



דו"ח כלי פיתוח מתקדם

SUPERPY

ROEE GROISER-318473477

LIRON ABRAHAM-319041570

תוכן

2	הגדרת המשימה
3	דרכי פעולה שונות לפתרון המשימה
4	הוראות התקנה
5	היסטוריה של שימוש בכלי פיתוח
6	שימוש בכלי
8	תוצאות שימוש בכלי פיתוח
11	מסקנות

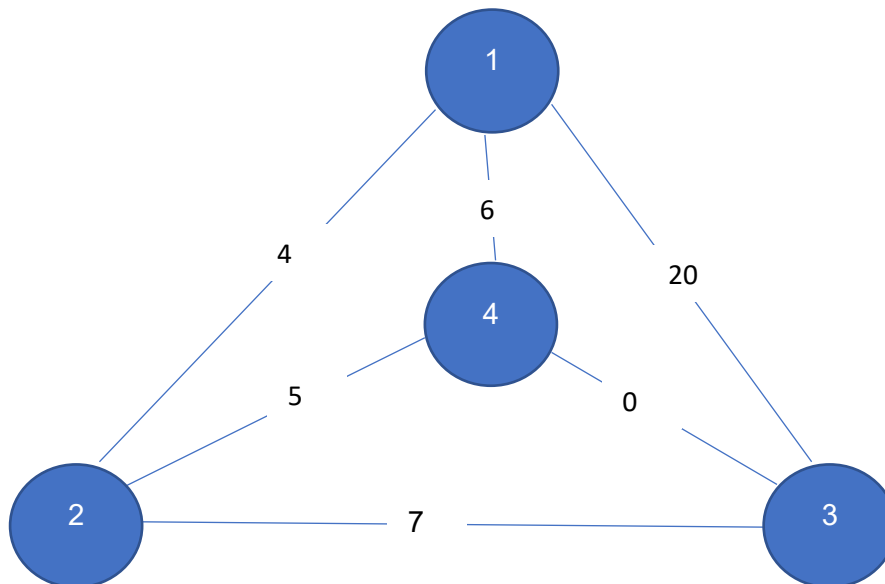
הגדרת המשימה

תארו לכם את הסיטואציה הבאה: עליכם לערוך קניות בסופרמרקט והינכם רוצים לחזור הביתה בהקדם האפשרי. לצורך כך, פיתחנו עבורכם ממשק משתמש ייעודי המציג מפה של הסופר אותו תבחרו, ותוכלו להזין את רשימת הקניות שלכם. לאחר הזנת המוצרים, יוצג מסלול יעיל באמצעות אנימציה העובר בין כל המוצרים שהוזנו לאפליקציה שקיימים בסופר.

דרכי פעולה שונות לפתרון המשימה

כאשר חפשנו ברשת על פתרונות ועבודות דומות שנעשו בעבר, נתקלנו בחברה aisle411 שמציעה אפליקציה לניווט בכמה סופרים ספציפיים בחו"ל. אך לפתרון הבעיה הם השתמשו באלגוריתם "המסלול הקצר ביותר בין 2 נקודות" ואנו סבורים שהאלגוריתם שלנו יעיל יותר מבחינת אופן בחירת המסלולים.

לדוגמא:



אנו רוצים להתחיל מנקודה מספר 1 ולסיים בה לאחר שעברנו בכל נקודה בדיוק פעם אחת. האלגוריתם שבו משתמשת aisle411 יבחר את המסלול 1-2-4-3-1 שמשקלו 29. לעומת זאת, האלגוריתם שלנו יעבור במסלול 1-4-3-2-1 שמשקלו 17.

הוראות התקנה

השתמשנו ב-2 ספריות :

mlrose בשביל לפתור את בעיית הסוכן הנוסע.

Tkinter בשביל העיצוב הגרפי.

הוראות התקנה:

Pip install mlrose

Pip install tkinter

הרצת הפרויקט תתבצע ע"י הפקודה:

Python view_super.py בתיקיית הפרויקט

היסטוריה של שימוש בכלי פיתוח

Mlrose

דוגמא לשימוש שמצאנו בכלי היא לפיתרון בעיית 8 המלכות, מכיוון ש mlrose היא ספרייה לאופטימיזציה של אלגוריתמים רנדומליים.

