**מעבדה 2**

**אחמד מסאלחה 207923483**

**META HEURISTICS SWARM INTELLIGENCE CVRP**

**אינטילגנצית הנחיל**

**\*\* מצורף קובץ EXE בשם CRPV\_Solver שניתו להפעיל ולעקוב אחרי ההוראות, או ניתן להפעיל קוד פייתון באותו שם של ה EXE רק קובץ פייתון**

**הערות ושיפורים:**

1. האלגוריתם הגנטי לא טוב, לא היה מספיק זמן להשקיע ולכן אפשר לשפר ביצועים מבחינת זמן ריצה ופתרון
2. היה אפשר לשפר סיבכיות של כל האלגוריתמים עיי שנבנה מטריצת מרחקים במקום לחשב מרחקים כל פעם מחדש
3. לינק ל EXE של התוכנית: [EXE file](https://drive.google.com/drive/folders/1tisdZF8CroxVk9D_k-Ia_4_iXscybMhT?usp=sharing)

**חלק א': טיפול בקלט/פלט, ניתוח הבעיה ובנית אב-טיפוס לפתרון:**

1. פורמט הקלט הוא קובץ, כאשר כל קורה מגידרה רכיב מסויים מהעיה כאשר:
   1. NAME זה בעצם השם של הבעיה שאנחנו רוצים לפתור, למשל: E-n33-k4
   2. COMMENT זאת השורה ממנה נקרא את הערך האופטימלי של הפתרון
   3. CAPACITY זה יהיה בעצם כתוכלת או נפח של כל משאית
   4. NODE\_COORD\_SECTION זאת בעצם שורה שאחריה מתחילה סדרה של קורדינאטות ומספר, כאשר המספר יהווה מספר החנות והקורדינאטה תהווה את המיקום של החנות, למשל:

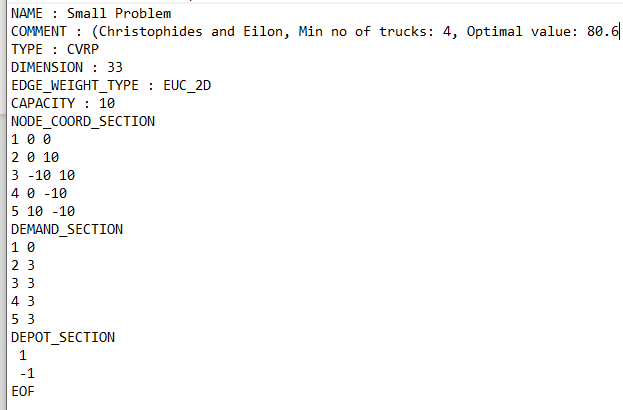
NODE\_COORD\_SECTION

1 292 495

2 298 427

הערה: בקבצי הקלט חנות מספר 1 הייתה בעצם נקודת הייצאה של המשאיות, ולכן בעצם החנויות החלו מ 2 והלאה.

* 1. DEMAND\_SECTION זה בעצם הנתון עבור כל חנות שהסברנו עליה ב d כמה היא צריכה, כלומר ה demand עבור החנות ה i (גם כאן 1 הייתה המחסן כי ה demand שלה 0).

1. מ סעיף קודם, ומהלינק המצורף, נבנה קלט המתאים לדוגמה שבדוח:
   1. 
   2. לבעיה קראנו Small Problem הפתרון אופטימלי עבורה הוא 80.6, תוכל הרכבים היא 10, מחסן הוא מס 1 עם קורדינאטה 0 0 ו demand 0 (שוב כמו שהסברנו, בגלל שהמחסן הוא 1 אז נקרא לו 0 וכל השאר נמספר מ 1 והלאה..)

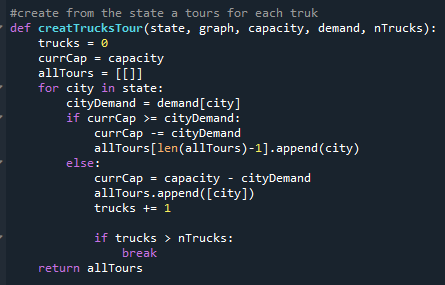
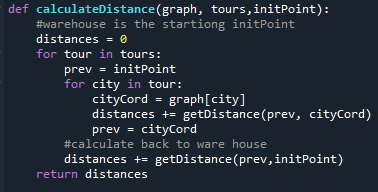
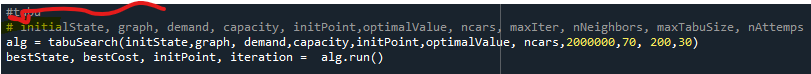
\*\* לגבי קריאת הקלט, אנחנו עוברים על הקלט, שורה שורה, ולפי ההסבר ב 1 אנחנו נבנה את המשתנים בהתאם, במימוש שלנו יש את הקבצים הבאים:

