מיוצגים

קובץ הקוד הראשי (שהוא חלק א׳ של התרגיל) מחולק לרכיבים הבאים:

dna

גנום מיוצג ע״י מופע של הקלאס DNA.

המידע שבו הוא מערך המכיל 26 אותיות (כאשר בתא הראשון המיפוי של a, בתא השני המיפוי של b, וכו׳).

נוסף על כך, אובייקט גנום מכיל פונקציות, בהן: Make-Crossover, Mutate-In-Place, Map-Text.

fitness

כאן יש את המימוש של פונקציית הפיטנס, שמחשבת את ה-״englishness״ של טקסט.

הסבר מפורט עליה בעמוד הבא. בינתיים נגיד שהיא נסמכת על קובץ מצורף בשם **freq-triplets.json** שמחזיק את התדירות של שלשות של אותיות באנגלית (למשל ל-״ing״ יש ציון גבוה, ול-״rwh״ יש ציון נמוך).

genetic_algorithm

כאן נמצא הלב של האלגוריתם, כלומר הלולאה הראשית שמחזיקה את אוכלוסיית הגנומים הקיימת, ומחליטה על מי מהם לבצע קרוסאובר, מוטציה, וכו. הפונקציה הראשית נקראית solve.

main

טיפול בארגומנטים משורת הפקודה (של האם verbose), קבלת קובץ לפיענוח, וקריאה ל-solve.

פונקציית ההערכה

פונקציית הפיטנס מודדת עד כמה טקסט "נראה באנגלית".

הקלט שלה הוא גרסה של ה-ciphertext שעברה מיפוי בידי DNA כלשהו, והפלט שלה הוא מספר.

לאחר ניסיונות רבים (שכללו ספירת זוגות אותיות, מילים שלמות, אותיות יחידות...) ולאחר שיטוטים באינטרנט, מצאנו שפונקציית ההערכה שעובדת לנו הכי טוב היא סכימה של ניקוד של שלשות רצופות של אותיות.

כאמור, בקוד יש קובץ בשם **freq-triplets.json** שתחילתו נראית כך:

{ "**the**": 3396, "**and**": 1852, "**ing**": 1499, "**her**": 1084, "**was**": 823, "**hat**": 814, "**ere**": 760, "**his**": 701, "**tha**": 689, "**you**": 651, "**ent**": 576, "**ver**": 565, ... }

הפונקציה עוברת על שלשות של אותיות רצופות בטקסט, וסוכמת את ה-log של הניקוד שלהן.

מצאנו שעידון הניקוד (ע"י לקיחת log שלו) מביא ליציבות רבה יותר ותורם להימנעות מהתכנסות מוקדמת.

דוגמאות לחישוב הפיטנס של טקסט:

fitness("**nice to be here**")

= log(score["nic"]) + log(score["ice"]) + log(score["her"]) + log(score["ere"])

= log(58) + log(162) + log(1084) + log(760) = **22.76977**

fitness("**ze mitz agvaniot**")

= log(score('mit')) + log(score('itz')) + log(score('agv')) + log(score('gva')) + log(score('van')) + log(score('ani')) + log(score('nio')) + log(score('iot')) = **13.40068**

נשים לב שלמרות שהטקסט השני מכיל יותר שלשות, הניקוד שלו נמוך יותר, כי הוא נראה פחות "באנגלית".

השיטה שלנו מתעלמת לחלוטין ממילים שאורכן 2. לכאורה חבל לפספס מילות זהב כמו is/be/to/in;

אך כשהכנסנו לחישוב גם תדירויות של זוגות, מצאנו שהדבר לא תורם ליעילות האלגוריתם בפועל.

כיצד אנחנו מבצעים cross-over בין שני גנומים

אנחנו לוקחים 5 גנים (באקראי) מהורה אחד, ואת השאר מההורה השני.

לאחר מכן, ייתכן שהגנים החדשים לא תקינים (כלומר, שהמערך מכיל את אותה אות יותר מפעם אחת, וגרוע מכך, יש אותיות שחסרות בו).

כדי לטפל בבעיית הגנום הלא תקין שנוצר,

ראשית נמצא את הגנים שחסרים (בקוד קוראים להם "gone") ואת הגנים הכפולים (בקוד: "dups").

כעת, עד שהקבוצה dups תהיה ריקה,

נחליף מופע של גן כפול, באחד הגנים שחסרים, ובכך נקטין את הקבוצות gone, dups.

