בעיית נהג המונית

פרויקט סיום ב-Ai

יהל רייס 318160470

יעקב אברבוך 200284446

נועה בבליקי 206090409

הצגת הבעיה

הבעיה שבחרנו לפתור היא בעיית נהג המונית. בעיה זו היא וריאציה של בעיית הסוכן הנוסע (traveling salesman), בה נהג מונית צריך להסיע כמות מוגדרת מראש של נוסעים, שלכל נוסע נקודת איסוף ונקודת הורדה, במסלול הקצר ביותר. המונית יכולה להכיל את כל הנוסעים בכל זמן נתון. נוסע לא יכול לרדת מהמונית לפני שהוא עלה.

הקלט שלנו הוא רשימה של נוסעים, כאשר לכל נוסע המאפיינים הבאים: מס' זהות, נקודת איסוף (טאפל של קואורדינטות), נקודת הורדה (טאפל של קואורדינטות).
הפלט שלנו הוא רשימה עם סדר תחנות האיסוף וההורדה של כל הנוסעים. כל איבר מכיל נקודה כלשהי (איסוף או הורדה), מס' זהות של נוסע, האם זו תחנת איסוף או הורדה (בוליאני).

בעיה זו ניתנת לפתרון ע"י אלגוריתם brute force רקורסיבי כאשר יש לנו מספר קטן של נוסעים, אך ככל שעולים במספר הנוסעים מספר המסלולים האפשריים עולה אקספוננציאלית, ולכן דורש כח רב והמון מקום בזיכרון מהמחשב. לכן, החלטנו לפתור בעיה זו על ידי מציאת מינימום לוקאלי, באמצעות אלגוריתמי חיפוש שלמדנו בקורס.

גישות לפתרון הבעיה

אלגוריתמים נאיביים

1. רנדומלי – בוחר באקראי את המהלך הבא באופן שמציית לחוקים, כלומר קודם כל הוא יכול לבחור תחנת העלאה של נוסע ורק בהמשך הוא יכול לבחור את תחנת ההורדה של אותו נוסע.
2. חמדן – בוחר כל פעם את התחנה החוקית שהכי קרובה אליו, כלומר, ייתכן שיש תחנת הורדה שקרובה לתחנה הנוכחית שהאלגוריתם נמצא בה, אבל מכיוון שהנוסע הרלוונטי עוד לא הועלה האלגוריתם מתעלם מתחנה זו בינתיים.

אלגוריתמי חיפוש מקומי

אלגוריתם הבסיס הוא Local Search. האלגוריתם מתחיל ממסלול חוקי כלשהו ובכל איטרציה מנסה למצוא שכן של המסלול הנוכחי בעל אורך מסלול טוב יותר.

ווריאציות על האלגוריתם:

1. Hill climbing – בכל איטרציה האלגוריתם מחפש את השכן הטוב ביותר. אלגוריתם זה מתכנס מהר יותר ובדרך כלל אף משיג תוצאות טובות יותר.
2. Simulated annealing - בוחר בכל שלב שכן רנדומלי, אם הוא טוב יותר מהנוכחי עוברים אליו, אחרת עוברים אליו בהסתברות גבוהה יותר ככל שהוא פחות גרוע ובהסתברות גבוהה יותר ככל שהאלגוריתם עוד לא ביצע מספר רב של איטרציות (באלגוריתם מוגדרת טמפרטורה גבוהה שהולכת ויורדת עם כל איטרציה).
3. Multiple Local Search – מכיוון שכל ריצה של ה Local Search היא רנדומלית מבחינת המסלול השכן שאליו עוברים בכל איטרציה של שיפור, ריצות שונות של האלגוריתם יגיעו למסלול סופי שונה. לכן ניתן להריץ את אלגוריתם הבסיס מספר נתון של פעמים ולהחזיר את התוצאה הטובה ביותר.
4. Beam Local Search – אלגוריתם שמחזיק קבוצה בגודל קבוע של פתרונות, ובכל איטרציה מנסה למצוא שכן כלשהו של אחד הפתרונות שטוב יותר מאחד הפתרונות בקבוצה. אם הוא מצליח, הפתרון החדש יכנס לקבוצה במקום הפתרון הכי פחות טוב.