Functions.py

זה בעצם קובץ המכיל פונקציות כלליות, למשל חישוב מרחקים, חישוב מסלול, חלוקת מסלול, וכמובן קריאת הקלט, שתחזיר לנו:

1. initPoint שזה קורדינאטת המחסן
2. Capacity תוכלת המשאית
3. Graph שזה מילון של קורדינאטות כאשר graph[i] יתן לנו את המיקום של החנות ה i
4. Demand כמו graph אבל מחזחיר את ה demand
5. optimalValue הערך האופטימלי הנתון עבור הבעיה

הייצוג שלקחנו ב SA ו TABU ן GA זה בעצם פירמוטציה של האינקסים של החנויות, ואת הפרמוטציה הזאת נחלק למשאיות בהיוריסטיקה שנסביר עליה ב 3.

1. היוריסטיקה שלנו היא יוריסטיקה בעצם KNN כאשר ניקח הכי טוב מבין שכנים שנייצר, איך נידע שהוא הכי טוב? חמדנית, אנחנו בעצם נקבל פרמוטציה, וננסה לחלק אותה למשאיות באופן חמדני, כך שכל עוד יש תוכלת במשאית עבור החנות הבאה אז נכניס אותה למשאית הנוכחית, אחרת, נוציאה את המשאית למסלול שקבענו, ונתחיל למלאה משאית חדשה.
   1. עבור המעבר מ state לאחר, או לשכן אחר, השתמשנו באופרטור שמחליף לנו 2 קשתות, כלומר נחלים מיקום של 2 חנויות.
   2. 
   3. ואז אחרי שנבנה מסלולים כאלה, נרצה לחשב את העלות, ולכן נחשב לכל משאית את המרחק של הדרך שהיא תיקח:
      1. 
      2. ובסוף נקבל ערך עבור הפתרון של ה state שיש לנו ביד.
2. קידוד האלגוריתמים:
   1. Tabu search
      1. מימוש האלגוריתם בקובץ tabuSearch.py, הממש מחלקת TABU עם פונצקיה בשם run שתחזיר לנו פתרון. אתחול אובייקט מסוג המחלקה הזאת יבקש:
         1. כאשר מה שנשלח לבנאי של המחלקה הוא מה שמסומן אחרי # (אחרי האדום)
         2. וקריאה לפונקצית run תחזיר את הפתרון הכי טוב הגענו אליו, עם העלות ונקודת ההתחלה
      2. עבור האלגוריתם של TABU השתמשנו בפסודו קוד מ: [TABU FROM WIKI](https://en.wikipedia.org/wiki/Tabu_search" \l ":~:text=Tabu%20search%20is%20a%20metaheuristic,1986%20and%20formalized%20in%201989.) עם שינוי קטן, שכל פעם יצרנו 100 שכנים, ולקחנו את השכן עם הערך הטוב לצורך השוואה עם ה BEST וה CURRENT
      3. שינוי שעשינו זה בעצם לשחרר 50 מקומות מהטאבו כאשר הוא מתמלאה: זו בעצם בחירה שניתן לשפר, למשל להקציב זמן עבור כל state ולא להוציאה 50 מצבים ישנים.
      4. האלגוריתם יעצור כאשר הזמן המוקצב נגמר, או שהגענו לפתרון האופטימלי, או שהתכנסנו לפתרון ולא מצליחים לצאת
   2. Simulated annealing
      1. האלגוריתם ממוש עיי המחלקה simulatedAnnealing זנמצאת ב simulatedAnnealing.py , הקלט למחלקה הוא כמו ב b בנוסף ל nAttemps שזה מספר האיטרציות לכל טמפרטורה, כאשר שמייצרים אובייקט מסוג הזה ניתן לקרוא לה עם הפונקציה simulate().
      2. האלגוריתם ממומש עיי הפסודוקוד הבא:

Temp = initialTemp

currState, bestState = initialState

for I in range(maxIter):

for j in range(maxIterForCurrTemp):

neighbor = getNeighbor(currState)

costDiff = getCost(neighbor) – getCost(currState)

if costDiff < 0:

currState = neighboor

if getCost(currState) < getCost(bestState):

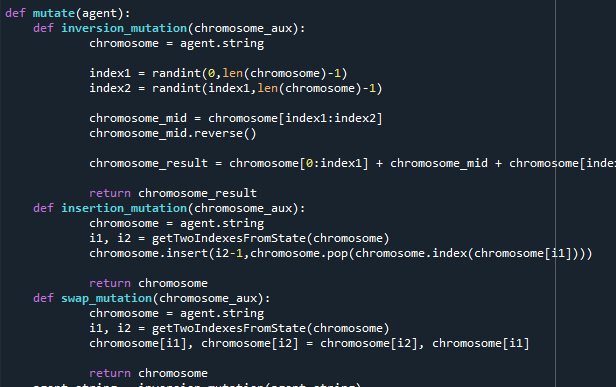
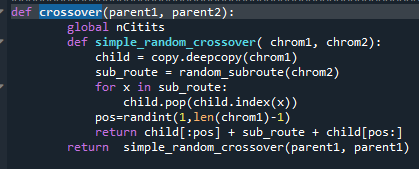
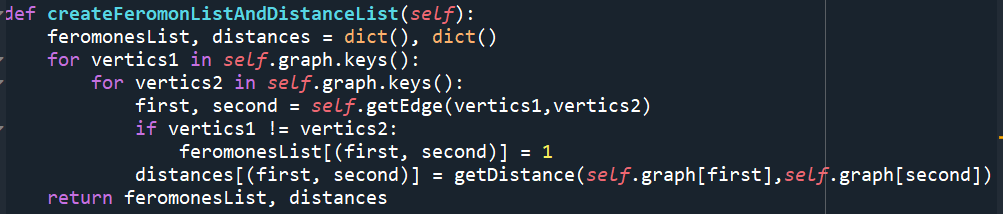
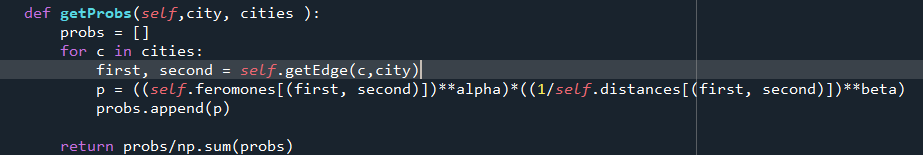
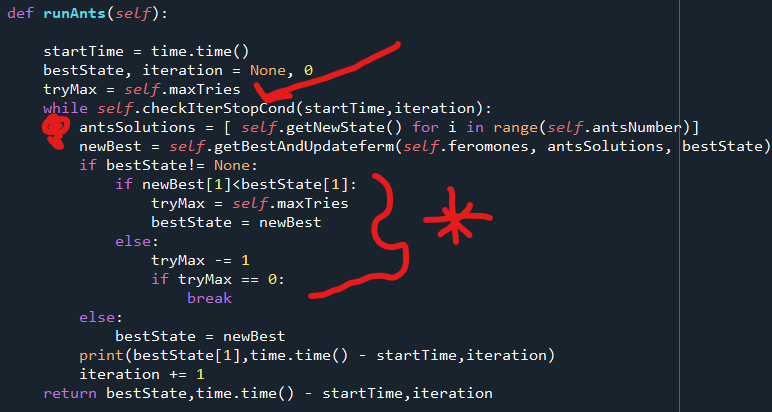
bestState = currState

else:

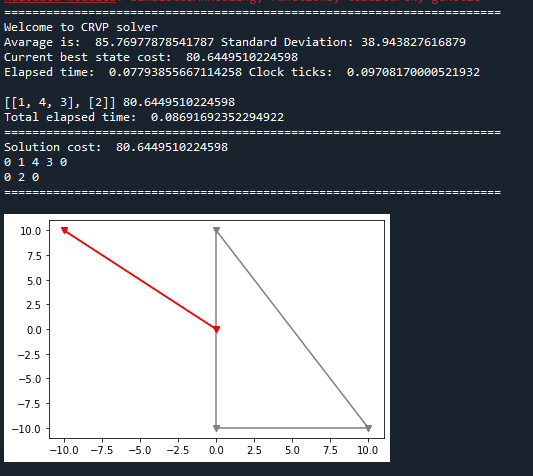
if randomNumber < exp(-costDiff/ temp):

