Traveling Salesman Problem Project

מגישות :

ענבל אלפיה
הדר ליבר

מבוא

בפרויקט זה החלטנו להתבונן בבעיית הסוכן הנוסע (Travelling salesman problem), בעיה המוגדרת כבעיה NP קשה. נרצה לבחון את פתרון בעיה זו על ידי אלגוריתמים שונים ולבצע השוואה- מי מביניהם יציג את הפתרון המדויק ביותר בזמן ריצה טוב ביותר.

<u>האלגוריתמים המוצגים בפרויקט:</u>

- אלגוריתמים מדויקים:
  - חישוב ישיר
  - תכנון דינאמי
- אלגוריתמים היוריסטיים:
  - אלגוריתם השכן הקרוב
  - אלגוריתם קירוב Christofides
- אלגוריתם אבולוציוני

<u>הגדרת הבעיה:</u>

- קלט:
  רשימת מיקומי ערים.
  בחרנו קלט עם 52 ערים - 52berlin. הקלט מיוצג בצורה של קובץ מסוג tsp. (מבנה הקובץ מוגדר [בדוקומנטציה]() של הספריה tsplib).
- פלט:
  אורך מסלול קצר ביותר אשר עובר בכל עיר פעם אחת וחוזר לעיר ממנה התחיל.

<u>ייצוג הבעיה:</u>

ניתן להציג את בעיית הסוכן הנוסע כגרף ממושקל לא מכוון כאשר כל עיר היא צומת ומרחק בין כל שתי ערים הוא קשת ממושקלת.

<u>רדוקציה:</u>

נריץ אלגוריתמים לבדיקת מהו המסלול הקל ביותר שעובר בכל צומת פעם אחת בלבד ומסיים את המסלול בנקודת ההתחלה.
בעיה זו היא בעצם חיפוש מעגל המילטון בגרף עם משקל מינימלי. (מעגל המילטון הינו מעגל המבקר בכל אחד מקודקודי הגרף בדיוק פעם אחת).

* החלטנו להתמקד בגרסה הסימטרית של הבעיה, כלומר הגרסה בה המרחק בין כל שתי ערים זהה למרחק בכיוון ההפוך בין אותן שתי ערים. במקרה כזה תוצג הבעיה כגרף לא מכוון. סימטריה זו מקטינה את מספר הפתרונות האפשריים פי שניים.

<u>פתרון אופטימלי:</u>

פתרון אופטימלי עבור הקלט הנבחר כפי שמצוין בספריית tsplib הוא: **7542**

פירוט הקבצים

1. main_tsp.py:
מכיל את הקוד הראשי שאחראי על הרצת כל האלגוריתמים בהינתן הקלט ולאחר מכן הדפסת התוצאות והגרפים בהתאמה עבור כל אלגוריתם.
2. algorithms.py:
מכיל את מימושי האלגוריתמים השונים.
3. city.py:
מכיל את המחלקה המאפיינת "עיר". עיר מיוצגת על ידי נקודה במרחב הדו מימדי, כלומר על ידי קואורדינטות x ו y. בנוסף הקובץ מכיל מחלקה המאפיינת "מרחק" כדי לחשב מרחק אוקלידי בין שתי ערים.
4. plot_tour:
קובץ זה מכיל מתודה אשר יוצרת גרף מהמסלול שהתקבל מכל אלגוריתמים מהערים בקובץ הקלט.
5. tspparse.py:
קובץ זה אחראי על פירסור קובץ הקלט והפיכתו למיליון אינפורמטיבי המכיל מידע על הערים הנתונות ומיקומן במרחב (ייצוגן כאובייקט עיר).
6. generation.py:
קובץ זה קשור לאלגוריתם האבולוציוני בלבד. מכיל מחלקה ששומרת מידע על כל דור ומבצעת עליו חישובים. מידע זה מסייע בחישובים סטטיסטיים בהמשך (לקיחת ממוצע ערכים של כל דור שנאסף מ20 ריצות).
7. graph.py:
קובץ זה קשור לאלגוריתם האבולוציוני בלבד. מכיל מחלקה ששומרת את הערכים הממוצעים שנאספו מכל דור ומכל אחת מ 20 הריצות עבור ניסוי ספציפי.
8. input.tsp:
קובץ הקלט. קובץ זה נלקח מספריית tsplib-95 ומכיל 52 ערים ומיקומן במרחב.
9. results.txt:
קובץ הפלט של האלגוריתם האבולוציוני. בקובץ זה נכתבים ערכי כל ריצה של האלגוריתם עם הפרמטרים השונים מכל אחת מבין 20 הריצות. בהמשך נשתמש בקובץ זה כדי לחשב ערכים ממוצעים עבור כל ניסוי.
10. average_results.txt:
קובץ זה מכיל את התוצאות של חישוב הממוצעים (מתבצעת קריאה מהקובץ "results.py"). חישוב הממוצעים נעשה לפי 20 הריצות המתאימות לאותה סדרת פרמטרים- גודל האוכלוסייה, מספר דורות וההסתברות לביצוע מוטציה.
11. evolution_graphs folder:
תיקייה זו קשורה לאלגוריתם האבולוציוני בלבד. תיקייה זו מכילה כל הגרפים עבור כל הניסויים שבוצעו.