בכל אלגוריתם חיפוש מקומי, השיפור האיטרטיבי נעשה על ידי ביצוע שינוי קטן בפתרון הקיים. כדי להגדיר את קבוצת השכנים של פתרון (חוקי) נתון הגדרנו פעולת החלפה (Swap) באופן הבא:

אנו בוחרים תחנה כלשהיא במסלול ומנסים לשנות את המיקום שלה בתוך המסלול. אם בחרנו לשנות מיקום של תחנת הורדה של נוסע מסויים, השכנים יהיו הפתרונות בהם התחנה תמוקם מחדש בכל המקומות החל בתחנה שאחרי תחנת ההעלאה של נוסע זה ועד התחנה האחרונה.

באופן דומה, אם בחרנו לשנות מיקום של תחנת העלאה של נוסע מסויים, השכנים יהיו הפתרונות בהם התחנה תמוקם מחדש בכל המקומות החל בתחנה הראשונה ועד תחנה אחת לפני תחנת ההורדה של אותו נוסע. כך, מובטח לנו שחוקיות הפתרון תישמר.

אלגוריתם גנטי

זיווג (crossover):

ניסיון ראשון – ניסינו לחשוב איזה מאפיינים של ההורים נרצה להוריש לילדים כאשר מתבצע זיווג. ראשית, חשבנו שאם לשני ההורים יש רצף כלשהו משותף של תחנות איסוף והורדה אז הגיוני לשמר את הרצף הזה גם אצל הילד. לדוגמא, אם נמספר את הנוסעים ב-1,...,5 ובאדום וכחול את תחנות העלייה והירידה בהתאמה, אז עבור צמד ההורים הבא:

1	2	3	4	5	4	3	5	1	2

1	1	4	3	5	4	2	2	3	5

שימור הרצף המשותף יהיה:

				5	4				

כעת, ניסינו לחשוב כיצד ניתן להשלים את הרצף הגנטי, והחלטנו שתוך התעלמות מהרצפים המשותפים שכבר העתקנו לילד, לקחת רצף באורך רנדומלי N מההורה הראשון ולהעתיק אותו ל-N מקומות הפנויים הראשונים. כלומר, עבור הדוגמא שלעיל, נניח שיצא N=2, אז נקבל

1	2			5	4				

ואז נשלים מההורה השני באותו אופן, כלומר

1	2	1	4	5	4	3	2	3	5

הבעיה הייתה, כשהרצנו בדיקות עם האלגוריתם הזה, שהוא התכנס מהר מדי. הדמיון בין כל ילד לאחד ההורים (או לשניהם) גרם לכך שתוך מספר איטרציות בודד האלגוריתם התכנס לתוצאה הטובה ביותר הנוכחית ולא הצליח לצאת מהמינימום הלוקאלי הזה.

לכן –

ניסיון שני – עבור צמד ההורים הבא:

6	5	4	3	5	4	1	2	1	2	3	6

6	6	1	2	2	3	3	5	4	1	4	5

נבחר רצף רנדומלי מההורה הראשון,

3	5	4	1	2

כדי שהבן יהיה מסלול חוקי, צריך שתחנות ההורדה 1,2,3 יהיו אחרי הרצף, ותחנות ההעלאה 4,5 יהיו לפני הרצף. את הסדר בין תחנות 4,5 נקבע לפי ההורה השני, כלומר: קודם 5 ואז 4, ובאותו אופן, את הסדר בין תחנות 1,2,3.

גם את התחנות שנותרו: 6,6 נכניס לרצף לפי ההורה השני. ונקבל:

6	6	5	4	3	5	4	1	2	2	3	1

האלגוריתם הזה מתכנס יותר מהר ומגיע לתוצאות יותר טובות.