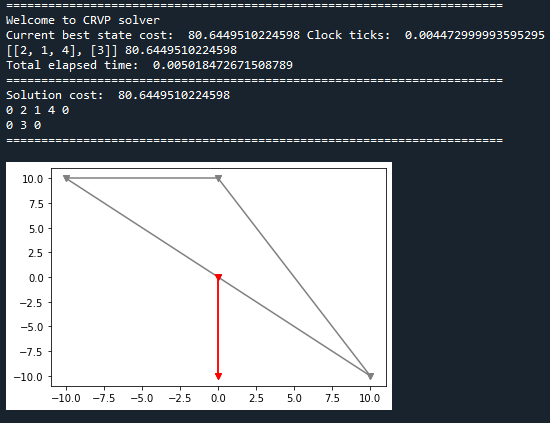
currState = neighbor

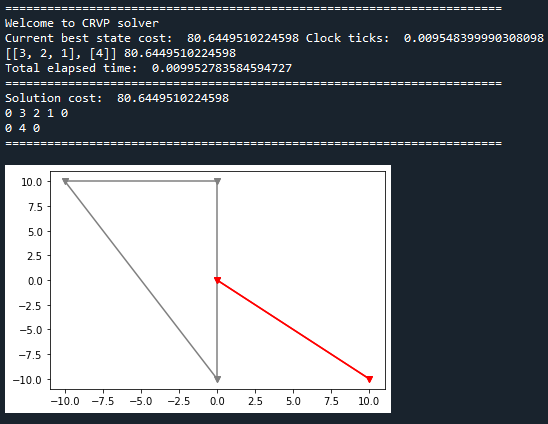
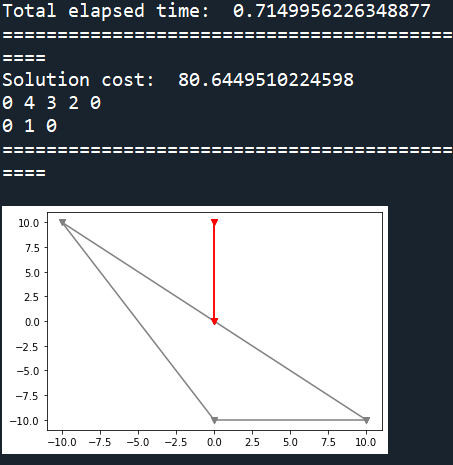
temp = temp \* 0.95

* + 1. אנחנו מורידים את הטמפרטורה כל פעם ב 0.95 מהקודמת, כאשר נעצור או שנגיע לאופטימלי, או שמתכנסים לעלות, או שנגמר הזמן.
  1. Genetic
     1. מומש עיי genetic.py
     2. התאמנו גיין לפרמוטציה
     3. inversion\_mutation כמוטציה ברירת מחדל
        1. אלה המוטציות הממומשות
     4. אופרטור שיחוף שמכניס סיור מהאם לאב לאינדקס רנדומלי תוך שהוא מוציא את כל החנויות שהאם ביקרה בהם בסיור שנבחר
        1. 
     5. \*\* האלגוריתם הגנטי הוא פחות טוב מהשאר כי לא היה לנו זמן להשקיע בו ☹
     6. הוא צורך זמן ולא מגיע לאופטימום
  2. ACO
     1. בשונה משאר האלגוריתמים כאן מימשתי מטריציית מרחקים ומטריציית פירומונים בגלל האופי של האלגוריתם, ומסתבר שזה משפר את המהירות בהרבה, לצערי לא היה זמן לשפר בשאר
        1. 
           1. כאשר GRAPH הוא קבוצת הקודקודים ו getEdge עושה סידור קשת, קבוצת הפירומונים מאותחלת ל 1 לכל קשת
        2. חישוב ההסתברויות הוא בדיוק כמו בהרצאה (כמות הפירומון בחזקת אלפא הכל כפול 1 חלקי המרחק בחזקת ביתא ומחלקים בסכום)
           1. 
        3. עדכון פירומונים וקבלת מצב טוב חדש להשוואה:
           1. 
        4. לולאה ראשית – לכל נמלה ניקח מצב חדש, נקרא לפונקציה ב 3 ונשווה המצב שיחזור עם הטוב ביותר עד עכשיו, נעשה את זה עד שיתקבל מצב עצירה ( אסביר ב 5)
           1. 