פירוט האלגוריתמים

- **אלגוריתם נאיבי**

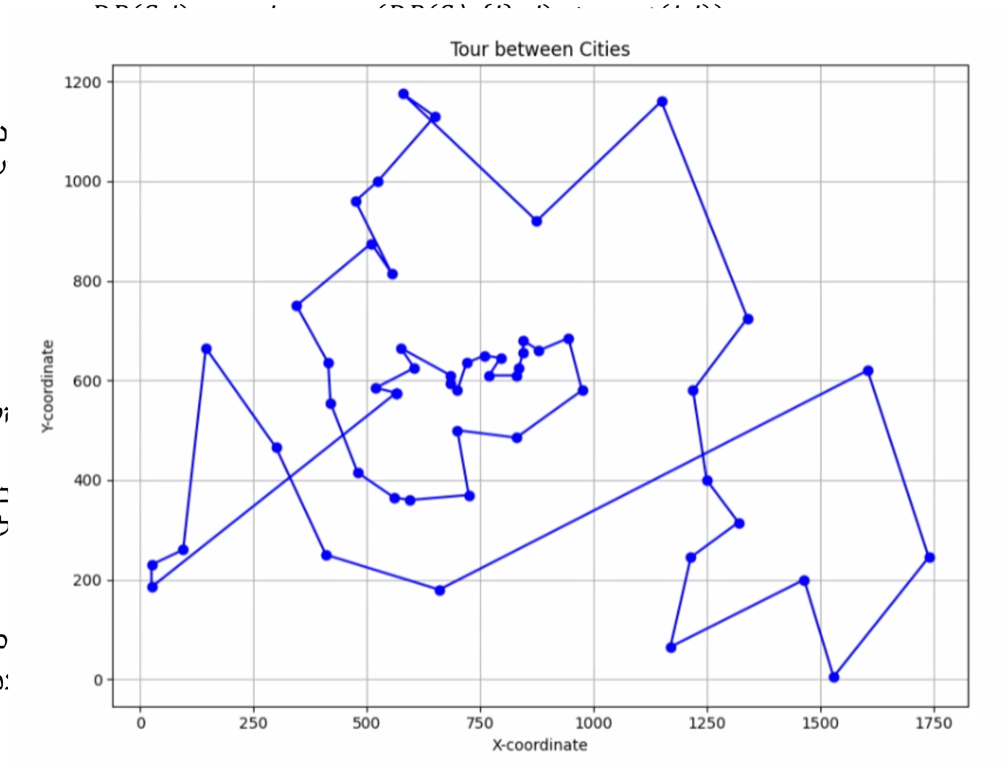
פתרון פשוט לבעיה הוא בדיקת כל התמורות האפשריות וחיפוש המסלול הקצר ביותר. פתרון זה מצריך בדיקה של $n!$ אפשרויות (עצרת של מספר הערים).
חישוב כזה הופך מהר מאוד לבלתי מעשי. דוגמה להמחשת הבעיה: אם ישנן 70 ערים בהן על הסוכן לבקר, יצטרך הסוכן לבחון !70 מסלולי נסיעה אפשריים, אשר מהווים מספר רב יותר ממספר האטומים ביקום כולו.
סיבוכיות זמן ריצה: $O(n!)$