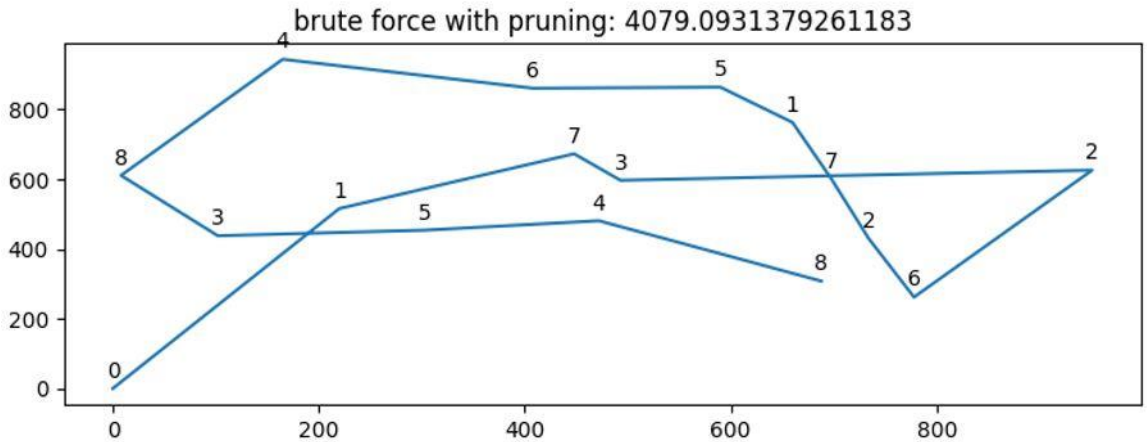
מוטציה (Mutation):

המוטציה שבחרנו לבצע דומה ל-Swap שביצענו בכל האלגוריתמים הלוקאלים, רק בלי לבדוק האם השינוי מוביל למצב טוב יותר.

אלגוריתם כוח גס

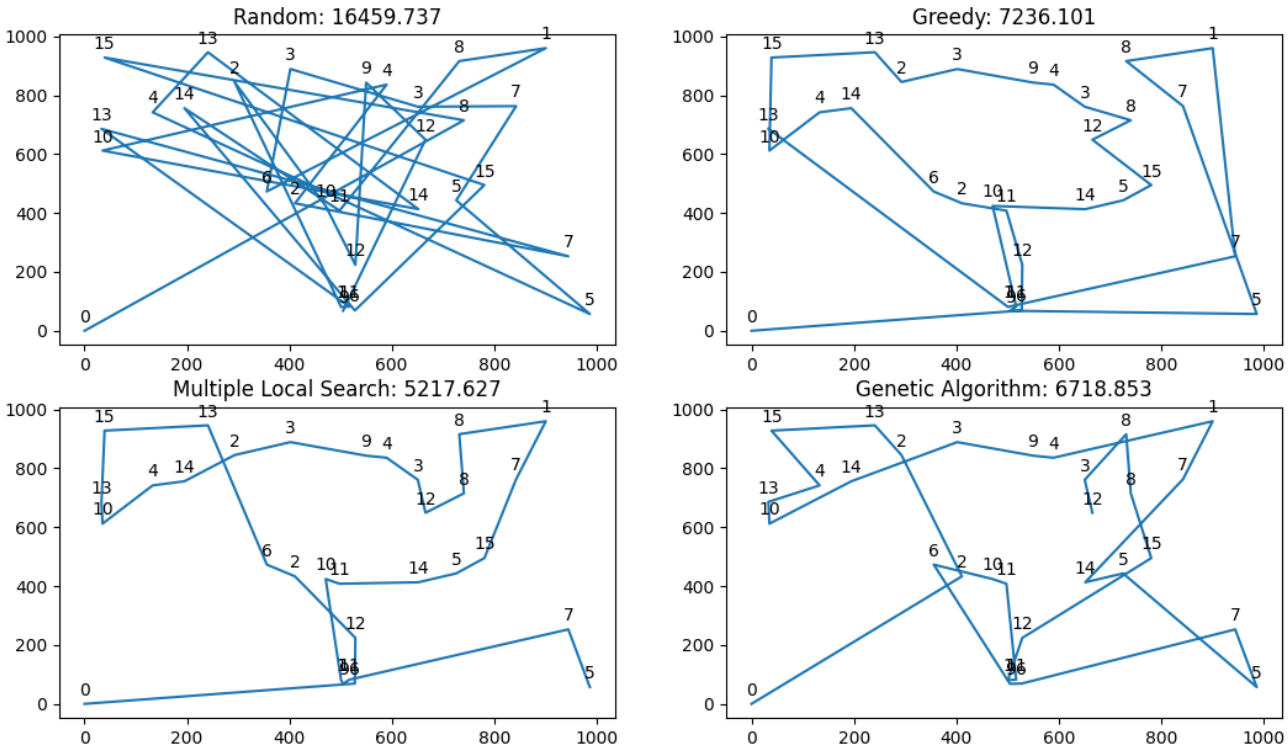
ניסיון נאיבי – הבאנו לאלגוריתם את רשימת הנוסעים והשתמשנו ב-itertools.permutations בשביל לקבל את כל דרכי הסידור האפשריים. אחרי זה מחקנו את הפתרונות הלא חוקיים (כאלו בהם הורדנו נוסע לפני שהעלנו אותו) ובדקנו מי מהפתרונות החוקיים הכי טוב. הבעיה, כמובן, הייתה שכבר עבור 6 נוסעים כמות הפרמוטציות האפשריות היא 479,001,600=12!, הרצה אפשרית אבל שלוקחת כבר קרוב לשעה. ב-7 נוסעים לפייתון נמאס לעבוד אחרי כמה שעות והוא החזיר שגיאה. לכן –