להלן הפונקציה הרלוונטית מהקוד:

```
def a_crossover_between(dna_1, dna_2):
    merged_genes = dna_2.genes.copy()

    parent_1_indices = random.sample(range(len(dna_1.genes)), k=MERGE_PARENT_1_GENES)

    new = set(map(dna_1.genes.__getitem__, parent_1_indices))
    old = set(map(dna_2.genes.__getitem__, parent_1_indices))

    gone = old - new  # letters that will be lost
    dups = new - old  # letters that we'll have more than once

    # assign some of parent_1's genes into the copy of parent_2's genes
    for idx in parent_1_indices:
        merged_genes[idx] = dna_1.genes[idx]

    # fix the dups/gone genes
    for idx in range(len(dna_2.genes)):
        if (dup := merged_genes[idx]) in dups:
            merged_genes[idx] = gone.pop()
            dups.remove(dup)
            if not dups:
                break

    return DNA(merged_genes)
```

איך התייחסנו לבעיית ההתכנסות המוקדמת

נקטנו כמה טכניקות, בהן:

1. האוכלוסייה של הדור הבא מורכבת מ-15% elite, ומעוד 85% crossovers של פריטים מכלל האוכלוסייה הקיימת. כדי להחליט איזה הורים יזכו להביא צאצאים, אנחנו מקיימים **הרבה טורנירים קטנים** של 5 משתתפים (משתתפי הטורניר נבחרים בהסתברות שווה מהאוכלוסייה).

   ל-DNA עם פיטנס גבוה יש סיכוי רב יותר לנצח בטורניר, אך מנגד, העובדה שהטורניר כל כך קטן מאפשרת **גיוון גנטי** רב יותר בדור הבא.

   מצאנו שבמיוחד עבור טקסטים קצרים יחסית, טורנירים בגודל 5 מצליחים למנוע הרבה מקרים של התכנסות מוקדמת.
2. הלולאה הראשית (שתפקידה ביצוע מוטציות ומדידת פיטנס) עוצרת רק אחרי שעברו 100 דורות רצופים שבהם לא נרשם שום שיפור בפיטנס. כלומר, היא נותנת הרבה זמן למוטציות אקראיות למצוא מקום לשיפור, במידה ויש כזה.
3. בהמשך לסעיף 2, הגדלנו את הסיכוי למוטציות אקראיות (שקורות לאחר crossover).
4. עידון הפיטנס (הרחבנו על כך בעמוד "פונקציית ההערכה").

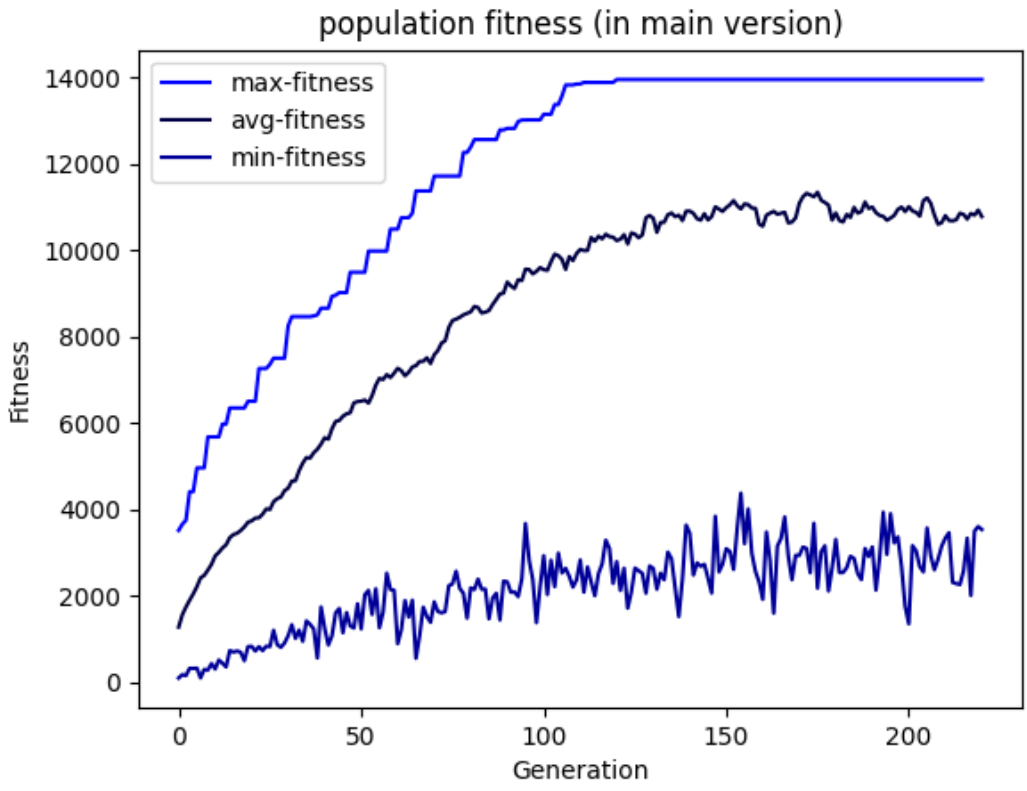
   הפיטנס קשור קשר הדוק לסיכוי לנצח בטורנירים, ומי שמנצח בהם מרכיב את מרבית האוכלוסיה).