- **תכנון דינאמי - אלגוריתם Held-Karp**

אלגוריתם Held–Karp, המכונה גם אלגוריתם Bellman–Held–Karp, הוא אלגוריתם תכנון דינאמי לפתרון בעיית הסוכן הנוסע (TSP).

פירוט האלגוריתם:

1. הגדרת הבעיה:
   נסמן ב n את מספר הערים בקלט ונגדיר S להיות קבוצת ערים לא כולל את העיר הראשונה.
   נגדיר $DP(S,i)$ להיות אורך המסלול הקצר ביותר שמתחיל בעיר הראשונה, מבקר בכל הערים בS ומסתיים בעיר i .
2. מקרה בסיס: לכל עיר i בקבוצה S, נאתחל $DP(\{i\},i)$ להיות אורך המסלול מעיר 1 לעיר i.
3. שלב הרקורסיה (בשילוב memoization table): לכל תת קבוצה S של ערים הכוללת את העיר הראשונה וכל עיר i ב S נחשב $DP(S,i)$ לפי הנוסחה הבאה:



- **אלגוריתם השכן הקרוב**

באלגוריתם חמדן זה בכל שלב הסוכן יפנה לעיר הקרובה אליו ביותר שעוד לא ביקר בה. בעזרת אלגוריתם זה ניתן לקבל במהירות מסלול קצר ויעיל. עבור מספר מסוים של ערים בפיזור אקראי לרוב יתקבל מסלול ארוך ב-25% מהמסלול הקצר ביותר.

<u>סיבוכיות זמן ריצה:</u> $O(n^2)$

ישנם מימושים שונים של האלגוריתם המשתמשים במבני נתונים שונים מה שיכול להשפיע על הסיבוכיות שלו.

המסלול שהתקבל על ידי הרצת אלגוריתם זה:

<u>אורך המסלול שהתקבל הינו:</u> **8314**

- **אלגוריתם קירוב Christofides**

אלגוריתם קירוב המבטיח פקטור קירוב של $3/2$ מהפתרון האופטימלי.
שלבי האלגוריתם :

1. יצירת גרף שלם מרשימת הערים, נסמנו בG. יצרנו גרף שבו הערים הם הקודקודים וישנן קשתות בין כל עיר לעיר. משקל הקשתות הינו המרחק בין הערים.
2. מציאת עץ פורש מינימלי לגרף G, נסמנו T.
3. נסמן ב O את קבוצת הקודקודים בעלי דרגה אי זוגית ב T. (נשים לב שמספר הקודקודים שהדרגה שלהם היא אי זוגית הוא זוגי - נכון לכל גרף).
4. מציאת שידוך מושלם בעל משקל מינימלי ב O, נסמנו M.
5. נאחד את הקשתות של M ו T ונקבל מולטי גרף קשיר H שבו כל קודקוד בדרגה זוגית.
6. נמצא מעגל אוילר ב H (בגלל ש H קשיר וכל הדרגות בו זוגיות, קיים בו מעגל אוילר).
7. נהפוך את מעגל האוילר שמצאנו למעגל המילטון על ידי קיצורי דרך - נסיר קודקוד שמופיע יותר מפעם אחת.

סיבוכיות זמן ריצה : $O(n^3)$

ניתוח זמן הריצה:

שלב 1: יצירת הגרף, מעבר על כל זוגות הערים ויצירת קשתות ביניהם: $O(n^2)$.