ניסיון שני, Brute Force with pruning – מכיוון שחלק ניכר מהפרמוטציות לא חוקי ומוביל להרבה עבודה מיותרת, החלטנו לכתוב פונקציה רקורסיבית שמתחילה ממערך עם נקודת ההתחלה בלבד ומוסיפה בכל שלב רק תחנות איסוף והורדה שחוקיות לפתרון. כדי לקצר עוד יותר את זמן הריצה, החלטנו גם לגזום קריאות מיותרות; העברנו לפונקציה משתנה best_dist שהוא מרחק של פתרון שהתקבל מאחד האלגוריתמים האחרים, ובכל שלב בבניה של הפתרון האופטימלי, אם האורך של הפתרון כבר היה גדול יותר מ-best_dist אז מחקנו את הענף, ואם מצאנו פתרון עם אורך טוב יותר אז עדכנו את best_dist בהתאם. כך, הצלחנו להריץ דוגמאות עם עד 9 נוסעים בזמן סביר יחסית (עד 9.5 שעות ריצה). להלן דוגמא להרצה של האלגוריתם על 8 נוסעים:


brute force with pruning: 4079.0931379261183
0
200
400
600
800
0
200
400
600
800
4
6
5
1
8
7
3
7
2
1
3
5
4
2
8
6
0

דוגמאות לפתרון

הרצנו 4 אלגוריתמים שונים על קלט של 15 נוסעים שמוגרלים רנדומלית עם seed=177.



תוצאות ומסקנות

התכנסות האלגוריתמים

כדי להשוות את הביצועים של האלגוריתמים אנו צריכים למצוא נקודת התכנסות של האלגוריתמים ולהשוות את הביצועים של האלגוריתמים בנקודת ההתכנסות. לאלגוריתם ה Local Search, Hill climbing יש נקודת התכנסות של האלגוריתם - כאשר כל שינוי (Swap) שנעשה ייתן מסלול שאינו קצר יותר.