פונקציית checkIterStopCond בודקת מתיי עוצרים, וזה בעצם כשמגיעים ל מספר איטרציות מקסימלי שמוגדר, למשל 100000, או שנגמר זמן בשניות, שזה כמו אלגוריתמים קודמים, 0.6 מה dimension (ראה הערות בהתחלת חלק ב) ואיפה שיש כוכבית זה גם עצירה וזה קורא שמתכנסים לאותו ערך 70 פעמים וזה הותאם ספיציפית לאלגוריתם זה (כלומר הקרבה של 70 איטרציות כדי לוודא על הפתרון)

1. היוריסטיקה החמדנית שהשתמשנו בה בעצם מחלקת פירמוטציה כלשהיא למסלולים, ולכן אם היא קרובה לפרמוטציה האופטימלית אז נקבל יותר חלוקה טובה, ולכן נתקרב יותר לפתרון האופטימלי.
2. תוצאות ריצות על דוגמה הצעצוע - הבעיה היא קטנה והאלגוריתמים (כולל הגנטי) מוציא אופטימלי ובזמן נמוך:
   1. Genetic
   2. SA



* 1. Tabu Search
     1. 
  2. ACO
     1. 

הביצועים מהירים ומגיעים לאופטמון במהירות גבוה בכל האלגוריתמים

**חלק ב': בחינה השוואתית של האלגוריתמים על בעיות "גדולות" וניתוח תוצאות**

\*\* בגלל המימושים הקצבנו 85% מה diminssion הנתון כשניות ל TABU ו GENETIC בגלל המימוש, ו 0.6 ל SA ו ACO

1. מצורף קבצי טקסט (SA\_results Tabu\_results Genetic\_results ACO\_results) בכל אחד עבור על בעיה תוצאה של 10 ריצות עם הפתרון, זמן, איטרציה, ועלותר הפתרון
2. הטבלה מתבססת על ממוצע כל 10 ריצות בסעיף 7

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| מס בעיה | פתרון אופטימלי | אלגוריתם | מצא פתרון אופטימלי? | עלות הפתרון | מרחק | זמן בשניות |
| E-n22-k4 | 375 | SA | באחד הריצות מ 11 | 405.9107 | 30 | 3.8 |
| E-n33-k4 | 835 | SA | X | 943.3573 | 108 | 7 |
| E-n51-k5 | 521 | SA | X | 715.0124 | 194 | 15 |
| E-n76-k8 | 735 | SA | X | 1006.572 | 270 | 43.20174 |
| E-n76-k10 | 832 | SA | X | 1117.578 | 285 | 40.48501 |
| E-n101-k8 | 817 | SA | X | 1253.372 | 436 | 58.8947 |
| E-n101-k14 | 1077 | SA | X | 1438.683 | 361 | 59.84243 |
| E-n22-k4 | 375 | Tabu | פעמיים | 404.9414 | 29 | 8.466369 |
| E-n33-k4 | 835 | Tabu | X | 919.9683 | 84 | 15.98411 |
| E-n51-k5 | 521 | Tabu | X | 636.9445 | 115 | 36.08181 |
| E-n76-k8 | 735 | Tabu | X | 966.0953 | 231 | 58.2853 |
| E-n76-k10 | 832 | Tabu | X | 1072.74 | 240 | 55.63668 |
| E-n101-k8 | 817 | Tabu | X | 1270.355 | 453 | 77.24029 |
| E-n101-k14 | 1077 | Tabu | X | 1468.413 | 391 | 77.7501 |
| E-n22-k4 | 375 | Genetic | X | 463.2479 | 88 | 18.9959 |
| E-n33-k4 | 835 | Genetic | X | 1117.205 | 282 | 28.426 |
| E-n51-k5 | 521 | Genetic | X | 1195.372 | 647 | 43.8634 |
| E-n76-k8 | 735 | Genetic | X | 1931.569 | 1196 | 65.460 |
| E-n76-k10 | 832 | Genetic | X | 1995.414 | 1163 | 65.3815 |
| E-n101-k8 | 817 | Genetic | X | 2663.339 | 1846 | 86.77468 |
| E-n101-k14 | 1077 | Genetic | X | 2825.382 | 1748 | 86.83677 |
| E-n22-k4 | 375 | ACO | X | 380.0802 | 5 | 4.239051 |
| E-n33-k4 | 835 | ACO | X | 873.1231 | 38 | 8.70439 |
| E-n51-k5 | 521 | ACO | X | 607.6115 | 86 | 16.49 |
| E-n76-k8 | 735 | ACO | X | 847.5375 | 112 | 29.04973 |
| E-n76-k10 | 832 | ACO | X | 959.0386 | 1127 | 34.8422 |
| E-n101-k8 | 817 | ACO | X | 1003.079 | 187 | 46.59369 |
| E-n101-k14 | 1077 | ACO | X | 1275.207 | 198 | 46.92338 |