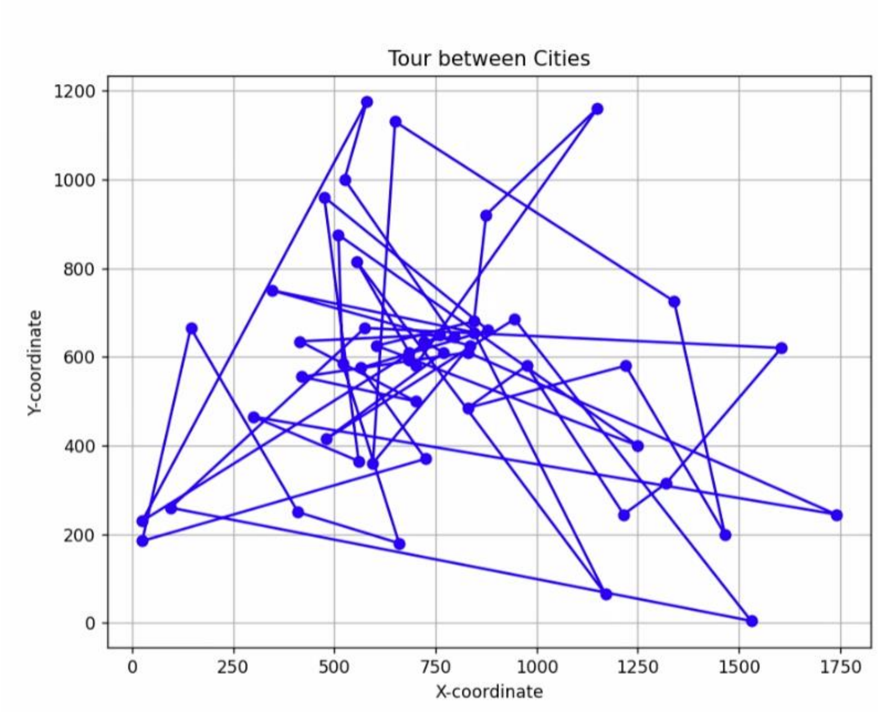
שלב 2: מציאת עץ פורש מינימלי: ניתן למצוא עץ פורש מינימלי ע"י אלגוריתם קרוסקל למשל עם סיבוכיות זמן $O(n^2\ log\ n)$ (שקול ל $O(|E|\ log\ |V|)$).

שלב 4: ישנו אלגוריתם למציאת שידוך מושלם מינימלי עם סיבוכיות $O(n^3)$.

שלב 6: סיבוכיות מציאת מעגל אוילר בגרף הינה $O(n^2)$ שקול ל $O(|E|)$.

שלב 7: סיבוכיות שלב קיצורי הדרך הוא $O(n^2)$ כמספר הצלעות בגרף.

<u>אורך המסלול שהתקבל הינו:</u> **8810**



- **<u>אלגוריתם אבולוציוני</u>**

אלגוריתמים אבולוציוניים זוהי משפחה של אלגוריתמי בינה מלאכותית המבוססים על רעיונות השאובים מאבולוציה בטבע כגון רעיון "הברירה הטבעית" על מנת לפתור בעיות.
אלגוריתמים אלו משמשים בעיקר כדי לפתור בעיות אופטימיזציה שלא ידוע עבורן פתרון דטרמיניסטי או הסתברותי העובד בזמן סביר.
העיקרון של אלגוריתם גנטי מייצג את עקרון "הדרוויניזם". עיקרון זה קובע כי כל מיני האורגניזמים נוצרים ומתפתחים באמצעות הברירה הטבעית של וריאציות תורשתיות המגבירות את יכולתו של הפרט לשרוד, כלומר- החזקים שורדים. אלגוריתם אבולוציוני מחקה את עיקרון זה ויוצר דורות חדשים, כאשר בכל דור בוחר את הפרטים "החזקים" באוכלוסייה, יוצר מהם דור חדש, ומבצע מוטציות על הצאצאים בדור החדש.
באופן זה, כל דור חדש מכיל צאצאים "חזקים" יותר. בכל שלב, אנו מחשבים את ה"חוזק" של כל פרט (על ידי פונקציית הפיטנס), וככל שהפרט יותר חזק, סיכוי גבוה יותר שהוא יבחר כהורה של הדור הבא. הילדים נוצרים מביצוע פעולות על ההורים שנבחרו (crossover). לעיתים על הילדים מתבצעת מוטציה (בהסתברות מסוימת) והם בעצם יהיו דור ההמשך. נמשיך ככה כמספר הדורות שנקבע, ולרוב, כמו הטבע, החזקים ישרדו וימשיכו.
באלגוריתם אבולוציוני קיים היבט הסתברותי (כמו בטבע) ושינוי ערכים של פרמטרים הסתברותיים (בהתאם לבעיה אותה מנסים לפתור) מאפשר מציאת פתרון מקורב יותר לפתרון המדויק של הבעיה.

