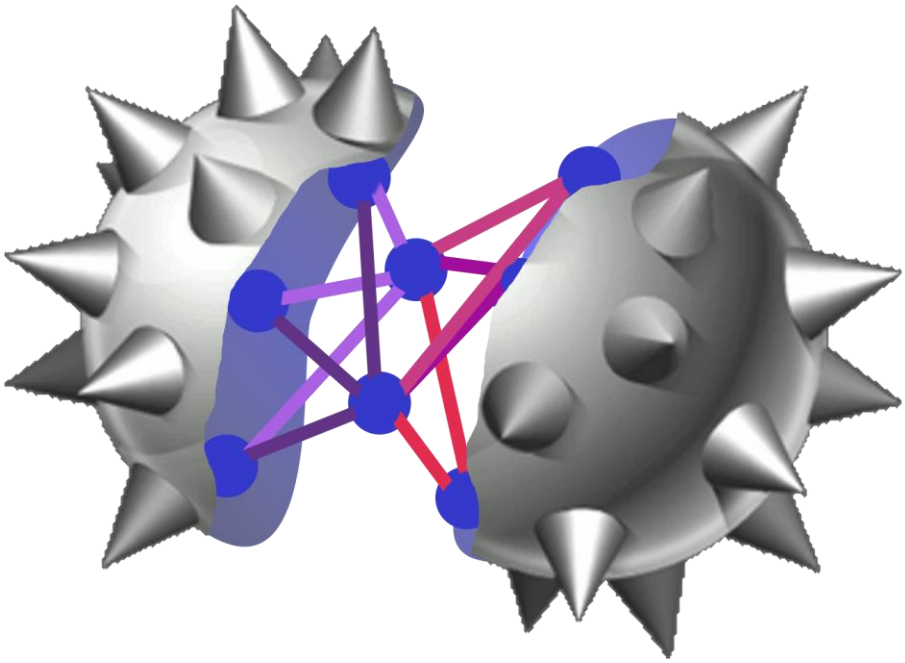
Minesweeper



Local Search & Genetic Algorithm

מרצה - ד''ר תמר שרוט

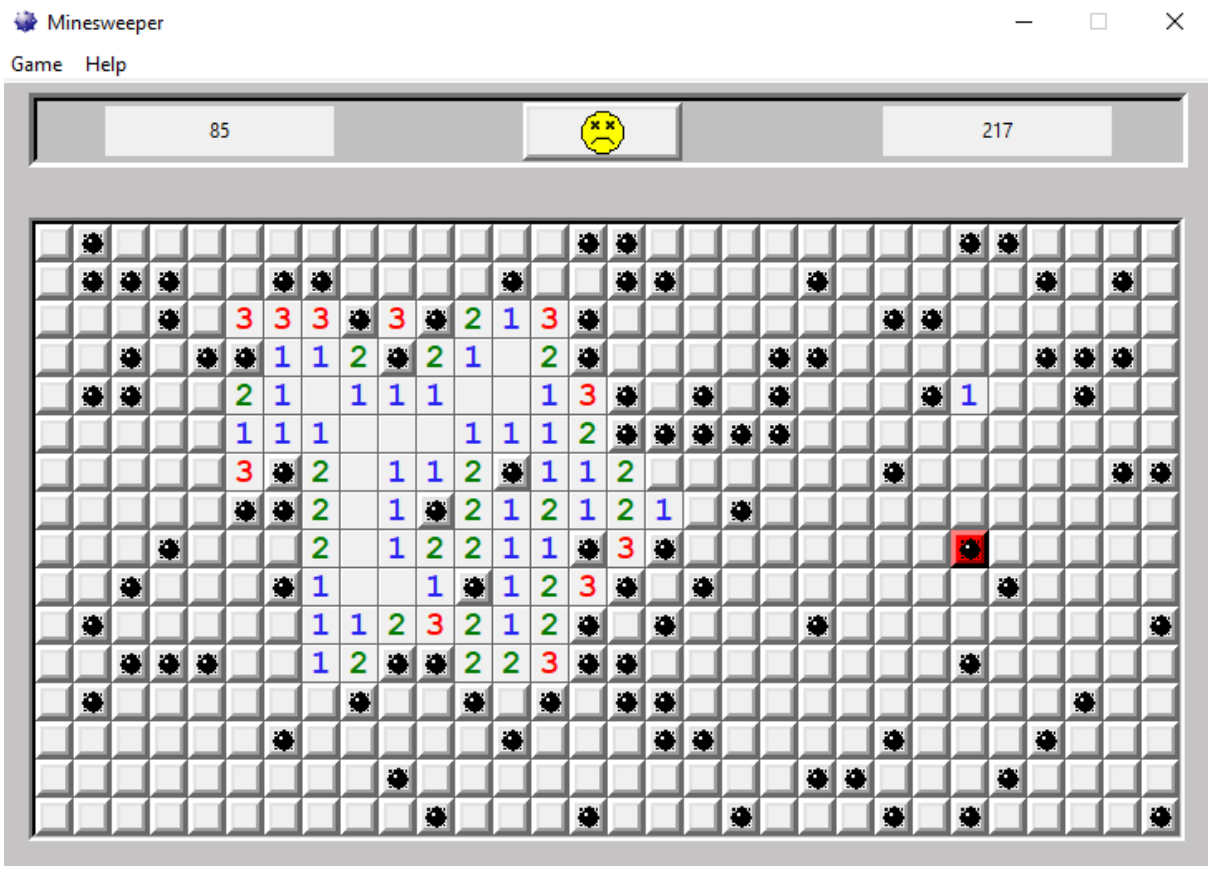
קורס - מבוא לבינה מלאכותית

מגישות-
שרון וזאנה
שירה אביטל מהגרפטה

Table Of Contents

The Problem

שולה המוקשים הוא משחק לוח לשחקן יחיד , לוח המשחק הוא לוח מרובע ,המחולק למשבצות.
ממדיו של הלוח אינם קבועים ומשפיעים על מידת הקושי של המשחק. בלוח מפוזרים "מוקשים" בלתי נראים במיקומים אקראיים עם הלחיצה הראשונה על הלוח.
הלוח מתחלק ל-2 סוגי משבצות: משבצות עם מוקשים ומשבצות ללא מוקשים. משבצות ללא מוקשים מתחלקות גם הן ל-2 סוגים: משבצות עם מספר ובלי מספר.
כל משבצת שבה אין מוקש תכיל מספר האומר כמה מוקשים יש סביבו. אם אין מספר, תיפתח קבוצה של משבצות - כל המשבצות הריקות עד לגבול שבו יש משבצת עם מספר, כולל המשבצות עם המספר. מטרת המשחק היא לחשוף את כל המשבצות נטולות המוקשים.
כאשר השחקן לוחץ על משבצת כלשהי, המשבצת נחשפת. אם יש במשבצת מוקש, המשחק נגמר.
אם חושפים משבצת שאין עליה מספר או מוקש, כל המשבצות הריקות בסביבתה נחשפות גם כן, עד שנוצר שטח של משבצות ריקות, המוקפות במשבצות ממוספרות.



הבעיה אותה אנו באות לפתור זה פתרון המשחק עם כמות צעדים המינימלית האפשרית, אנחנו לא יודעים לאמוד את כמות המשאבים הנדרשת לפתרון המשחק , המשאבים שלנו הם כמות הלחיצות והסדר בו מבצעים אותן - מה שנפתור באמצעות האלגוריתמים שנבחרו.
במשחק שולה המוקשים בכל לוח ישנן אפשרויות רבות לפתרון כאשר פתרון מורכב מהתאים עליהם נלחץ, ככל שהלוח גדל כך גם מספר האפשרויות לפתרון גדל ובהתאמה גם מספר הצעדים בפתרון גדל, נרצה למצוא את הפתרון של המשחק בעל כמות הצעדים המינימלית.

לאחר שהבנו איך המשחק פועל וחקרנו על מגוון שיטות לבינה מלאכותית שנלמדו בשיעור וגם כאלה שלא ראינו כי ישנן מספר שיטות לפתרון הבעיה - רשת נוירונים , חיפוש מקומי , אלגוריתם גנטי וכו..
התמקדנו בשלושת השיטות האלה ולבסוף החלטנו לממש רק שתיים מהשיטות על בעיית המשחק שלנו - שולה המוקשים.

האלגוריתמים שבחרנו: אלגוריתם גנטי, וחיפוש מקומי מסוג טיפוס גבעות.

במהלך כל הפרויקט נדון בהחלטתנו על שתי גישות האלה.

בחרנו בשיטות אלו מכיוון שלאחר שחקרנו לעומק מגוון שיטות אחרות כמו רשת נוירונים , אלגוריתם בייסיאני, csp ו- double set single point הגענו לכך שחלק מהשיטות היו קשות לנו לביצוע וחיבור עם הפרויקט וחלקן היו בינה מלאכותית ברמה מאוד נמוכה, כשהגענו לשתי השיטות שנבחרו הרגשנו שאנחנו מספיק מסוגלות לבצע אותן, להבין אותן ולנתח אותן בצורה עמוקה ובנוסף הן גם מאוד עניינו אותנו ושתיהן עוזרות לנו בפתרון בעיות אופטימיזציה.