Tkinter

קיימות דוגמאות רבות ברשת לאופן שבו משתמשים בספרייה על מנת לעצב אלמנטים גרפיים (למשל הוספת כפתורים, צביעה שלהם, מיקום grids במסך ועוד).

שימוש בכלי

mlrose

השתמשנו בכלי זה על מנת לפתור את בעיית הסוכן הנוסע:

בעיית הסוכן הנוסע (באנגלית: Travelling Salesman Problem ובראשי תיבות TSP)

היא בעיה ידועה בתורת הגרפים ובתורת הסיבוכיות, המעלה את השאלה הבאה: "בהינתן רשימת ערים והמרחק בין כל שתי ערים, מהו המסלול הקצר ביותר, אשר יעבור בכל עיר פעם אחת, ויחזור לעיר ממנה התחיל?"

הבעיה נכללת במחלקת הסיבוכיות NP-קשיות, והיא אחת מהבעיות המרכזיות בתחום האופטימיזציה.

הכלי מציע אופטימיזציות לפתרון בעיה זו ובעיות נוספות כגון בעיית N המלכות ובעיית התרמיל.

השתמשנו בפונקציות:

mlrose.TravellingSales – בנאי להגדרת פונקציית הכושר של האלגוריתם

mlrose.TSPOpt – הגדרת אובייקט לייצוג בעיית האופטימיזציה

mlrose.genetic_alg – פתרון הבעיה ע"י שימוש באלגוריתם הגנטי

קישור ל-API המלא: <https://mlrose.readthedocs.io/en/stable/#api-reference>

Tkinter

השתמשנו בכלי זה על מנת לבנות את ממשק המשתמש.

Tkinter היא ספרייה שמאפשרת ליצור רכיבים גרפיים וליצור קשרים גרפיים ביניהם באמצעות תכנות מונחה עצמים. הרכיב המרכזי שהיא מאפשרת לייצר הוא Frame שהוא למעשה החלון שמכיל בתוכו אלמנטים נוספים (ייתכן שמכיל גם Frames נוספים בתוכו).

הרכיבים הגרפיים העיקריים שבהם השתמשנו הם Frame כאשר החלון הראשי שלנו מחולק לשני חלקים:

החלק השמאלי – מאפשר הכנסת מוצרים אותם המשתמש ירצה לקנות באמצעות אובייקט Entry, כאשר המוצרים שהוזנו בעבר מוצגים באובייקט ListBox. לממשק גם כפתור המאפשר לעדכן את הרשימה, וכפתור לשליחת הרשימה אל האלגוריתם. כמו כן יש אובייקט נוסף OptionMenu המאפשר לבחור סופר מבין כמות סופית של אפשרויות שהתוכנה מציעה.

החלק הימני – תפקידו הוא להציג את הסופר. מבחינה טכנית, הסופר מוצג באמצעות Grid – אובייקט ב Tkinter המאפשר לחלק את החלון לתאים בעלי שטח זהה. בכל תא כזה הלבשנו Button שיהיה אפשר לצבוע אותו ולהציגו בצורה ברורה למשתמש.

מבחינה ויזואלית, דאגנו שלאופן צביעת המשבצת תהיה משמעות עבור המשתמש. משבצת שצבועה בכחול-היא משבצת שמייצגת מוצר, לבן-מעבר, שחור-קיר. הצגת המסלול מתבצעת באופן דינאמי, כלומר המסלול "מתקדם" בסופר, על מנת שהמשתמש יבין באיזה סדר עליו לעבור בסופר. משבצת הצבועה בצהוב-היא משבצת המייצגת מוצר שאותו המשתמש רוצה לאסוף, ירוק-משבצת שבה המשתמש צריך לעבור.

פונקציות עיקריות שהשתמשנו בהם בספרייה Tkinter:

tk.Tk() – מחזיר Frame שהוא החלון המרכזי של התוכנית.