<u>מבנה הפרט</u>

כל פרט מייצג פתרון חוקי לבעיה (תמורה של הערים).

הפרט הוא מערך בגודל מספר הערים כאשר כל עיר מופיעה בו פעם אחת בלבד.

אתחול האוכלוסייה
נאתחל m (גודל האוכלוסיה שנבחר) מערכים, כאשר כל מערך הוא באורך מספר הערים באופן רנדומלי. כלומר, הגרלנו m תמורות רנדומיות של הערים.

אופן חישוב הFitness
עבור כל פרט חישבנו את אורך המסלול שהוא מייצג על ידי הפונקציה calc_distance במחלקה City. פונקציה זו מחשבת מרחק אוקלידי בין שתי ערים.
פונקציית הפיטנס שבחרנו היא {אורך המסלול של הפרט / 1} משום שרצינו להפוך את הבעיה לבעיית מקסימיזציה. ניתן לראות שככל שאורך המסלול קטן, ערך פונקציית הפיטנס עולה.

יצירת הדור הבא
כל עוד גודל הדור הבא קטן מגודל האוכלוסייה הנדרשת:
- נבחר שני הורים ע"י ביצוע טורניר בגודל חצי מהאוכלוסייה.
- נבצע crossover על שני ההורים שנבחרו ונקבל שני ילדים. ה- crossover שבחרנו לבצע הוא בחירה של אינדקס רנדומלי מאחד ההורים. הילד יהיה מורכב מההורה הראשון עד האינדקס שהוגרל ולאחר מכן נעבור על ההורה השני ונוסיף לילד כל עיר שלא נמצאת לפי סדר הערים בהורה השני.
- על כל אחד מהווקטורים באוכלוסיה החדשה נבצע מוטציה בהסתברות p (אנחנו נשווה בין הסתברויות שונות לביצוע המוטציה). המוטציה מגרילה שתי ערים באופן אקראי ומחליפה ביניהן.
- נשים לב שגם לאחר ביצוע crossover וגם לאחר ביצוע מוטציה, כל הפרטים באוכלוסייה מהווים פתרון חוקי לבעיה, כלומר תמורה של הערים.

סיבוכיות זמן ריצה: $O(gen * (pop^2 + n * pop))$
ניתוח זמן הריצה:
סימונים:
- pop - גודל אוכלוסייה
- gen - כמות הדורות
- prob - הסתברות לביצוע מוטציה
- n - מספר הערים

1. אתחול האוכלוסייה: יצירת pop תמורות בגודל n $O(pop * n)$
שלבים 2-5 מבוצעים בכל דור, נתייחס לעלות החישוב עבור דור יחיד:
2. חישוב הפיטנס: נשים לב שעבור פרט יחיד, עלות חישוב הפיטנס היא $O(n)$, ועבור חישוב הפיטנס לכלל האוכלוסייה נקבל $O(pop * n)$.
3. Selection: $O(pop)$. מבצעים זאת מספר פעמים כגודל האוכלוסייה לכן $O(pop^2)$
4. Crossover: $O(n)$. מבצעים זאת מספר פעמים כגודל האוכלוסייה לכן $O(n * pop)$
5. Mutation: $O(1)$ בממוצע. מבצעים זאת על כל פרט באוכלוסייה לכן: $O(pop)$.
סך הכל עבור שלבים 2-5 נקבל, $O(pop^2 + n * pop)$
עבור כלל הדורות באלגוריתם, זמן הריצה יהיה: $O(gen * (pop^2 + n * pop))$