אופטימיזציה

אופטימיזציה זוהי פעולה המאפשרת להשיג את התוצאה הקרובה ביותר למטרה שהצבנו בהינתן הכלים שברשותנו ותחת אילוצים מוגדרים (כמו למשל בכמה שפחות זמן וכסף).

Genetic Algorithm

אלגוריתם גנטי (GA) הוא מטאוריסטיקה בהשראת תהליך הברירה הטבעית השייך למעמד הגדול יותר של אלגוריתמים אבולוציוניים (EA). אלגוריתמים גנטיים משמשים בדרך כלל ליצירת פתרונות באיכות גבוהה לבעיות אופטימיזציה וחיפוש על ידי הסתמכות על אופרטורים בעלי השראה ביולוגית כגון מוטציה, הצלבה ובחירה.
באלגוריתם הגנטי, אוכלוסיית פתרונות מועמדים (הנקראים אינדיבידואלים, יצורים או פנוטיפים) לבעיית אופטימיזציה מתפתחת לעבר פתרונות טובים יותר.

לכל פתרון מועמד יש קבוצה של מאפיינים (הכרומוזומים או הגנוטיפ שלו) הניתנים למוטציה ולשינוי.

באופן מסורתי, פתרונות מיוצגים בצורה בינארית כמחרוזות של 0-ים ו- 1-ות, אך קידודים אחרים אפשריים גם הם.

במקום לייצר פתרון אקראי יחיד ולשפר אותו, אנחנו יוצרים "אוכלוסייה" של פתרונות. אנו מחשבים את האיכות ("כושר") של כל פתרון. באיטרציה הבאה, אנו משיגים "אוכלוסיה" חדשה על ידי הדמיית תהליך הברירה הטבעית: לפתרונות בעלי כושר טוב יותר יש יותר צאצאים.
כמו באבולוציה הביולוגית, כל פתרון באוכלוסייה החדשה מבוסס על שילוב שני "הורים" מהאוכלוסייה הוותיקה בדרך כלשהי ("הצלבה").
אנו מיישמים גם מוטציות - שינויים אקראיים בצאצאים.

אלגוריתם גנטי לפתרון בעיית "שולה המוקשים", תחילה בשלב האתחול ה Agent שלנו ישחק במשחק עד לקבלת פתרון - רצף צעדים שפותרים את המשחק ומנצחים אותו ,מתבצע בקובץ
. generate_population.py

הפתרון שלנו בעצם מורכב מרשימה של צעדים כל צעד מהווה גן (Gen) בפתרון, כל פתרון הוא חלק מהאוכלוסייה (Population) ואוכלוסייה היא תת קבוצה של פתרונות בדור הנוכחי. היא גם יכולה להיות מוגדרת כקבוצה של כרומוזומים, וכמה אוכלוסיות ביחד הן בעצם הדור שלנו (Generation).

אתחול אוכלוסייה, אתחול רנדומלי – מאכלס את האוכלוסייה ההתחלתית עם פתרונות רנדומליים לחלוטין.

לאחר שמצאנו שני פתרונות טובים נגדיר אותם כתור ההורים שלנו , עם הגנים של ההורים עושים שילובים (crossover) ויוצרים תינוקות, לתינוקות עושים מוטציות (mutation) – שינויים בגנים.

נריץ את המשחק שוב עם אוכלוסייה שמכילה את התינוקות שיצרנו, התינוקות שיביאו לנו את מספר הצעדים הנמוך ביותר עד לניצחון במשחק ברוב המקרים ייבחרו להיות ההורים החדשים ובמעט מקרים ייבחרו דווקא לא הטובים ביותר כדי להרחיב את "בריכת הגנים" ולמנוע מהאלגוריתם להיתקע.

נחזור על התהליך הזה מספר פעמים עד שנתכנס לפתרון או עד הגעה לתנאי הנוסף לעצירה שהוא כמות הדורות , אצלנו בתוכנה החלטנו להגביל את זה עד ליצירת חמש דורות , ראינו שגם כאשר הוספנו יותר דורות ברוב המקרים כבר בדור ה 3/4 הפתרון האופטימלי נמצא , לכן הוספנו דור נוסף ליתר ביטחון – במקרה הגרוע ביותר נשתמש בו . (יכולנו גם להוסיף 100 דורות אך במקרה שלנו זה לא באמת משנה תמיד זה יעצור באותו טווח וזה יהיה בזבוז משאבים).

כל גן אצלנו (Gen) מורכב מ X ו Y ערכים אלו בעצם מדמים לחיצות על תאים במשחק.

X – השורה שנבחרה.
Y – העמודה שנבחרה.

כל פתרון (Solution) מורכב מכמה גנים, לדוגמא : [[Gen],[Gen]....].

כל אוכלוסייה (Population) מורכבת מכמה פתרונות , לדוגמא: [[Solution] , [Solution] , [Solution]..].

כל דור (Generation) זה אוכלוסייה חדשה.

פסאודו קוד של אלגוריתם גנטי :

Algorithm: $\mathrm{GA}(n, \chi, \mu)$

// Initialise generation 0:
$k := 0;$
$P_k :=$ a population of n randomly-generated individuals;
// Evaluate P_k:
Compute *fitness*(i) for each $i \in P_k$;
do
{ **// Create generation $k+1$:**
// 1. Copy:
Select $(1-\chi) \times n$ members of P_k and insert into P_{k+1};
// 2. Crossover:
Select $\chi \times n$ members of P_k; pair them up; produce offspring; insert the offspring into P_{k+1};
// 3. Mutate:
Select $\mu \times n$ members of P_{k+1}; invert a randomly-selected bit in each;
// Evaluate P_{k+1}:
Compute *fitness*(i) for each $i \in P_k$;
// Increment:
$k := k+1;$
}
while fitness of fittest individual in P_k is not high enough;
return the fittest individual from P_k;

1. אתחול האוכלוסייה הראשונית

```
initial_population = generate_population(ms_board, BEGINNER_POPULATION, all_uncovered_neighbors)
```

2. לכל דור מסך כל הדורות שהחלטנו שיהיו :
   2.1 . בחירת כמות ההורים המינימלית מבין כמות האוכלוסייה חלקי שתיים לכמות ההורים שהוגדרה מראש (4).
   2.2 . בחירות האוכלוסייה שהביאה לפתרון עם הערך המקסימלי (calc_fitness).
   2.3 . בחירת הורים.
   2.4 . יצירת ילדים חדשים על ידי שילוב בין ההורים (crossover), ועליהם עשינו מוטציות (mutation).
   2.5. בניית אוכלוסייה חדשה ע"י חיבור ההורים והילדים החדשים.
   2.6 . מחיקת גנים כפולים (לחיצות שיכתבו פעמיים באותו כרומוזום [x,y]).
   2.7. בדיקה האם הכרומוזום החדש קטן בגודלו מהכרומוזום הישן:
      2.7.1. במידה וכן הוא ישמר ככרומוזום האופטימלי.
      2.7.2. במידה ולא הכרומוזום האופטימלי האחרון ישאר.
3. הדפסת הכרומוזום האופטימלי וגודלו.

Local Search

חיפוש מקומי הוא שיטה היוריסטית לפתרון בעיות אופטימיזציה קשות מבחינה חישובית. ניתן להשתמש בחיפוש מקומי על בעיות שניתן לנסח כמציאת פתרון תוך מקסום קריטריון בין מספר פתרונות מועמדים. אלגוריתמי חיפוש מקומיים עוברים מפתרון לפתרון במרחב של פתרונות מועמדים (מרחב החיפוש) על ידי החלת שינויים מקומיים, עד שנמצא פתרון הנחשב לאופטימלי או שחלף פרק זמן שהוגדר מראש.

לחיפוש מקומי ישנם כמה סוגים , בפרויקט שלנו ממשנו את השיטה הזאת באמצעות סוג של טיפוס גבעות (Hill Climbing).

אלגוריתם טיפוס גבעות הוא אלגוריתם אופטימיזציה פשוט מאוד.

זה כרוך ביצירת פתרון והערכתו, זוהי נקודת ההתחלה אשר לאחר מכן משתפרת בהדרגה עד שלא ניתן להשיג שיפור נוסף או שנגמר לנו הזמן או המשאבים.

פתרונות מועמדים חדשים נוצרים מהפתרון המועמד הקיים . בדרך כלל, זה כרוך בביצוע שינוי יחיד בפתרון המועמד ($mutation$), הערכתו ובמידה והוא טוב יותר מהפתרון ה"טוב ביותר" הנוכחי הוא יישמר במקומו, אחרת הוא מושלך וכך הלאה עד שלא ניתן למצוא שיפורים נוספים.