1. סיבכיות מקום של האלגוריתמים:
   1. ב TABU אורך הטאבו הוא 300 ויש לנו את ה BEST ו המצב בנוכחי, כאשר כל אחד בגודל n (מספר החנויות) ולכן סיבכיות המקום היא O(C\*n) כאשר c תלוי בגודל ה TABU במקרה שלנו 300
   2. עבור SA בכל רגע יש לנו את ה best curr next כאשר כל אחד בגודל n ולכן סיבכיות המקום היא O(n)
   3. ב genetic גודל האוכלסיה C כאשר כל אחד מכיל מצב בעל n איברים, ולכן נקבל סיבכיות של O(c\*n) במקרה שלנו C = 2048
   4. ב ACO אנחנו מחזיקים פירומונים וטבלט מרחקים שזה n\*n כל אחת ופתרונות של כל נמלה שזה c \* N כאשר c מספר הנמלות ו N מספר החנויות לכן נקבל:

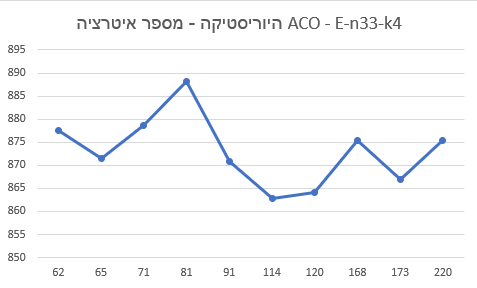
|  |  |
| --- | --- |
| אלגוריתם | סבכיות מקום |
| SA | O(n) |
| Tabu Search | O((2+c)\*n) כאשר C גודל הטאבו |
| Genetic | O(c\*n) כאשר C גודל האוכלסיב |
| ACO |  |

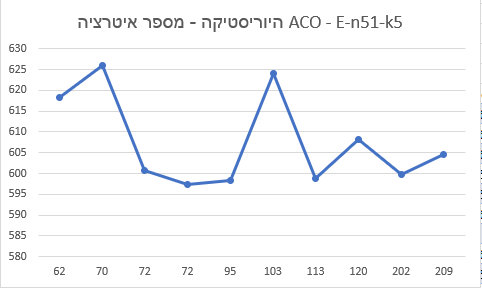
* 1. היה אפשר לשפר סיבכיות זמן ריצה עיי לוותר על עוד מקום למטריצה של מרחקים במקום כל פעם לעשות חישוב מחדש (מה שקורה ב ACO)

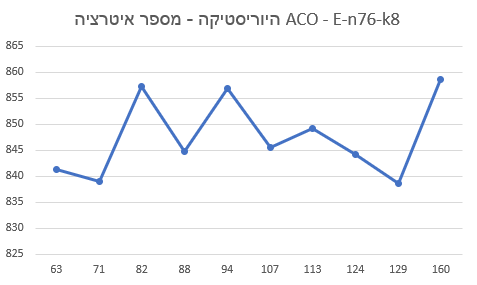
1. גרפים:
   1. של ערך ההיוריסטיקה הטוב ביותר בכל איטרציה של האלגוריתם:
      1. בעצם האלגוריתמים הם כמו anytime ובכל איטרציה יש לנו את ה best שיש לו את הערך היוריסטיקה הטובה ביותר – העלות הנמוכה, ולכן, אציג גרפים המראים העלות הכי טובה בכל איטרציה, ובעצם נקבל גרף יורד (ככל שמתקדמים העלות יותר טובה - נמוכה) ולכן הסעיף 10.2 נקבל אותם גרפים רק עולים, כלומר מתחילים באיכות נמוכה ועולים עם האיטרציות, לכן נציג סוג אחד של גרפים, והשני שקול רק הפוך (עולה במקום יורד)
   2. של איכות הפתרון כפונקציה של מספר האיטרציות