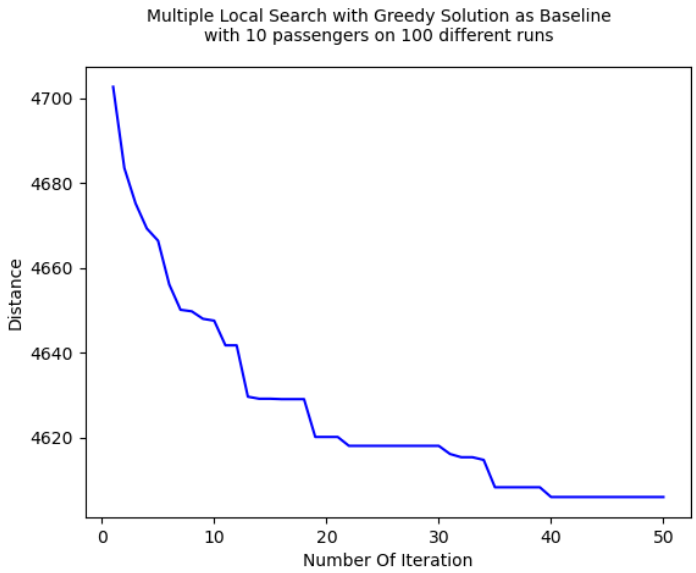
מתוך אלגוריתמי החיפוש הלוקאלי, מצאנו שהאלגוריתם שמגיע לתוצאות היותר טובות הוא הרצה של Local Search מספר רב של פעמים ובחירת הטוב מבניהם (Multiple Local Search). אפשר להריץ את ה Local Search אינסוף פעמים ולהתקרב לפתרון האופטימלי, אך אנו ניסינו לבדוק האם ניתן להגדיר נקודת התכנסות - מספר הרצות כך שממנו והלאה לא נראה שיפור משמעותי.

בנוסף, אנו צריכים נקודת התכנסות עבור האלגוריתם הגנטי.

התכנסות של Multiple Local Search Algorithm

כאמור, אנו מריצים את Local Search מעל פתרון בסיסי ומנסים לשפר אותו בצורה רנדומלית. פתרון הבסיס יכול להיות פתרון רנדומלי שהוא ארוך יותר או פתרון של האלגוריתם החמדני.

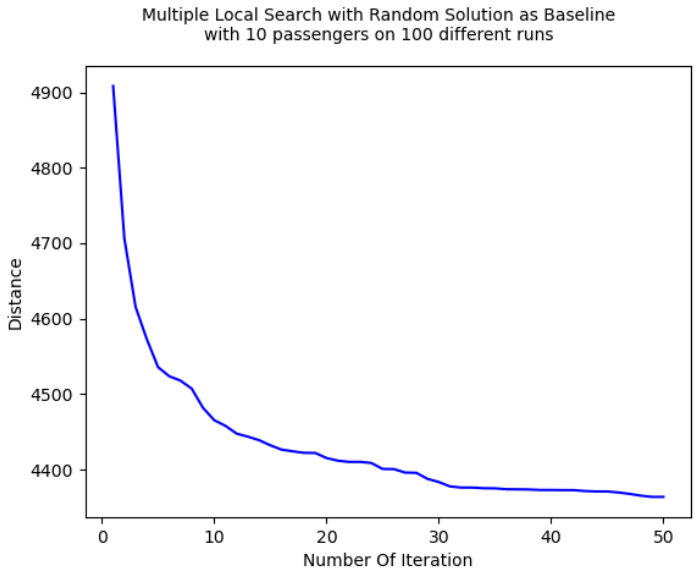
כדי לבדוק התכנסות של אלגוריתם ה Multiple Local Search בדקנו את הריצה שלו מעל פתרון החמדני, עם 10 נוסעים ומעל 100 ריצות שונות:



ציר ה X מתאר את מספר ההרצות השונות של אלגוריתם ה Local Search. ציר ה Y מתאר את אורך הפתרון **הממוצע** (מעל 100 הריצות השונות - seed שונים) לאחר x איטרציות של הרצה של ה Local Search.

ניתן לראות שהשיפור על ידי הגדלת מספר הריצות השונות של ה Local Search הוא לא מאוד משמעותי, וניתן להסביר זאת בכך שהפתרון החמדני הוא בדרך כלל פתרון שאין לו הרבה צורות שונות לשיפורים ולכן הגדלת מספר ההרצות של ה Local Search מעל אותו פתרון חמדני לא נותנת שונות מספיקה בין הריצות השונות של ה Local Search.