שיטת חיפוש מקומי בפתרון "שולה המוקשים":

תחילה בדומה לשיטה הקודמת אנחנו נותנים ל Agent שלנו לשחק במשחק ויוצרים אוכלוסייה של פתרונות ,מתבצע בקובץ generate_population.py .
לאחר מכן, במקום ליצור פתרון אקראי חדש לחלוטין, אנו מנסים להציג שינוי מינורי בכל אחת מהאוכלוסיות, בפרויקט שלנו השינוי בוצע כך : לוקחים כל אוכלוסייה וחותכים חלק ממנה, האחוזים שנשמור מתוכם הם: 75% ,50% או 25% (ניתן לשנות זאת בקובץ config.py) ,לאחר מכן מתחילים לפתור את המשחק לפי הצעדים שנשארו ובמידה ועדיין לא הגענו לניצחון מוסיפים לפתרון צעדים חדשים אשר ידועים שמהם לא מפסידים במשחק ואותם מכניסים כחלק מהצעדים שנוספו עד לקבלת פתרון, אם השינוי הקטן הזה מניב פתרון טוב יותר (כמות צעדים קטנה יותר) נשמור אותו, אחרת נשאר עם הפתרון הקודם.

אנו יכולים לדמיין אדם שמנסה לטפס על ההר הגבוה ביותר בערפל. אנחנו לא רואים היכן נמצא ההר הגבוה ביותר, אבל אנחנו יכולים לדעת לאיזה כיוון עלינו ללכת כדי לטפס על הגבעה הקרובה ביותר.

פסאודו קוד חיפוש מקומי:

```
procedure Iterated-Local-Search(n_max : IN) : S;
    begin
        Create starting solution s;
        s := Local-Search(s);
        n := 0;
        repeat
            s' := Mutate(s);
            s' := Local-Search(s');
            if f(s') < f(s) then s := s';
            n := n + 1;
        until n = n_max;
        return s;
    end;
```

פסאודו קוד טיפוס גבעות :

```
i = initial solution
While f(s) ≤ f (i) s € Neighbours (i) do
        Generates an s € Neighbours (i);
        If fitness (s) > fitness (i) then
            Replace s with the i;
        End If
```

1 אתחול האוכלוסייה הראשונית (אותו אתחול של האלגוריתם הגנטי - יצרנו העתק)

```
initial_population = generate_population(ms_board, BEGINNER_POPULATION, all_uncovered_neighbors)
```

2 נרוץ על הפתרונות (כרומוזומים) מהאוכלוסייה הראשונית ועבור כל אחד נבצע:

2.1 נשמור את הפתרון בתור ה"טוב ביותר" ונשמור גם את מספר הצעדים (גנים) בו.

2.2 עבור הפתרון נבצע 3 ריצות כאשר בכל ריצה האחוז שאנו שומרים מהפתרון בתהליך המוטציה קטן (0.75, 0.5, 0.25) .

2.2.1 ננסה לפתור את המשחק ע" ביצוע הצעדים שבפתרון החתוך ,אם פתרנו את הלוח נמשיך לאיטרציה הבאה. אחרת, נחפש צעד חדש ונוסיף אותו לפתרון החתוך - נבצע זאת עד שנפתור את הלוח.

2.2.2 נשווה את הפתרון החדש ל"טוב ביותר", אם היה שיפור נעדכן את ה"טוב ביותר" ואת מספר הצעדים בו, במידה ולא היה שיפור נעבור לאיטרציה הבאה ישר.

2.3 בסיום האוכלוסייה מכילה את אותה כמות פתרונות אך כל אחד הוא הטוב ביותר מבין 3 הריצות שביצענו עליו.

3 הפתרון האופטימלי הוא זה שפתר את הלוח עם כמות הצעדים המינימלית.

Advantages and Disadvantage

אלגוריתם גנטי-

יתרונות:

- קל להבין את הרעיון.
- החיפוש מתבצע על כמה פתרונות ולא על פתרון יחיד.
- עוזר בפתרון בעיות אופטימיזציה.
- עובד על ייצוגים שונים של פרמטרים.
- קל לגלות את הפתרון האופטימלי.
- עמיד מאוד בפניי קשיים בהערכת הפונקציה האובייקטיבית.

חסרונות:

- קושי בבחירת הפרמטרים, מי הפתרון איך נעשית מוטציה והצלבה.
- בעיית זיהוי פונקציית הכושר.
- בחירת גודל אוכלוסייה מתאים.
- נוטה התכנסות מהירה למקסימום לוקאלי.
- לא טוב בזיהוי אופטימום מקומי.
- יקר מבחינה חישובית.

חיפוש מקומי-

יתרונות:

- ניתן להשתמש בו באופן רציף.
- יעיל בפתרון בעיות אופטימיזציה.
- זיהוי פתרון אופטימלי שנמצא עד כה.
- בחינת מספר עצום של פתרונות אפשריים בזמן חישוב קצר.
- גמישים - מותאמים בקלות למגוון רחב של בעיות.

חסרונות:

- מציאת פתרון אופטימלי מקומי (local optimum) נתקע אם אין צעד מקומי יותר טוב.
- הגבלת זמן ליצירת פתרון.
- שיטה לא שלמה.
- הפתרון האופטימלי שנקבל הוא מקומי ולא בהכרח נקבל את הפתרון האופטימלי הכללי.

System Overview

הרצת המערכת מתבצעת דרך PyCharm עם Python 3.9.
השתמשנו בתוכנת Git על מנת לשתף ולמזג שינויים בקוד בזמן אמת.

הקודים מהם בנויה המערכת הם קודים כלליים של השיטות ובנוסף מצאנו ב Github את קוד המשחק ממומש עם אלגוריתם גנטי, לקחנו את הקוד הוספנו הדפסות ופונקציונליות נוספת לקוד ופירקנו אותו עשינו לו שינויים ושיפוצים על מנת שיכיל את שני האלגוריתמים ויציג את הפלטים הרצויים, גם הוספנו מחולל לוחות משחק אקראיים ושליפת נתונים נוספים לצורך השוואה מעמיקה.

דרישות המערכת:

על מנת להפעיל את המערכת נדרש-

תוכנה אשר בה ניתן להריץ קוד בשפת Python גרסה 3.9.

והספריות הבאות-

+ **scipy** - היא ספריית קוד פתוח המשמשת לפתרון בעיות מתמטיות, מדעיות, הנדסיות וטכניות. זה מאפשר למשתמשים לתפעל את הנתונים ולהמחיש את הנתונים באמצעות מגוון רחב של פקודות (דורשת התקנה על מנת לייבא).

+ **time** - השתמשנו על מנת למדוד את זמן הריצה של כל אלגוריתם.
+ **copy** - מאפשרת לנו לעשות העתקה עמוקה (השתמשנו עבור האוכלוסייה ועבור הדפסת לוח המשחק.(
+ **random** - מחוללת ערכים אקראיים (השתמשנו ביצירת לוחות משחק חדשים).

Program Execution

לחיצה ראשונית על לחצן ה Play מתחילה את ריצה התוכנית .

תחילה לוח המשחק נבנה באופן רנדומלי על גודל לוח קבוע 8X8 ומספר קבוע של פצצות - 10.
הלוח נבנה דרך מחולל לוחות שבונה אותו לפי הערכים האלה ומחזיר לוח רנדומלי פתיר.

לאחר שהלוח בנוי ומוכן ה Agent פותר אותו כמה פעמים ומהפתרונות מייצר אוכלוסייה בגודל 10 (את גודל האוכלוסייה ניתן לשנות בשדה BEGINNER_POPULATION בקובץ config.py).

בקונסול הם מוצגים בצורה הבאה:

```
chromosome- 5  =  [[6, 7], [4, 0], [0, 6], [0, 0], [5, 5], [6, 4], [2, 6], [5, 3], [4, 6], [3, 2], [3, 1], [1, 1], [7, 5], [1, 2], [5, 6]]
num of genes in chromosome:  15
```

את האוכלוסייה הראשונית אנחנו שולחים לשתי השיטות.

בשיטה הראשונה האלגוריתם הגנטי מקבל את האוכלוסייה וממנה מתחיל לבנות את הדורות הבאים לפי בחירת הורים וביצוע פעולות על הגנים שלהם כפי שהוזכר בפרק Genetic Algorithm ויצירת תינוקות חדשים.
לבסוף יתקבל הפתרון האופטימלי שיכיל את המספר הקטן ביותר של צעדים לפתרון.

בשיטה השנייה אלגוריתם החיפוש המקומי מקבל את אותה אוכלוסייה ראשונית שה Agent מצא, ולאחר מכן על אותה אוכלוסייה מתבצעים פעולות כפי שהוזכרו בפרק Local Search , עושים חיתוך של כרומוזום עם הוספת צעדים הכרחיים לפתרון, לכל אחד מ 10 הפתרונות באוכלוסייה הראשונית מריצים שלושה משחקים ולבסוף התוצאה הטובה ביותר נשמרת מבין כל שלושים המשחקים כפתרון האופטימלי, כמות המשחקים יכולה להשתנות במידה ומשנים את גודל האוכלוסייה וכמות הפעמים שנרצה לחתוך אחוז מסוים מכל כרומוזום (ניתן לשנות את ערכי האחוזים וכמותם בשדה LOCAL_SEARCH_KEPT_PERCENT בקובץ config.py).

לאחר שהתוכנית מסיימת את ריצתה, מודפסת בקונסול כמות הצעדים המינימלית שכל שיטה עשתה על מנת לפתור את אותו לוח, הפלט ייראה כך:

```
optimal GA = 15
optimal LS = 6
```

optimal GA - מייצג את כמות הצעדים האופטימלית (מינימלית) שהאלגוריתם הגנטי מצא.

optimal LS - מייצג את כמות הצעדים האופטימלית (מינימלית) שאלגוריתם החיפוש מקומי מצא.