tk.geometry מגדירים באמצעותו את גודל החלון

tk.title הכותרת של החלון המרכזי

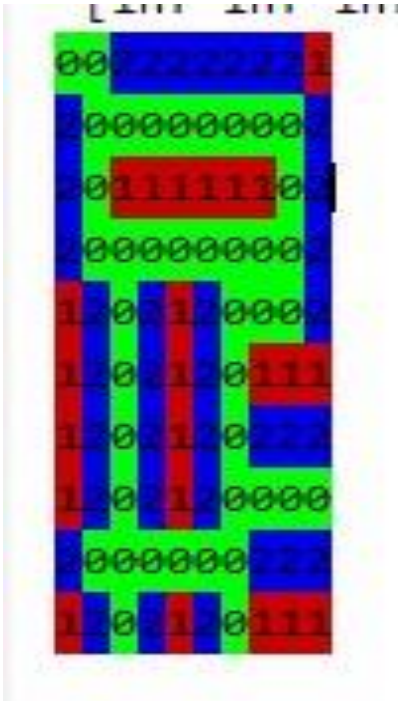
tk.pack – מארגן את האובייקטים בבלוק לפני שהם ממקומים ע"י אובייקט האב.

קישור ל tkinter API: <https://docs.python.org/3/library/tkinter.html#>

תוצאות שימוש בכלי פיתוח

Tkinter

בהתחלה הרצת התוכנה שלנו התבצעה דרך ה `cmd`, כלומר האינטראקציה בין המשתמש למערכת התבצעה באמצעות ממשק טקסטואלי, כאשר גם הפלט היה על בסיס טקסט בצבעים (מצורפת תמונה להמחשה).



לאחר השימוש בכלי, ממשיך המשתמש נראה כך

Choose a supermarket

Super 1

Type Next Product

Calculate Route

Product Name

Selected 0		Onions	Apples	Pears	Watermelons	Melons	Strawberries	Blackberries	
Potatoes									Tomatoes
Cucumbers									
Pineapples									Peppers
Bananas		Peas							Garlic
Eggs		Beans							
Donuts		Peppercorns					Frozen corn	Frozen chips	Ice cream
Chips		Red Bananas							
Selected Bread							Eggs	Hummus	Sauces
Milk			White cheese		Butter				

הבעיה שעומדת לפנינו היא NP- קשה, כך שיצירת כל הפרמוטציות האפשריות וחישוב משקלן ייקח זמן רב, לכן חיפשנו כלי מתאים לפתרון הבעיה ומצאנו את mlrose. באמצעות הכלי ושימוש באלגוריתם הגנטי שלו הצלחנו לפתור את הבעיה בזמן סביר.

אלגוריתמים גנטיים משמשים בעיקר כדי לפתור בעיות אופטימיזציה שלא ידוע עבורן פתרון דטרמיניסטי או הסתברותי העובד בזמן סביר.

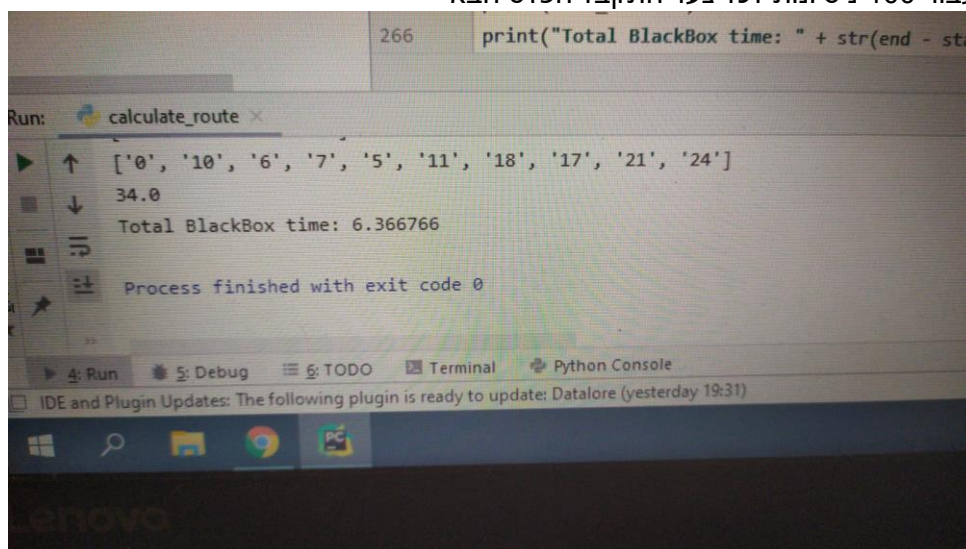
העיקרון של אלגוריתם גנטי הוא שכמו כל יצור בטבע רק המתאימים שורדים. הבעיה שעמדה כל זמן צריך להתמודד היא התאמה לסביבה והשינויים שחלים בה. שילוב אלמנטים של פתרונות אפשריים לבעיה, הפעלת הליכים של ברירה מלאכותית כדי לבחור את המועמדים שיעברו לשלבים הבאים. רעיון תכנותי בסיסי זה מושפע מההצלחה של האבולוציה בפתרון בעיות אמיתיות. לכן אם ניקח אוכלוסייה של פתרונות ונבחר מתוכם רק את המתאימים ביותר לפתרון בעיה, נערבב אותם אחד עם השני ונוסיף קצת רעש, נקבל דור חדש של פתרונות הקרוב צעד נוסף לפתרון הבעיה הנתונה. נחזור על התהליך מספר רב של פעמים (דורות) ולבסוף נגיע לפתרון הקרוב ביותר.