לעומת זאת, אם נריץ את אלגוריתם ה Multiple Local Search מעל פתרון רנדומי נקבל שיפור משמעותי יותר על ידי מספר גדול יותר של הרצות. הרצנו את האלגוריתם Multiple Local Search **מעל אותם דוגמאות** (seeds) כמו בריצה הקודמת, כאשר פתרון הבסיס הוא פתרון רנדומי:

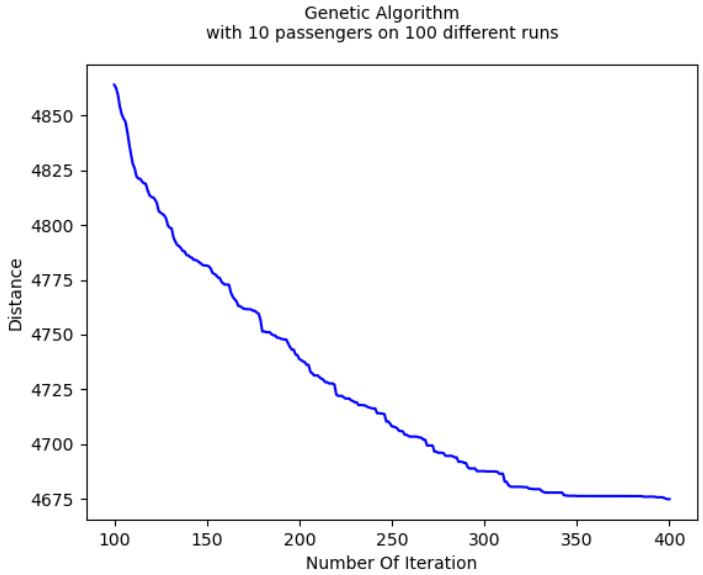


ניתן לראות שעבור מספר נמוך של איטרציות (עד 5 איטרציות) האלגוריתם מעל הפתרון הרנדומי משיג (בממוצע) תוצאות פחות טובות מאלו של הריצות מעל פתרון חמדני, אך לאחר מכן הוא מגיע בממוצע לפתרונות טובים משמעותית מאלו של האלגוריתם מעל הפתרון החמדני (פחות מ - 4400 עבור ריצות מעל פתרון רנדומלי, לעומת יותר מ 4600 עבור ריצות מעל פתרון חמדני).

כמו כן נראה שעבור הרצה על 10 נוסעים בתנאים שהצגנו, האלגוריתם מעל פתרון רנדומי מתכנס בערך לאחר 35 איטרציות.

התכנסות של Genetic Algorithm

הרצנו את האלגוריתם הגנטי על אותם 100 מקרים :



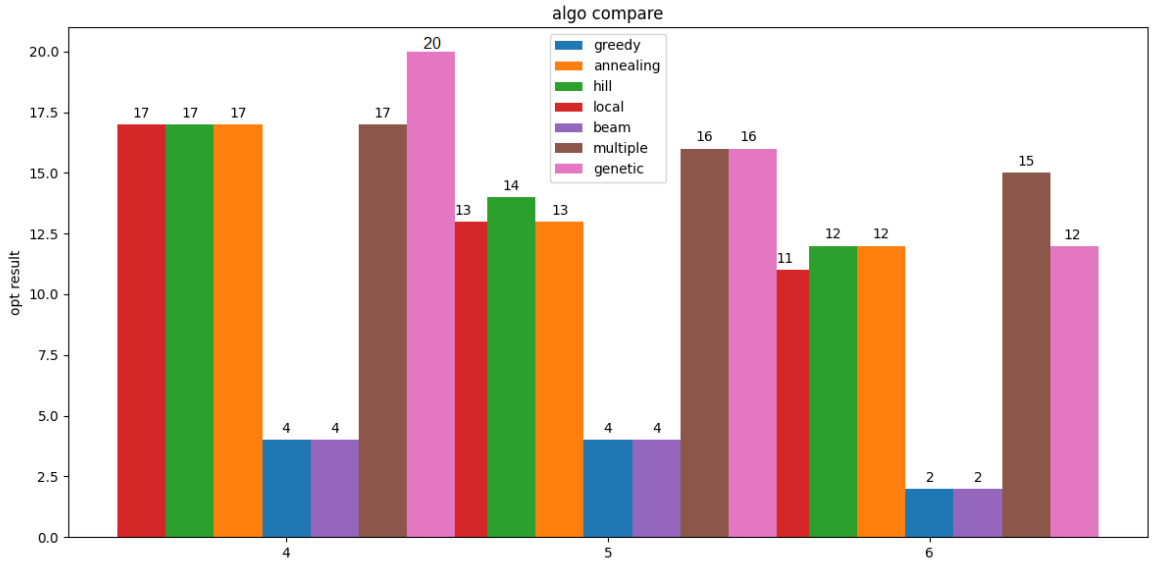
(הגרף מתחיל מהאיטרציה ה 100 מכיוון שבאיטרציות הראשונות הפתרון האופטימלי גדול בהרבה)