<u>הערה</u> - כדי להחזיר את ההדפסה של הלוח בכל שלב באלגוריתם הגנטי צריך לעשות uncomment לשורה 56 בקובץ main.

Comparison between the methods

השוואה בין האלגוריתמים:

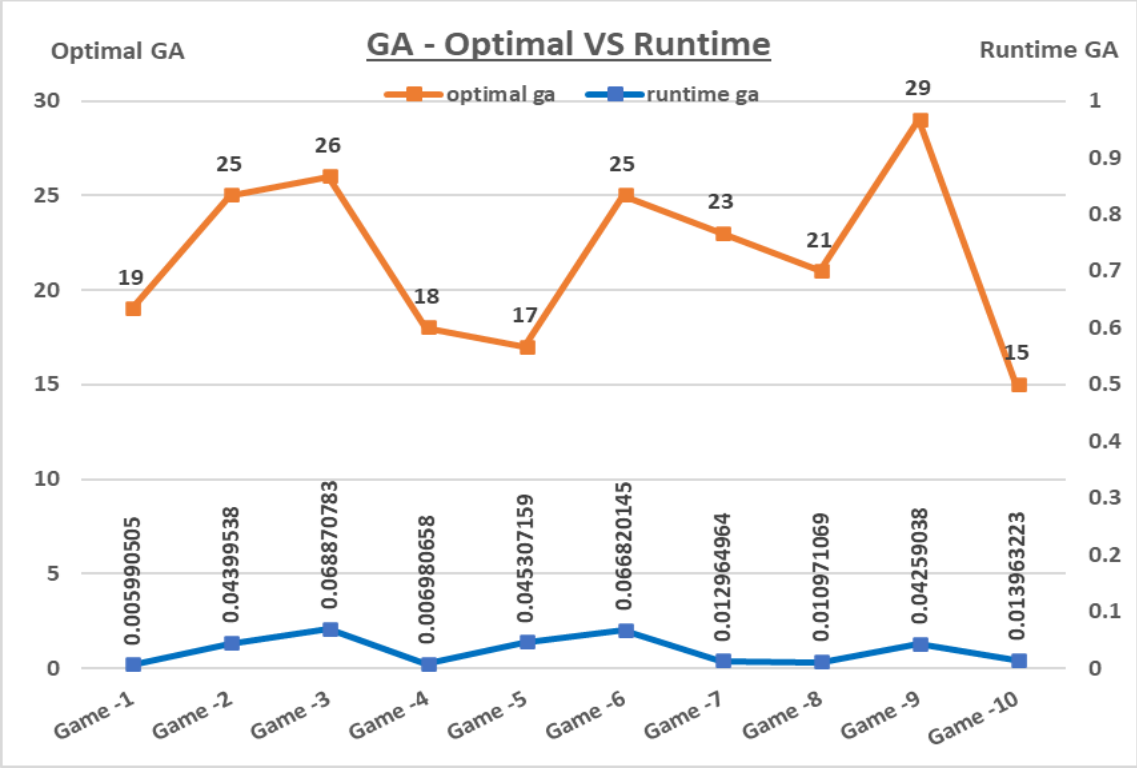
קושי למתכנת - לאורך כתיבת הקוד ראינו שהאלגוריתם הגנטי הרבה יותר קשה ומורכב להבנה ויישום מאשר אלגוריתם החיפוש המקומי, למרות שהאלגוריתם הגנטי מתאר תהליך אבולוציה מובר היה לנו קשה לתרגם את הבעיה לפורמט הזה (כרומוזומים, גנים, מוטציה וכו') לעומת זאת אלגוריתם החיפוש המקומי הוא גם פשוט להבנה וגם יחסית קל ליישום כל עוד בוחרים קריטריון מתאים עבור המוטציה.

אוסף הפתרונות - בשני האלגוריתמים התחלנו מאותה אוכלוסייה, כל אלגוריתם עשה עליה את השינויים שלו.
(הגנטי בכל דור יצר צאצאים ועליהם הפעיל מוטציה, והחיפוש המקומי חתך את הפתרון לפי קריטריון המוטציה והוסיף צעדים חדשים לפתרון בצורה רנדומלית).
באלגוריתם הגנטי עם התקדמות הדורות אנו מקטינים את האוכלוסייה עד להתכנסות לפתרון אופטימלי יחיד לעומת זאת בחיפוש המקומי אני מקבלים חזרה את אותה כמות פתרונות כאשר כל פתרון הוא אופטימלי עבור כל פתרון באוכלוסייה הראשונית.
בנוסף, בחיפוש המקומי אנו יכולות בכל נקודת זמן לדעת מי הוא הפתרון האופטימלי ובאלגוריתם הגנטי רק בסוף כל הדורות יוחזר לי האופטימלי וזה מאופן בחירת ההורים (לעיתים בחירת הורים פחות טובים יכולה להוביל לפתרון טוב יותר).

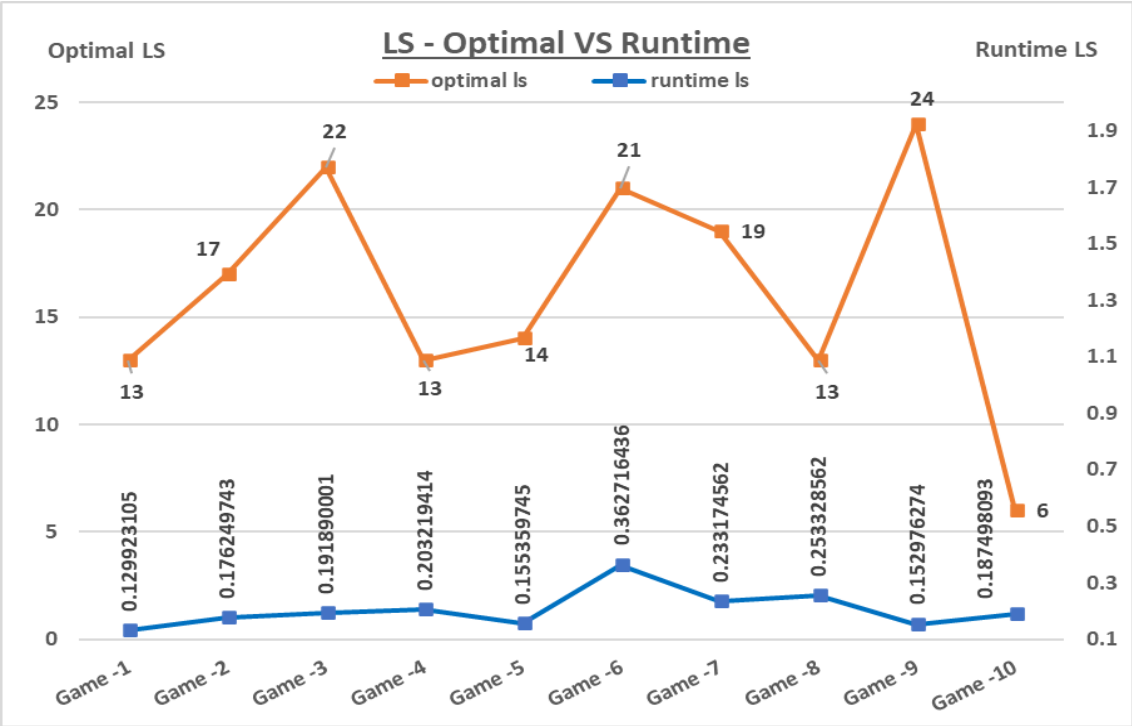
אחוזי הצלחה - מכיוון שהבעיה שלנו היא לא באמת פתירת המשחק אלא מציאת פתרון בעל מספר צעדים מינימלי, אנו מודדות את אחוזי ההצלחה לפי "בכמה אחוזים מתוך סך המשחקים שבוצעו אלגוריתם אחד פתר בפחות **צעדים** מאשר השני" ובנוסף "בכמה אחוזים מתוך סך המשחקים שבוצעו אלגוריתם אחד פתר בפחות **זמן** מאשר השני".

ביצענו 10 הרצות רצופות (עם 10 לוחות שונים) של הקוד תוך הפעלת שני האלגוריתמים על כל לוח, מצאנו שבאופן גורף זמן הריצה הקצר יותר היה של האלגוריתם הגנטי ולעומת זאת באופן גורף מספר הצעדים המינימלי היה אצל החיפוש המקומי.

גרף הממחיש את זמני הריצה עבור 10 הרצות של האלגוריתם הגנטי:



גרף הממחיש את זמני הריצה עבור 10 הרצות של החיפוש המקומי:



זמן ומקום -

אלגוריתם החיפוש המקומי הוא לא שלם ולא אופטימלי, בעל סיבוכיות זמן של $O(\infty)$, סיבוכיות מקום של $O(b)$. האלגוריתם הגנטי בעל סיבוכיות זמן ומקום של $O(NP * NC)$, כאשר NP מייצג את גודל האוכלוסייה ו NC מייצג את גודל הכרומוזום,
בפרויקט שלנו מכיוון שהאלגוריתם מלכתחילה מקבל פתרונות אפשריים ורק מבצע עליהם אופטימיזציה ניתן לומר שהוא שלם, באופן כללי אינו מניב פתרון אופטימלי (כפי שגם ראינו בפרויקט ונדון בהמשך).