   עידון הפיטנס מאפשר לפריטים הפחות "מוצלחים" לנצח לעיתים קרובות יותר, מה שתורם גם הוא לגיוון גנטי.

גרפים והשוואות

לאלגוריתם הראשי לוקח [90-120] דורות להגיע לפיתרון של הטקסט שניתן בתרגיל.

אח״כ הוא ממשיך לרוץ למשך עוד 100 דורות כדי לוודא ששום מוטציה לא מוצאת פיתרון יותר טוב לפני שהוא עוצר.

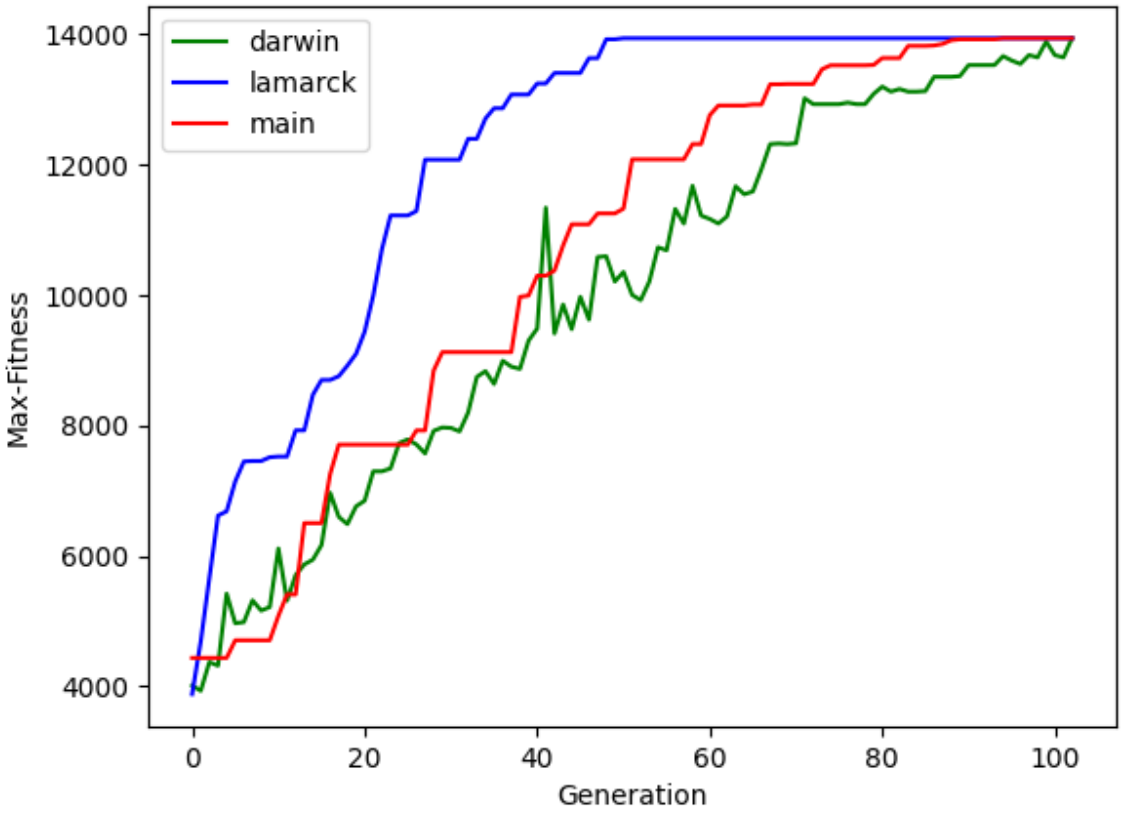


(גרפים נוספים בעמוד הבא)

האלגוריתם בשיטת למארק מצליח להגיע לפיתרון תוך חצי ממספר הדורות שלוקח לאלגוריתם הראשי;