ניתן לראות התכנסות של האלגוריתם לאחר 350 איטרציות, ונראה כי התוצאות של האלגוריתם הגנטי פחות טובות מהתוצאות של אלגוריתם ה Multiple Local Search.

השוואה בין האלגוריתמים

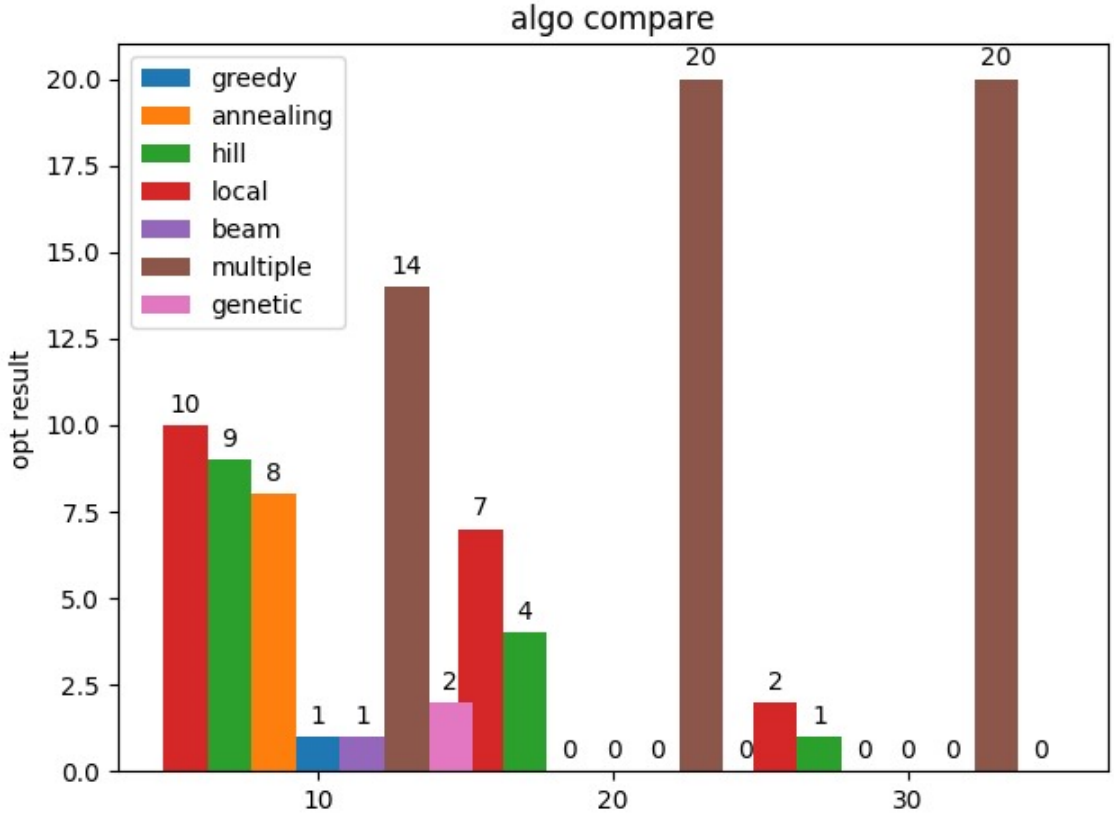
על מנת להשוות בין כל האלגוריתמים שיצרנו בשביל לפתור את הבעיה, הרצנו את כל האלגוריתמים במקביל על אותם קלטים. בשביל להגיע למסקנות מקיפות כמה שניתן, הרצנו את האלגוריתמים מספר פעמים על קלטים קטנים, כאלה שניתן להשוות בהם את התוצאה לפתרון האופטימלי שיתקבל מהרצת Brute Force, ומספר פעמים על קלטים גדולים יותר, כאלה שההשוואה בהם יכולה להתבצע רק בין האלגוריתמים לבין עצמם.