לשם מציאת הפרמטרים האידיאליים לבעיה שלנו הרצנו ניסיונות רבים ובחנו את זמן הריצה וטיב המסלול שהתקבל:

להלן חלק מהניסיונות וזמן הריצה הפלט מוצג באופן הבא:
המסלול הנבחר בין רשימת המוצרים
משקל המסלול
זמן הריצה (בשניות)

יש לציין שקיימות 10! פרמוטציות לרשימת המוצרים- כלומר 3628800 מסלולים אפשריים

עבור 100 ניסיונות לכל צעד התקבל הפלט הבא



```

266 print("Total BlackBox time: " + str(end - sta
Run: calculate_route x
↑ ['0', '10', '6', '7', '5', '11', '18', '17', '21', '24']
↓ 34.0
Total BlackBox time: 6.366766
Process finished with exit code 0

```

עבור 500 ניסיונות לצעד:

```
265 print(best_fitness)
266 print("Total BlackBox time: " + str(end - start))
```

Run: calculate_route

```
↑ ['0', '24', '21', '17', '18', '7', '11', '6', '5', '10']
↓ 32.0
Total BlackBox time: 41.8072693
Process finished with exit code 0
```

IDE and Plugin Updates: The following plugin is ready to update: Datalore (yesterday 19:31)

נשים לב כי משקל המסלול השתנה בכ-5 אחוזים, אך זמן הריצה גדל ביותר מ600%
לכן בחרנו במספר 200 ניסיונות לצעד, שהניב את התוצאות הבאות:

```
37 CREDBG = '\033[41m'
38 CGREENBG2 = '\033[102m'
```

Run: calculate_route

```
↑ ['0', '10', '21', '24', '17', '18', '11', '6', '7', '5']
↓ 33.0
Total BlackBox time: 15.667709799999999
Process finished with exit code 0
```

IDE and Plugin Updates: The following plugin is ready to update: Datalore (yesterday 19:31)

מסקנות

Tkinter

היינו מציעים למפתחי הכלי שיאפשרו לגרור אובייקטים בדומה לאופן השימוש ב Windows Forms ולא ע"י ייצוגם אך ורק בקוד ע"י ייצוג בפיקסלים או אחוזים.

נוסף על כך, היינו מצפים שמיקום האובייקטים בתוך ה Frame יתבצע באופן גמיש יותר, כלומר היינו רוצים למקם תמונות בתוך Grid – דבר שלא מתאפשר בספרייה הזאת, או הגדרה של סדר מיקום האובייקטים במסך מסוים במקום הסתמכות על אופן הסידור של Tkinter שהרגיש ממש שרירותי למדי.

כמו כן, בשלב מתקדם בקורס למדנו על ספרייה נוספת בפיתוח לעיצוב ממשק משתמש גרפי – kivy, שמאפשרת שליטה טובה יותר בסידור האובייקטים בתוך frame כלשהו. אם היינו מכירים את הספרייה הזו בזמן ייתכן והיינו משתמשים בה על מנת ליצור ממשק משתמש מפותח אפילו יותר.

MIrose

התרשמנו מאוד מהנוחות של ה API והפירוט שלו.

היינו מציעים אוטומציה לקביעת הפרמטרים משום שכל הרצה בודדת של האלגוריתם הגנטי לוקחת זמן רב. היינו רוצים לתת לכלי רשימה של פרמטרים עבורם היה מריץ את האלגוריתם ובתור פלט היינו מצפים לקבל ממנו רשימה של פלטים, וכך ניתן היה להשוות בין הפרמטרים השונים באופן מהיר יותר.