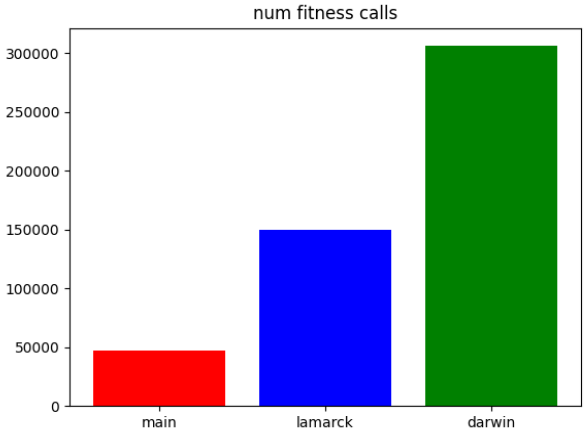
בגרף לעיל ניתן לראות את מספר הדורות שלקח לכל אחת מהשיטות להגיע לפיתרון.

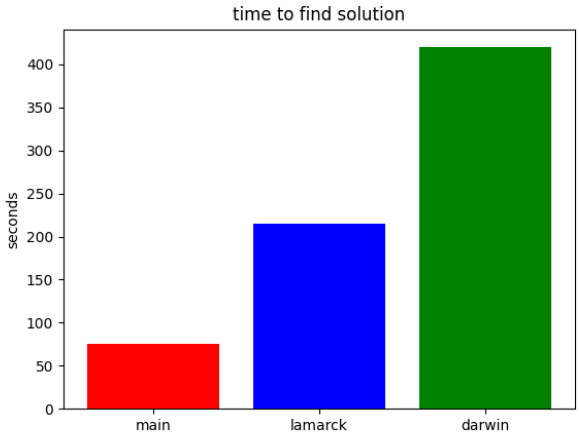
ניתן גם לראות את ה-spikes בגרף של darwin; כל spike כזה מלמד שחושב פיטנס גבוה, מפני שנמצא שינוי בהתנהגות של הפרט שמוביל לשיפור משמעותי, אך השיפור הזה לא הועבר לדור הבא.



למרות שהאלגוריתם בשיטת למארק מגיע לפיתרון תוך פחות דורות, זמן ריצתו בפועל ארוך פי כמה וכמה, בשל מספר הקריאות הרב לפונקציית הפיטנס (אפילו כשהסתפקנו ב-5 קריאות לכל פרט בכל דור).

נראה שאצלנו האלגוריתם בשיטת דארווין הפגין ביצועים נחותים על פני האחרים בכל המדדים.





היפר-פרמטרים

יש הרבה היפר-פרמטרים לשחק איתם.

בחרנו הצבה שלהם שעבדת טוב על טקסטים באורך של הטקסט מהמודל.

עבור טקסטים קצרים יותר, כדאי להגדיל את ה-PLATEAU_TOLLERANCE, וגם את הסיכוי למוטציות.

להלן כל ההיפר-פרמטרים שלנו (הם מוגדרים בראש הקובץ part_a.py ואפשר לשחק איתם):

This many genes (out of 26) will come from parent 1 on crossover

MERGE_PARENT_1_GENES = 5

We'll have this many DNA's competing every generation.

POPULATION_SIZE = 500

This many places in the next generation are for the fittest dna's.

ELITISM_SIZE = 76 (~15% of population)

This many places in the next generation are for crossovers between existing dna's

CROSSOVER_SIZE = 424 (rest of population)

Every new crossover will have this chance to mutate.

MUTATION_CHANCE = 0.5

If in this last many generations, max fitness hasn't increased, halt.

PLATEAU_TOLLERANCE = 100

The larger, the better chances the fitter will have.

TOURNAMENT_SIZE = 5

בטורניר, המשקל של כל מתמודד הוא לא בדיוק הפיטנס שלו,

אלא הפיטנס כפול קבוע דעיכה אקספוננציאלי, שמנמיך אפילו יותר את הסיכוי של המתמודדים ה"גרועים".

המשקל של המתמודד הכי איכותי בטורניר הוא בדיוק הפיטנס שלו;

המשקל של המתמודד השני באיכותו בטורניר הוא הפיטנס שלו פחות 15%,

המשקל של המתמודד השלישי באיכותו בטורניר הוא הפיטנס שלו פחות 27.75%,

וכן הלאה, כמבואר כאן:

β controls the exponential decay on the fitnesses in a tournament.

0.00 == no decay: the chance to win is proportional to one’s fitness.

1.00 == decay at max: the tournament's fittest will always win.

β = 0.15

The tournament's competitors are sorted by fitness,

and the bad one are further punished,

as their fitness will then be multiplied by a small number.

EXPONENTIAL_DECAY = [(1 - β) ** 0, (1 - β) ** 1, (1 - β) ** 2, …, (1 - β) ** 5]