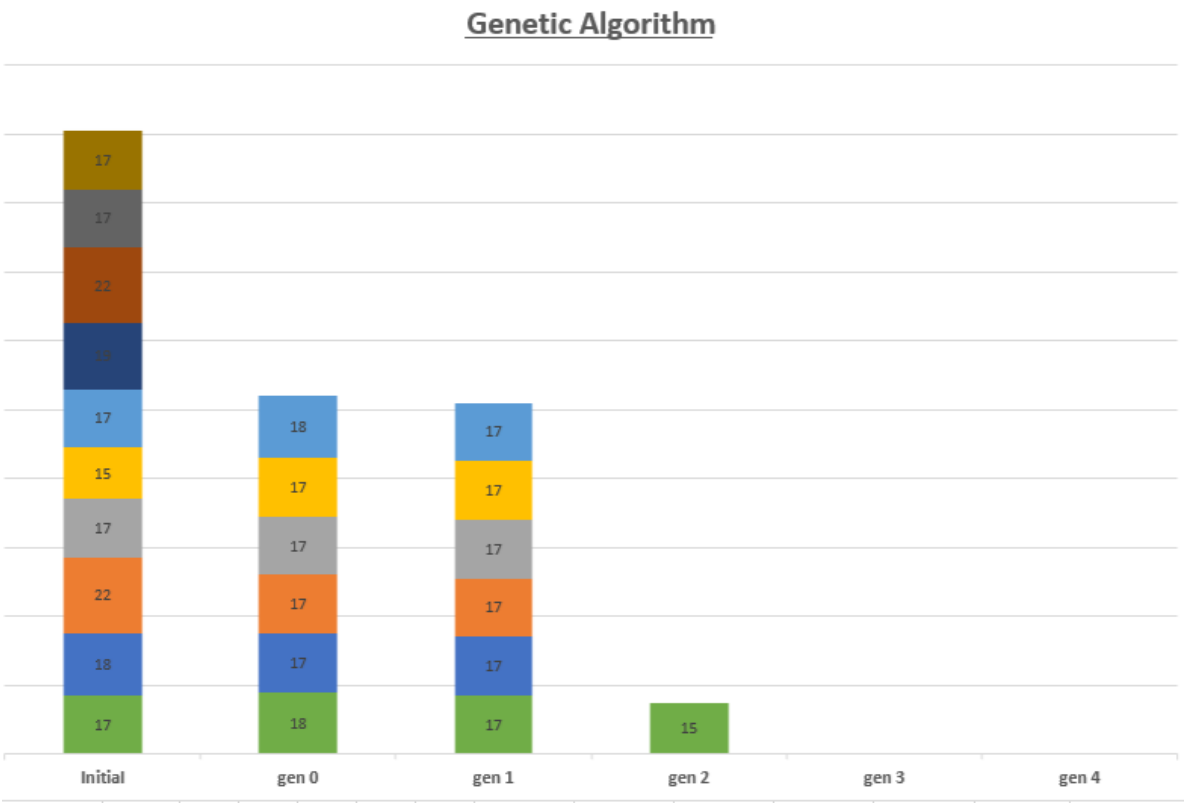
מסקנות מהתהליך -

כשהתחלנו את הפרויקט הקוד בו נעזרנו היה בנוי מאלגוריתם גנטי שנועד כדי לשפר את האוכלוסייה הראשונית שבהמשך משמשת כפלט עבור אלגוריתם החיפוש המקומי ועליה מבצע אופטימיזציה נוספת, לכן חשבנו שבלי האלגוריתם הגנטי כבסיס התוצאות של החיפוש המקומי יהיו פחות טובות.
היינו סקרניות לראות מה יהיו ההבדלים בין הפלט של האלגוריתם הגנטי לבד לעומת החיפוש המקומי לבד וזו הסיבה שבחרנו דווקא בהם.
רק מאוחר יותר בתוך הפרויקט כשכבר הספקנו לחקור יותר לעומק, התברר לנו שאלגוריתמים גנטיים הם טכניקה של חיפוש מקומי וכיום הם נחשבים על פי רוב למוצלחים פחות מאלגוריתמים המבוססים על טיפוס בהר.

כפי שתיארנו למעלה (בחלק ההשוואה) גם בפרוייקט שלנו ראינו שאכן התוצאות מהחיפוש המקומי היו טובות יותר מאשר התוצאות מהאלגוריתם הגנטי מבחינת כמות הצעדים אך באופן מפתיע האלגוריתם הגנטי התעלה מבחינת זמן הריצה שחישבנו.

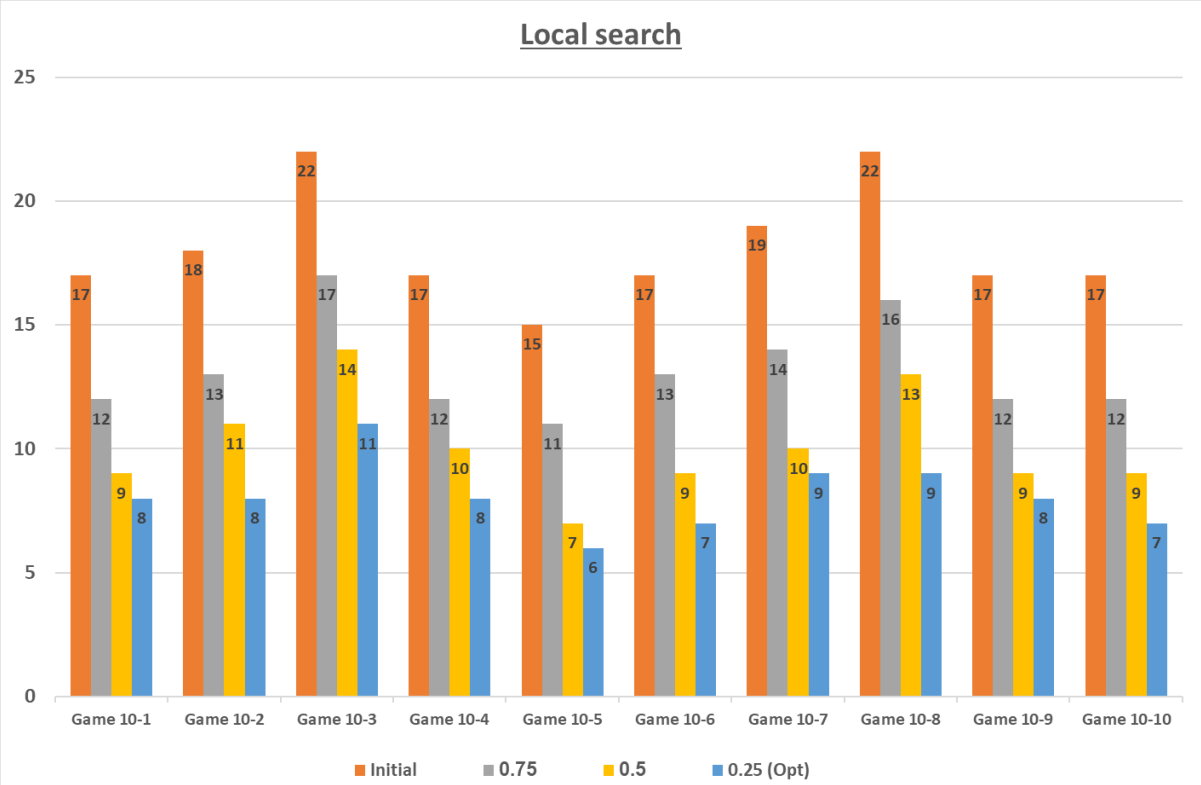
בנוסף מצורפים הגרפים עבור ההרצה ה 10 של המשחק בשני האלגוריתמים:

<u>האלגוריתם הגנטי</u> -



בצד שמאל יש את אורכי הכרומוזומים באוכלוסיה הראשונית שהתקבלה, וניתן לראות שכאשר מתקדמים בדורות אנו מתכנסים לתוצאה אפילו לפני שהסתיימו הדורות. בריצות אחרות אפילו ראינו שהוא נתקע על ערך מסוים ולא מצליח להשתנות מספיק כדי למצוא פתרון יותר אופטימלי ולכן בדור האחרון הוא בוחר את המינימלי מבין הערכים שמצא למרות שהוא לא בהכרח האופטימלי.

אלגוריתם החיפוש המקומי-



בגרף זה ניתן לראות את התוצאות עבור כל אחת מ 3 הריצות על כל פתרון מהאוכלוסייה הראשונית (כתום זה האורך של הפתרון הראשוני עליו עובדים כרגע) ולידו יש את האפור שמייצג את הריצה בה שומרים 0.75 מהפתרון ושאר הצעדים הם רנדומליים, הצהוב מייצג את הריצה בה שומרים 0.5 מהפתרון והכחול מייצג את הריצה בה שומרים 0.25 מהפתרון.