ניסויים- בחירת ערכים אופטימליים
לאלגוריתם אבולוציוני יש פרמטרים רבים אותם אפשר לבחון עד למציאת ערכם האופטימלי: פונקציית fitness, גודל האוכלוסייה, פונקציית crossover, פונקציית mutation, הסתברות למוטציה, אופן בחירת ההורים (ביצוע טורניר או באופן אחר), מספר הדורות ועוד.
אנחנו בחרנו להתמקד במציאת פרמטרים אופטימליים עבור גודל האוכלוסייה, מספר הדורות וההסתברות לביצוע מוטציה.

הרצנו סך הכל 27 ניסויים.

ניסוי משקף קבוצה ייחודית של פרמטרים בהם נשתמש להרצת האלגוריתם.
כל ניסוי נריץ 20 פעמים.
הרצה במשך 20 פעמים חיונית משום שהאלגוריתם מבצע בחירות אקראיות, הרצה יחידה לא תמיד משקפת בצורה אופטימלית את טיבו. בדקנו עבור כל ניסוי את הממוצע של 20 הריצות - מבחינת הפיטנס הטוב ביותר, הפיטנס הממוצע והפיטנס הגרוע ביותר.
לבסוף ננתח כל ניסוי לפי הערכים הממוצעים שהתקבלו עבורו על מנת לבחור בפרמטרים אופטימליים לפתרון בעיית TSP.

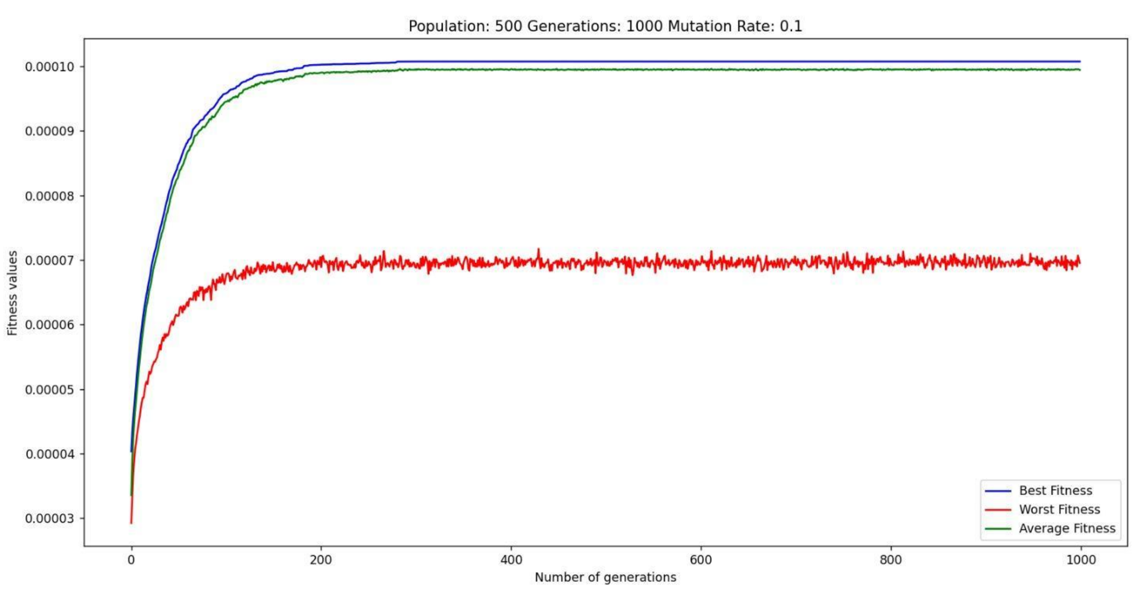
פרמטרים וערכיהם :

- גודל האוכלוסייה : [100,300,500]
- מספר דורות : [100,500,1000]
- הסתברות למוטציה : [0.05,0.09,0.1]