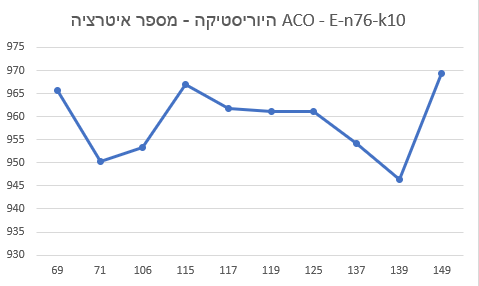
--------------------------------------------------------------------------------------------------------------------------------------

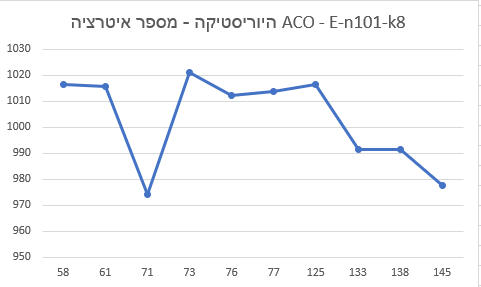
--------------------------------------------------------------------------------------------------------------------------------------

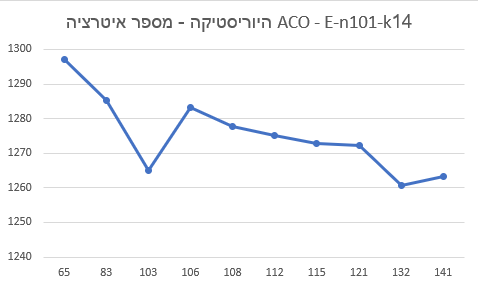












1. הסבר על הגרפים:
   1. לגבי ACO אומנם הם לא מונוטונים, וזה בגלל מימוש תנאי העצירה, אבל נותנים תוצאות טובות מכל האלגוריתמים + הדילתא בין התוצאות היא קטנה
   2. הגרפים הם יורדים, כלומר בכל האלגוריתמים העלות הולכת ויורדת להיות יותר טבה, עד שהיא מתכנסת.
   3. ב TABU ו SA האלגוריתם עוצר רק כאשר הוא מתכנס 30 פעם על אותו ערך, נגמר הזמן, ולכן רואים מספר איטרציות יותר גבוה בשניהם
   4. האלגוריתמים יעילים על בעיות קטנות, ככל שה dimension של הבעיה עולה, אז כך גם מספר האיטרציות וגם השגיאה מהערך האופטימלי.
   5. במימוש לא נעשה שום שימוש בנתון של הפתרון האופטימלי כך שזה נתון שיכל לשפר את עקומת הגרפים
   6. האלגוריתם הגנטי וה SA רואים מהעקומה שלוקח לה מספר איטרציות יותר מה TABU לרדת לעלות טובה, כאשר בTABU הוא מתקדם יותר מהר לאופטימלי, וזה רואים מהעקומה של הגרף.
   7. TABU מתנהג בצורה טובה גם עם בעיות יותר גדולות אך עדיין עם שגיאה טובה ככל שהבעיה גדלה.
   8. האלגוריתם הגנטי מתקשה, ורואים שלוקח לא מספר איטרציות יותר גדול כדי להתקדם.
2. הסקלביליות:
   1. לגבי האלגוריתם GENETIC רואים שהוא מתקשה גם בבעיה הקטנה, ולכן ככל שהבעיה תגדל נרצה לוותר עליו.
   2. ה SA יותר טוב מה GENETIC, ולא הולך ונהיה גרוע בשגיאה כמו GENETIC, ואולי עם מימוש יותר טוב, כמו מטריציית מרחקים וסידור דינאמי של פרמטרים יעבוד טוב
   3. ה TABU מתקדם לפתרון האופטימלי יותר טוב משאר האלגוריתמים, אך יש פחד גם שהוא יקנס ללוקאל יותר מהר, אבל ככל שהבעיה גדלה הוא התאים את עצמו יותר טוב.
   4. ACO נהדר לדעתי והוא מתנהג טוב עם בעיות גדולות, לדעתי מימוש יותר טוב והתנהגות פרמטרית דינאמית במקום קבועים יכול לשפר