עד שהכנסנו את הנתונים לגרף לא ראינו זאת אך ניתן לשים לב שבאופן גורף שמירה של 0.25 מהפתרון (עמודה בכחול) מניבה את הפתרון האופטימלי בכל אחד מ 10 הפתרונות.
כשראינו זאת ניסינו לבצע מספר הרצות נוספות עם מספרים בסביבה של 0.25 ומטה אך לא ראינו שיפור נוסף.

תובנות בתחום הבינה המלאכותית -

ראינו שתחום הבינה המלאכותית הוא תחום מאוד עשיר ומעניין , תמיד מתפתח וגדל , ככל שחקרנו יותר על שיטות לפתרון בעיה ראינו כמה יש מגוון עצום של דרכים לפתרון שיכול לדמות סוג של שוני בין צורת חשיבה בין בני האדם , ראינו כי הגדרת הבעיה והפרמטרים וכן הבנה איך להשתמש באלגוריתם הם מאוד קשים , אך כשמבינים זה מספק מאוד.
בכל שחקרנו את השיטות גילינו גם כמה ניתן לעשות שילובים בין השיטות , במיוחד ברשת נוירונים הרגשנו שהנושאים מאוד עמוקים ומורכבים מה שנתן לנו פרופורציות לכמה הקורס הזה הוא "קצה המזלג" של הנושא הזה.

בנוסף נתקלנו בבעיות ולא הצלחנו להסתדר עם ספריות לדוגמא משיטת רשת נוירונים מה שגרם לנו לקחת צעד אחורה ולחפש דרך שיותר תתאים לנו , יותר נבין ונתחבר אליה.

Bibliography

פסאודו קוד אלגוריתם גנטי:

http://www.cs.ucc.ie/~dgb/courses/tai/notes/handout12.pdf

פסאודו קוד טיפוס הרים :

https://www.researchgate.net/figure/Pseudo-code-of-the-Hill-Climbing-method_fig2_326426215

פסאודו קוד חיפוש מקומי :

https://www.researchgate.net/figure/Iterated-Local-Search-pseudo-code_fig1_221461027

אלגוריתם גנטי וחיפוש מקומי :

https://github.com/happygirlzt/minesweeper

קישור לספר על בינה מלאכותית בעברית:

https://github.com/AvrahamRaviv/Deep-Learning-in-Hebrew/blob/main/02%20-%20Machine%20Learning.pdf

הסברים על שיטות לפתרון שולה המוקשים:

https://www.mimuw.edu.pl/~erykk/algods/lecture11.html

עיצוב תמונת הפרויקט : דויד מהגרפטה

נספח - הרצת התוכנית עם צילומים

דוגמא לריצת המשחק:

זהו הלוח שנוצר לאותה ריצה - עליו שני האלגוריתמים יפעלו

```
ooooo generated board ooooo
 !  !  2  2  !  !  2  1
 2  2  3  !  5  3  2  !
 0  0  2  !  !  1  1  1
 0  0  1  2  2  2  1  1
 0  0  0  0  0  1  !  1
 0  0  0  0  0  1  1  1
 0  0  1  1  1  0  0  0
 0  0  1  !  1  0  0  0
```

<u>אלגוריתם גנטי</u> -

chromosome- 1 = [[5, 6], [2, 1], [3, 6], [6, 7], [1, 5], [4, 7], [1, 6], [2, 7], [2, 6], [0, 2], [2, 5], [0, 6], [0, 3], [3, 7], [0, 7], [1, 4], [1, 2], [3, 5]]

num of genes in chromosome: 18

chromosome- 2 = [[6, 0], [0, 3], [2, 7], [0, 6], [0, 2], [1, 5], [7, 4], [6, 7], [0, 7], [4, 7], [1, 6], [1, 4], [2, 6], [1, 2], [3, 7], [3, 6], [2, 5], [3, 5]]

num of genes in chromosome: 18

chromosome- 3 = [[5, 5], [3, 5], [1, 6], [3, 1], [5, 7], [6, 6], [3, 7], [0, 7], [2, 6], [0, 3], [1, 2], [2, 7], [3, 6], [4, 7], [0, 6], [0, 2], [1, 5], [2, 5], [1, 4]]

num of genes in chromosome: 19

chromosome- 4 = [[6, 6], [5, 4], [1, 2], [0, 6], [3, 5], [0, 2], [2, 7], [1, 4], [3, 6], [2, 6], [0, 3], [3, 7], [1, 6], [1, 5], [0, 7], [2, 5], [4, 7]]

num of genes in chromosome: 17

chromosome- 5 = [[5, 1], [1, 5], [7, 5], [3, 7], [0, 7], [0, 2], [1, 4], [2, 5], [3, 5], [3, 6], [0, 6], [1, 6], [4, 7], [0, 3], [2, 6], [1, 2], [2, 7]]

num of genes in chromosome: 17

chromosome- 6 = [[2, 0], [3, 7], [5, 7], [0, 2], [6, 6], [4, 7], [2, 5], [3, 6], [2, 7], [3, 5], [0, 3], [1, 5], [1, 2], [2, 6], [1, 4], [0, 7], [0, 6], [1, 6]]

num of genes in chromosome: 18

chromosome- 7 = [[3, 0], [7, 6], [1, 2], [4, 7], [3, 5], [1, 4], [2, 5], [0, 6], [3, 7], [0, 2], [2, 7], [3, 6], [0, 7], [1, 5], [1, 6], [2, 6], [0, 3]]

num of genes in chromosome: 17

chromosome- 8 = [[2, 0], [5, 7], [1, 5], [3, 7], [1, 4], [2, 5], [1, 6], [0, 6], [3, 6], [6, 5], [2, 6], [2, 7], [0, 7], [0, 2], [1, 2], [4, 7], [3, 5], [0, 3]]

num of genes in chromosome: 18

chromosome- 9 = [[2, 0], [1, 5], [6, 5], [1, 2], [3, 6], [2, 6], [3, 5], [2, 7], [0, 2], [1, 4], [4, 7], [0, 6], [3, 7], [2, 5], [0, 7], [1, 6], [0, 3]]

num of genes in chromosome: 17

chromosome- 10 = [[5, 6], [2, 0], [0, 6], [2, 5], [7, 7], [1, 5], [3, 7], [0, 3], [4, 7], [1, 6], [3, 5], [0, 2], [3, 6], [2, 6], [0, 7], [1, 2], [2, 7], [1, 4]]