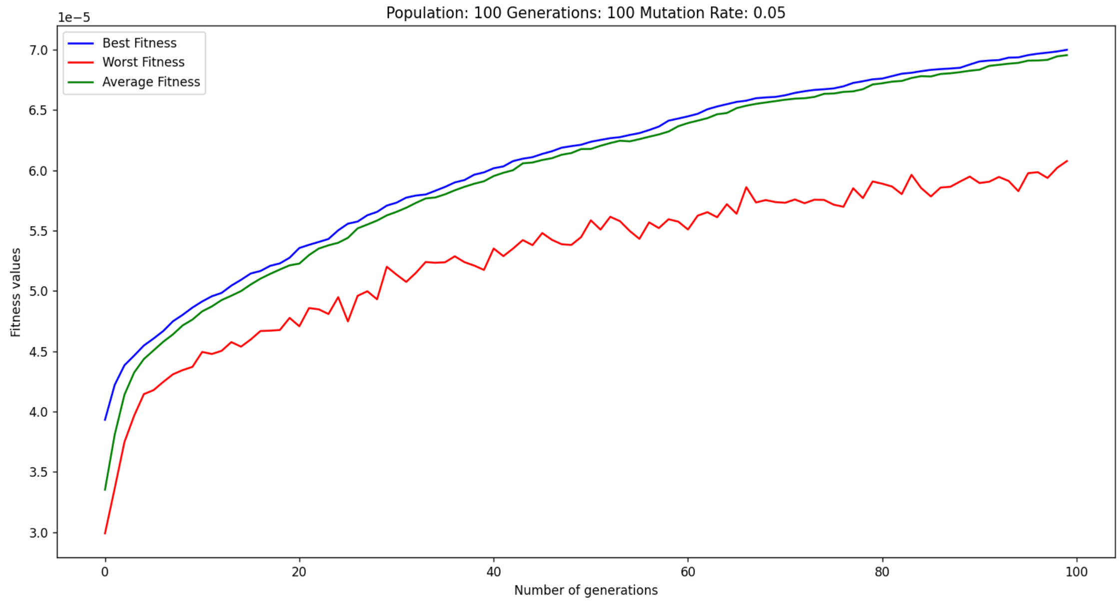
בדקנו את כל הקומבינציות של אפשרויות אלו.

תוצאות

תוצאה אופטימלית :
מבין כל הניסויים, הניסוי שהשיג את התוצאה הטובה ביותר (כלומר הממוצע של אורך המסלולים של 20 ההרצות היה הנמוך ביותר) הוא הניסוי בו גודל האוכלוסייה היה 500, מספר הדורות היה 1000 וההסתברות למוטציה הייתה 0.1. ניסוי זה הגיע לתוצאה ממוצעת של 9943.



תוצאה גרועה ביותר :
הניסוי שהשיג את התוצאה הגרועה ביותר הוא הניסוי בו גודל האוכלוסייה היה 100, מספר הדורות היה 100 וההסתברות למוטציה הייתה 0.05. ניסוי זה הגיע לתוצאה ממוצעת של 14331.



הגרפים עבור שאר הניסויים נמצאים בתיקייה evolution_graphs. תוצאות כל ניסוי נמצאים בקובץ results.txt וממוצע התוצאות של כל ניסוי נמצא בקובץ average_results.txt.

מסקנות

1. כאשר גודל האוכלוסייה גדל, האלגוריתם מוצא פתרונות יותר טובים, כלומר מסלולים יותר קצרים.
למשל לאחר התבוננות בניסויים שבהם מספר הדורות הוא 500 וההסתברות למוטציה היא 0.09 קיבלנו שכאשר גודל האוכלוסייה הוא 100, אורך המסלול הממוצע הוא 10852.2, כאשר גודל האוכלוסייה הוא 300, אורך המסלול הממוצע הוא 10235.05 וכאשר גודל האוכלוסייה הוא 500 אורך המסלול הממוצע הוא 10081.35.
ניתן לשים לב שככל שגודל האוכלוסייה גדל, אורך המסלול הממוצע קטן. עם זאת, צריך לשקול גם שככל שגודל האוכלוסייה גדל, כך גם סיבוכיות החישוב גדלה וזמן ריצת האלגוריתם מתארך.