עבור מספר קטן של נוסעים (4,5,6 נוסעים) הרצנו עשרים ריצות שונות ובדקנו כמה פעמים כל אלגוריתם מגיע לתוצאה האופטימלית (לפי הפלט של אלגוריתם brute force).



ניתן לראות שאלגוריתם Multiple Local Search מגיע לפתרון האופטימלי פעמים רבות יותר מאשר שאר האלגוריתמים, עבור שישה נוסעים.

עבור מספר גדול יותר של נוסעים (10,20,30) הרצנו שוב עשרים ריצות שונות ובדקנו כמה פעמים כל אלגוריתם מגיע לתוצאה הטובה ביותר, (ביחס לאלגוריתמים האחרים):



ניתן לראות שאלגוריתם ה Multiple Local Search משיג את התוצאות הטובות ביותר.

הרצת הפרויקט

על מנת להריץ את התוכנית, יש להכניס ל-command line את הפרמטרים האופציונאליים הבאים:

- -n=N כאשר N מספר טבעי גדול ממש מ-0; מספר הנוסעים עליו רוצים להריץ את הקלט, ה-default זה 20.
- -s=Z כאשר Z מספר שלם; ה-seed שאיתו רוצים להריץ את התוכנית, ה-default זה 5.
- -a=ALGORITHM כאשר ALGORITHM זה האלגוריתם שאיתו רוצים להריץ את התוכנית, ה-default זה Greedy.
- -d; דגל שכאשר הוא נבחר מריץ ב-real time ריצה של האלגוריתם LocalSearch. כאשר משתמשים ב-d אין משמעות לבחירת אלגוריתם.

מספר הערות נוספות:

- כאשר מריצים את התוכנית עם מספר קטן של נוסעים, עד 6 נוסעים, התוכנית מריצה אוטומטית את האלגוריתם BruteForce. זאת, מכיוון שהוא מחזיר את הפתרון האופטימלי בזמן קצר מאוד.
- לא מומלץ להריץ את האלגוריתם BruteForce על יותר מ-9 נוסעים. גם ריצה של 8 או 9 נוסעים בסבירות גבוהה תיקח מספר שעות, אם כי עם קצת היא יכולה להיות קצרה משמעותית (בשל ה-pruning שהאלגוריתם מבצע).
- מספר דוגמאות לשורות הרצה:

- python3 main.py
- python3 main.py -n=13 -s=17 -a=HillClimbing
- python3 main.py -n=23 -d

```
Taxi Driver Problem Solver

optional arguments:
  -h, --help            show this help message and exit
  -n NUM_OF_PASSENGERS, --num_of_passengers NUM_OF_PASSENGERS
                        Number of passengers, must be a positive int.
  -s RANDOM_SEED, --random_seed RANDOM_SEED
                        The seed for the passengers stations.
  -a {Greedy,LocalSearch,HillClimbing,SimulatedAnnealing,BeamLocalSearch,Multiple,Genetic,BruteForce}
                        choose which algorithm to run.
  -d, --demo            Run demo on Local Search over random solution.
```