num of genes in chromosome: 18

generated population = 17

```
~~~~~~~~~~~~~~~~~~~~~
    Generation 0
~~~~~~~~~~~~~~~~~~~~~
Game 1 - Generation 0
•••••••• move = [5, 6]

 *  *  *  *  *  *  *  *
 *  *  *  *  *  *  *  *
 *  *  *  *  *  *  *  *
 *  *  *  *  *  *  *  *
 *  *  *  *  *  *  *  *
 *  *  *  *  *  *  1  *
 *  *  *  *  *  *  *  *
 *  *  *  *  *  *  *  *
```

```
•••••••• move = [2, 1]

 *  *  *  *  *  *  *  *
 2  2  3  *  *  *  *  *
 0  0  2  *  *  *  *  *
 0  0  1  2  2  2  *  *
 0  0  0  0  0  1  *  *
 0  0  0  0  0  1  1  1
 0  0  1  1  1  0  0  0
 0  0  1  *  1  0  0  0
```

```
•••••••• move = [3, 6]

 *  *  *  *  *  *  *  *
 2  2  3  *  *  *  *  *
 0  0  2  *  *  *  *  *
 0  0  1  2  2  2  1  *
 0  0  0  0  0  1  *  *
 0  0  0  0  0  1  1  1
 0  0  1  1  1  0  0  0
 0  0  1  *  1  0  0  0
```

```
•••••••• move = [6, 7]

 *  *  *  *  *  *  *  *
 2  2  3  *  *  *  *  *
 0  0  2  *  *  *  *  *
 0  0  1  2  2  2  1  *
 0  0  0  0  0  1  *  *
 0  0  0  0  0  1  1  1
 0  0  1  1  1  0  0  0
 0  0  1  *  1  0  0  0
```

```
•••••••• move = [1, 5]

 *  *  *  *  *  *  *  *
 2  2  3  *  *  3  *  *
 0  0  2  *  *  *  *  *
 0  0  1  2  2  2  1  *
 0  0  0  0  0  1  *  *
 0  0  0  0  0  1  1  1
 0  0  1  1  1  0  0  0
 0  0  1  *  1  0  0  0
```

```
•••••••• move = [4, 7]

 *  *  *  *  *  *  *  *
 2  2  3  *  *  3  *  *
 0  0  2  *  *  *  *  *
 0  0  1  2  2  2  1  *
 0  0  0  0  0  1  *  1
 0  0  0  0  0  1  1  1
 0  0  1  1  1  0  0  0
 0  0  1  *  1  0  0  0
```

```
•••••••• move = [1, 6]

 *  *  *  *  *  *  *  *
 2  2  3  *  *  3  2  *
 0  0  2  *  *  *  *  *
 0  0  1  2  2  2  1  *
 0  0  0  0  0  1  *  1
 0  0  0  0  0  1  1  1
 0  0  1  1  1  0  0  0
 0  0  1  *  1  0  0  0
```

```
•••••••• move = [2, 7]

 *  *  *  *  *  *  *  *
 2  2  3  *  *  3  2  *
 0  0  2  *  *  *  *  1
 0  0  1  2  2  2  1  *
 0  0  0  0  0  1  *  1
 0  0  0  0  0  1  1  1
 0  0  1  1  1  0  0  0
 0  0  1  *  1  0  0  0
```

```
•••••••• move = [2, 6]

 *  *  *  *  *  *  *  *
 2  2  3  *  *  3  2  *
 0  0  2  *  *  *  1  1
 0  0  1  2  2  2  1  *
 0  0  0  0  0  1  *  1
 0  0  0  0  0  1  1  1
 0  0  1  1  1  0  0  0
 0  0  1  *  1  0  0  0
```

```
•••••••• move = [0, 2]

 *  *  2  *  *  *  *  *
 2  2  3  *  *  3  2  *
 0  0  2  *  *  *  1  1
 0  0  1  2  2  2  1  *
 0  0  0  0  0  1  *  1
 0  0  0  0  0  1  1  1
 0  0  1  1  1  0  0  0
 0  0  1  *  1  0  0  0
```

```
•••••••• move = [2, 5]

 *  *  2  *  *  *  *  *
 2  2  3  *  *  3  2  *
 0  0  2  *  *  1  1  1
 0  0  1  2  2  2  1  *
 0  0  0  0  0  1  *  1
 0  0  0  0  0  1  1  1
 0  0  1  1  1  0  0  0
 0  0  1  *  1  0  0  0
```

```
•••••••• move = [0, 6]

 *  *  2  *  *  *  2  *
 2  2  3  *  *  3  2  *
 0  0  2  *  *  1  1  1
 0  0  1  2  2  2  1  *
 0  0  0  0  0  1  *  1
 0  0  0  0  0  1  1  1
 0  0  1  1  1  0  0  0
 0  0  1  *  1  0  0  0
```

```
•••••••• move = [0, 3]

 *  *  2  2  *  *  2  *
 2  2  3  *  *  3  2  *
 0  0  2  *  *  1  1  1
 0  0  1  2  2  2  1  *
 0  0  0  0  0  1  *  1
 0  0  0  0  0  1  1  1
 0  0  1  1  1  0  0  0
 0  0  1  *  1  0  0  0
```

```
•••••••• move = [3, 7]

 *  *  2  2  *  *  2  *
 2  2  3  *  *  3  2  *
 0  0  2  *  *  1  1  1
 0  0  1  2  2  2  1  1
 0  0  0  0  0  1  *  1
 0  0  0  0  0  1  1  1
 0  0  1  1  1  0  0  0
 0  0  1  *  1  0  0  0
```

```
•••••••• move = [0, 7]

 *  *  2  2  *  *  2  1
 2  2  3  *  *  3  2  *
 0  0  2  *  *  1  1  1
 0  0  1  2  2  2  1  1
 0  0  0  0  0  1  *  1
 0  0  0  0  0  1  1  1
 0  0  1  1  1  0  0  0
 0  0  1  *  1  0  0  0
```

```
******** move = [1, 4]

 *  *  2  2  *  *  2  1
 2  2  3  *  5  3  2  *
 0  0  2  *  *  1  1  1
 0  0  1  2  2  2  1  1
 0  0  0  0  0  1  *  1
 0  0  0  0  0  1  1  1
 0  0  1  1  1  0  0  0
 0  0  1  *  1  0  0  0
```

```
******** move = [1, 2]

 *  *  2  2  *  *  2  1
 2  2  3  *  5  3  2  *
 0  0  2  *  *  1  1  1
 0  0  1  2  2  2  1  1
 0  0  0  0  0  1  *  1
 0  0  0  0  0  1  1  1
 0  0  1  1  1  0  0  0
 0  0  1  *  1  0  0  0
```

```
******** move = [3, 5]

 *  *  2  2  *  *  2  1
 2  2  3  *  5  3  2  *
 0  0  2  *  *  1  1  1
 0  0  1  2  2  2  1  1
 0  0  0  0  0  1  *  1
 0  0  0  0  0  1  1  1
 0  0  1  1  1  0  0  0
 0  0  1  *  1  0  0  0
```

```
Game 2 - Generation 0
•••••••• move = [5, 6]

 *  *  *  *  *  *  *  *
 *  *  *  *  *  *  *  *
 *  *  *  *  *  *  *  *
 *  *  *  *  *  *  *  *
 *  *  *  *  *  *  *  *
 *  *  *  *  *  *  1  *
 *  *  *  *  *  *  *  *
 *  *  *  *  *  *  *  *
```

בסיום כל המשחקים של דור 0 כך תחושב האוכלוסייה החדשה:

** Parents **

Parents 1 = [[6, 0], [0, 3], [2, 7], [0, 6], [0, 2], [1, 5], [7, 4], [6, 7], [0, 7], [4, 7], [1, 6], [1, 4], [2, 6], [1, 2], [3, 7], [3, 6], [2, 5], [3, 5]]

Parents 2 = [[5, 1], [1, 5], [7, 5], [3, 7], [0, 7], [0, 2], [1, 4], [2, 5], [3, 5], [3, 6], [0, 6], [1, 6], [4, 7], [0, 3], [2, 6], [1, 2], [2, 7]]

Parents 3 = [[2, 0], [1, 5], [6, 5], [1, 2], [3, 6], [2, 6], [3, 5], [2, 7], [0, 2], [1, 4], [4, 7], [0, 6], [3, 7], [2, 5], [0, 7], [1, 6], [0, 3]]

** Babies **

Babies 1 = [[6, 0], [0, 3], [2, 7], [0, 6], [0, 2], [1, 5], [7, 4], [2, 5], [3, 5], [3, 6], [0, 6], [1, 6], [4, 7], [0, 3], [2, 6], [1, 2], [2, 7]]

Babies 2 = [[5, 1], [1, 5], [7, 5], [3, 7], [0, 7], [0, 2], [1, 4], [2, 5], [3, 5], [3, 6], [0, 6], [1, 6], [4, 7], [0, 3], [2, 6], [1, 2], [0, 3]]

Babies 3 = [[2, 0], [1, 5], [6, 5], [1, 2], [3, 6], [2, 6], [3, 5], [6, 7], [0, 7], [4, 7], [1, 6], [1, 4], [2, 6], [1, 2], [3, 7], [3, 6], [2, 5], [3, 5]]

** Mutant Babies **

Mutant Babies 1 = [[6, 0], [0, 3], [6, 3], [0, 6], [0, 2], [1, 5], [7, 4], [2, 5], [3, 5], [3, 6], [0, 6], [1, 6], [4, 7], [0, 3], [2, 6], [1, 2], [2, 7]]

Mutant Babies 2 = [[5, 1], [1, 5], [7, 5], [3, 7], [0, 7], [0, 2], [1, 4], [2, 5], [3, 5], [3, 6], [0, 6], [1, 4], [4, 7], [0, 3], [2, 6], [1, 2], [0, 3]]

Mutant Babies 3 = [[2, 0], [1, 5], [6, 5], [1, 2], [3, 6], [0, 4], [3, 5], [6, 7], [0, 7], [4, 7], [1, 6], [1, 4], [2, 6], [1, 2], [3, 7], [3, 6], [2, 5], [3, 5]]

** Population **

Population 1 = [[6, 0], [0, 3], [2, 7], [0, 6], [0, 2], [1, 5], [7, 4], [6, 7], [0, 7], [4, 7], [1, 6], [1, 4], [2, 6], [1, 2], [3, 7], [3, 6], [2, 5], [3, 5]]

Population 2 = [[5, 1], [1, 5], [7, 5], [3, 7], [0, 7], [0, 2], [1, 4], [2, 5], [3, 5], [3, 6], [0, 6], [1, 6], [4, 7], [0, 3], [2, 6], [1, 2], [2, 7]]

Population 3 = [[2, 0], [1, 5], [6, 5], [1, 2], [3, 6], [2, 6], [3, 5], [2, 7], [0, 2], [1, 4], [4, 7], [0, 6], [3, 7], [2, 5], [0, 7], [1, 6], [0, 3]]

Population 4 = [[6, 0], [0, 3], [6, 3], [0, 6], [0, 2], [1, 5], [7, 4], [2, 5], [3, 5], [3, 6], [0, 6], [1, 6], [4, 7], [0, 3], [2, 6], [1, 2], [2, 7]]

Population 5 = [[5, 1], [1, 5], [7, 5], [3, 7], [0, 7], [0, 2], [1, 4], [2, 5], [3, 5], [3, 6], [0, 6], [1, 4], [4, 7], [0, 3], [2, 6], [1, 2], [0, 3]]

Population 6 = [[2, 0], [1, 5], [6, 5], [1, 2], [3, 6], [0, 4], [3, 5], [6, 7], [0, 7], [4, 7], [1, 6], [1, 4], [2, 6], [1, 2], [3, 7], [3, 6], [2, 5], [3, 5]]

population shape = [18, 17, 17, 17, 17, 18]

best result is 17

ובסוף ריצת כל הדורות וכל המשחקים זו האוכלוסיה שנבחרה:

** Population **

Population 1 = [[5, 1], [1, 5], [7, 5], [3, 7], [0, 7], [0, 2], [1, 4], [2, 5], [3, 5], [3, 6], [0, 6], [1, 4], [4, 7], [0, 3], [2, 6], [1, 2], [0, 3]]

population shape = [17]

best result is 17

פלט נוסף עבור האלגוריתם הגנטי:

optimal GA = 17

Genetic RunTime: 0.200219202041626

חיפוש מקומי -

האוכלוסייה הראשונית (זהה לזו שאיתה מתחילים באלגוריתם הגנטי)

local search population [[[5, 6], [2, 1], [3, 6], [6, 7], [1, 5], [4, 7], [1, 6], [2, 7], [2, 6], [0, 2], [2, 5], [0, 6], [0, 3], [3, 7], [0, 7], [1, 4], [1, 2], [3, 5]], [[6, 0], [0, 3], [2, 7], [0, 6], [0, 2], [1, 5], [7, 4], [6, 7], [0, 7], [4, 7], [1, 6], [1, 4], [2, 6], [1, 2], [3, 7], [3, 6], [2, 5], [3, 5]], [[5, 5], [3, 5], [1, 6], [3, 1], [5, 7], [6, 6], [3, 7], [0, 7], [2, 6], [0, 3], [1, 2], [2, 7], [3, 6], [4, 7], [0, 6], [0, 2], [1, 5], [2, 5], [1, 4]], [[6, 6], [5, 4], [1, 2], [0, 6], [3, 5], [0, 2], [2, 7], [1, 4], [3, 6], [2, 6], [0, 3], [3, 7], [1, 6], [1, 5], [0, 7], [2, 5], [4, 7]], [[5, 1], [1, 5], [7, 5], [3, 7], [0, 7], [0, 2], [1, 4], [2, 5], [3, 5], [3, 6], [0, 6], [1, 6], [4, 7], [0, 3], [2, 6], [1, 2], [2, 7]], [[2, 0], [3, 7], [5, 7], [0, 2], [6, 6], [4, 7], [2, 5], [3, 6], [2, 7], [3, 5], [0, 3], [1, 5], [1, 2], [2, 6], [1, 4], [0, 7], [0, 6], [1, 6]], [[3, 0], [7, 6], [1, 2], [4, 7], [3, 5], [1, 4], [2, 5], [0, 6], [3, 7], [0, 2], [2, 7], [3, 6], [0, 7], [1, 5], [1, 6], [2, 6], [0, 3]], [[2, 0], [5, 7], [1, 5], [3, 7], [1, 4], [2, 5], [1, 6], [0, 6], [3, 6], [6, 5], [2, 6], [2, 7], [0, 7], [0, 2], [1, 2], [4, 7], [3, 5], [0, 3]], [[2, 0], [1, 5], [6, 5], [1, 2], [3, 6], [2, 6], [3, 5], [2, 7], [0, 2], [1, 4], [4, 7], [0, 6], [3, 7], [2, 5], [0, 7], [1, 6], [0, 3]], [[5, 6], [2, 0], [0, 6], [2, 5], [7, 7], [1, 5], [3, 7], [0, 3], [4, 7], [1, 6], [3, 5], [0, 2], [3, 6], [2, 6], [0, 7], [1, 2], [2, 7], [1, 4]]]

תחילת ריצה על הפתרון הראשון באוכלוסייה:

population : [[5, 6], [2, 1], [3, 6], [6, 7], [1, 5], [4, 7], [1, 6], [2, 7], [2, 6], [0, 2], [2, 5], [0, 6], [0, 3], [3, 7], [0, 7], [1, 4], [1, 2], [3, 5]]

משחק 1 על פתרון 1 עם מוטציה של 0.75 (מה שנשאר ממנה לאחר החיתוך):

Local Search Game 0.75

Click [[5, 6], [2, 1], [3, 6], [6, 7], [1, 5], [4, 7], [1, 6], [2, 7], [2, 6], [0, 2], [2, 5], [0, 6], [0, 3], [3, 7], [0, 7], [1, 4], [1, 2], [3, 5]]

clicks_return [[5, 6], [2, 1], [3, 6], [6, 7], [1, 5], [4, 7], [1, 6], [2, 7], [2, 6], [0, 2], [2, 5], [0, 6], [0, 3]]

הפתרון לאחר מוטציה בתוספת הצעדים הרנדומליים לפתרון:

ls_result: [[5, 6], [2, 1], [3, 6], [6, 7], [1, 5], [4, 7], [1, 6], [2, 7], [2, 6], [0, 2], [2, 5], [0, 6], [0, 3], [3, 7], [1, 4]]

משחק 2 על פתרון 1 עם מוטציה של 0.5 (מה שנשאר ממנה לאחר החיתוך):

Local Search Game 0.5

Click [[5, 6], [2, 1], [3, 6], [6, 7], [1, 5], [4, 7], [1, 6], [2, 7], [2, 6], [0, 2], [2, 5], [0, 6], [0, 3], [3, 7], [0, 7], [1, 4], [1, 2], [3, 5]]

clicks_return [[5, 6], [2, 1], [3, 6], [6, 7], [1, 5], [4, 7], [1, 6], [2, 7], [2, 6]]

הפתרון לאחר מוטציה בתוספת הצעדים הרנדומליים לפתרון:

ls_result: [[5, 6], [2, 1], [3, 6], [6, 7], [1, 5], [4, 7], [1, 6], [2, 7], [2, 6], [3, 7], [2, 5], [1, 4], [0, 7], [0, 6], [0, 3]]

משחק 3 על פתרון 1 עם מוטציה של 0.25 (מה שנשאר ממנה לאחר החיתוך):

Local Search Game 0.25

Click [[5, 6], [2, 1], [3, 6], [6, 7], [1, 5], [4, 7], [1, 6], [2, 7], [2, 6], [0, 2], [2, 5], [0, 6], [0, 3], [3, 7], [0, 7], [1, 4], [1, 2], [3, 5]]

clicks_return [[5, 6], [2, 1], [3, 6], [6, 7]]

הפתרון לאחר מוטציה בתוספת הצעדים הרנדומליים לפתרון:

ls_result: [[5, 6], [2, 1], [3, 6], [6, 7], [4, 7], [3, 7], [2, 7], [2, 6], [2, 5], [1, 6], [1, 5], [1, 4], [0, 7], [0, 6], [0, 3]]

מעבר לאוכלוסייה הבאה וחזרה על התהליך.

האוכלוסיה בסיום כל הריצות:

local_search_population=[[[5, 6], [2, 1], [3, 6], [6, 7], [1, 5], [4, 7], [1, 6], [2, 7], [2, 6], [0, 2], [2, 5], [0, 6], [0, 3], [3, 7], [1, 4]], [[6, 0], [0, 3], [2, 7], [0, 6], [0, 2], [1, 5], [7, 4], [6, 7], [0, 7], [4, 7], [1, 6], [1, 4], [2, 6], [3, 7]], [[5, 5], [3, 5], [1, 6], [3, 1], [5, 7], [6, 6], [3, 7], [0, 7], [2, 6], [0, 3], [1, 2], [2, 7], [3, 6], [4, 7], [2, 5], [1, 5]], [[6, 6], [5, 4], [1, 2], [0, 6], [3, 5], [0, 2], [2, 7], [1, 4], [3, 6], [2, 6], [0, 3], [3, 7], [4, 7], [2, 5]], [[5, 1], [1, 5], [7, 5], [3, 7], [0, 7], [0, 2], [1, 4], [2, 5], [3, 5], [3, 6], [0, 6], [1, 6], [4, 7], [2, 7]], [[2, 0], [3, 7], [5, 7], [0, 2], [6, 6], [4, 7], [2, 5], [3, 6], [2, 7], [3, 5], [0, 3], [1, 5], [1, 2], [2, 6]], [[3, 0], [7, 6], [1, 2], [4, 7], [3, 5], [1, 4], [2, 5], [0, 6], [3, 7], [0, 2], [2, 7], [3, 6], [2, 6], [1, 6]], [[2, 0], [5, 7], [1, 5], [3, 7], [1, 4], [2, 5], [1, 6], [0, 6], [3, 6], [6, 5], [2, 6], [2, 7], [0, 7], [4, 7]], [[2, 0], [1, 5], [6, 5], [1, 2], [3, 6], [2, 6], [3, 5], [2, 7], [0, 2], [1, 4], [4, 7], [0, 6], [3, 7], [2, 5]], [[5, 6], [2, 0], [0, 6], [2, 5], [7, 7], [1, 5], [3, 7], [0, 3], [4, 7], [1, 6], [3, 5], [0, 2], [3, 6], [2, 7], [2, 6]]]

גדלים של הפתרונות באוכלוסיה המעודכנת:

local_search_population shape = [15, 14, 16, 14, 14, 14, 14, 14, 14, 15]

פלט נוסף בסיום החיפוש המקומי:

optimal LS = 14

LocalSearch RunTime: 0.13945794105529785