2. שמנו לב גם כי כאשר מספר הדורות גדל, האלגוריתם מתקרב לפתרון האופטימלי.
למשל לאחר התבוננות בניסויים שבהם גודל האוכלוסייה הוא 100 וההסתברות למוטציה היא 0.1 קיבלנו שכאשר מספר הדורות הוא 100, אורך המסלול הממוצע הוא 12986.5, כאשר מספר הדורות הוא 500, אורך המסלול הממוצע הוא 10896.85 וכאשר מספר הדורות הוא 1000 אורך המסלול הממוצע הוא 10512.65.
ניתן לשים לב שככל שכמות הדורות גדלה, אורך המסלול הממוצע קטן.

3. לגבי ההסתברות למוטציה, בחלק מהניסויים הסתברות 0.1 הביאה לפתרון הטוב ביותר ובחלק מן הניסויים הסתברות 0.09 הייתה יותר טובה. בכל המקרים, שתי ההסתברויות האלו הביאו לפתרונות יותר טובים מאשר הסתברות 0.05.

לאחר התבוננות בניסויים בהם גודל האוכלוסייה הייתה 100 ומספר הדורות 500, ניתן לראות שכאשר ההסתברות למוטציה היא 0.05, אורך המסלול הממוצע הוא 14431.65 , כאשר ההסתברות למוטציה היא 0.09, אורך המסלול הממוצע הוא 13218.95 וכאשר ההסתברות למוטציה היא 0.1, אורך המסלול הממוצע הוא 12986.5. בניסוי זה, ככל שההסתברות למוטציה עלתה, אורך המסלול הממוצע ירד.

אך לאחר התבוננות בניסויים בהם גודל האוכלוסייה הייתה 100 ומספר הדורות 1000, ניתן לראות שכאשר ההסתברות למוטציה היא 0.05, אורך המסלול הממוצע הוא 10695.8 , כאשר ההסתברות למוטציה היא 0.09, אורך המסלול הממוצע הוא 10376.05 וכאשר ההסתברות למוטציה היא 0.1, אורך המסלול הממוצע הוא 10512.65. בניסוי זה, ההסתברות למוטציה 0.09 הביאה לתוצאות יותר טובות.
לכן לא ניתן להגיע למסקנה ישירה לגבי ההסתברות האופטימלית למוטציה.

סיכום

	זמן ריצת האלגוריתם	פלט אופטימלי
אלגוריתם נאיבי	$O(n!)$	7542
תכנון דינאמי	$O(2^n n^2)$	7542
אלגוריתם השכן הקרוב	$O(n^2)$	8314
אלגוריתם Christofides	$O(n^3)$	8810
אלגוריתם אבולוציוני	$O(gen * (pop^2 + n * pop))$	9943

עבור הקלט שעל בסיסו בחנו את דיוק האלגוריתמים (52 ערים), ניתן לראות כי אלגוריתם השכן הקרוב הינו האלגוריתם אשר חישב את אורך המסלול הקצר ביותר מבין שאר האלגוריתמים המקורבים.
היבט נוסף בו אלגוריתם זה הביע תוצאה טובה יותר הינו סיבוכיות זמן הריצה.

יש לציין כי עבור ערכים שונים עליהם היינו מריצות את האלגוריתמים ייתכן והיו מתקבלות תוצאות אחרות. לדוגמא, עבור מספר ערים גדול יותר, זמן הריצה של האלגוריתם האבולוציוני היה גדל באופן לינארי בעוד שזמן הריצה של אלגוריתם השכן הקרוב היה גדל באופן ריבועי.
נקודה נוספת שיש לציין הינה שהאלגוריתמים המקורבים הגיעו לתוצאה אופטימלית בקירוב נמוך ובזמן ריצה סביר, בעוד שלא ניתן לחשב עבור גודל קלט זה את הפתרון המדויק על ידי האלגוריתם הנאיבי ועל ידי תכנון דינאמי.