מעבדה בבינה מלאכותית

דו"ח מעבדה 3

שמות:

עוביידה חטיב, 201278066

אסיל נחאס, 212245096

סביבת ההרצה והכלים בהם נעשה שימוש

- המפענח שהשתמשנו בו: GNU bash 3.2.57
- גרסת הפייתון: 3.9.6
- חבילות לא סטנדרטיות שהשתמשנו בהם: matplotlib 3.9.4

הרצת קובץ הפייתון

קוד הפייתון רץ באמצעות הפקודה הבאה:

```
python lab3.py <time_limit> <problem> <file_path>
```

```
python lab3.py <time_limit> <problem>
```

כאשר:

- time_limit: זמן ההרצה המקסימלי. במקרה שזמן ההרצה מגיע לסף הזה ההרצה מפסיקה והגינום של הפרט בעל הפיטניס הטוב ביותר שהתקבל עד אז יודפס.
- problem: סוג הבעיה מבין שתי הבעיות בהם הקוד שלנו מטפל - CVRP לבעיית ניתוב הרכיבים, ו- ACKLEY לבעיית פונקציית Ackley.
- file_path: הנתיב לקובץ הקלט של הבעיה. ארגומנט זה נדרש רק עבור בעיית ניתוב הרכבים.

דוגמאות לפקודות הרצה חוקיות:

```
python lab3.py 300 CVRP P-n16-k8.vrp.txt
```

```
python lab3.py 500 ACKLEY
```

הערה: הקוד מריץ את האלגוריתמים השונים אחד אחרי השני, תוך שהוא מציג את התוצאות לכל אחד, ובסוף מציג את התוצאות של ההשוואה בין האלגוריתמים השונים. על מנת להריץ אלגוריתם אחד בלבד מבין האלגוריתמים נא לבחור אותו כערך של המשתנה הגלובלי (ALGORITHM) המופיע בתחילת הקוד, בנוסף, להחליף בין שתי השורות האחרונות בקוד, כך ש- $run\_full\_comparison(input\_file)$ תהפוך להערה, ותבוטל ההערה מהשורה $main(input\_file)$.

הרצת קובץ ה- EXE

קוד ה- EXE רץ באמצעות שתי הפקודה הבאה:

lab3.exe <time_limit> <problem_type> <file_path>

כאשר הארגומנטים שהוא מקבל הם אותם ארגומנטים שהוסבר לגבם בהסבר לגבי הרצת קובץ הפייתון.

ייצוג הפרטים, האוכלוסייה, והאלגוריתמים:

- **בעיית ה- CVRP:** הפרט (הפתרון) מיוצג ע"י מופע של המחלקה CVRPIndividual המכילה 3 שדות: מסלולים\רכבים, הפיטניס\העלות, והאוכלוסייה (אוסף הפתרונות) אליה משתייך הפתרון. שדה המסלולים הוא רשימה המכילה רשימות המייצגות מסלולים ומכילה את האינדקסים של הערים במסלול לפי סדר הביקור בהם. בנוסף המחלקה מכילה פונקציה המחשבת את הפיטניס של הפתרון, והיא נקראת במקומות שונים בתוך האלגוריתמים. הפיטניס מחושב כאורך המסלולים הכולל כאשר כל אחד מהמסלולים מתחיל מהמחסן וחוזר אליו. המרחק בין שני ערים הוא המרחק האיקלידי בין הקואורדינטות של שתי הערים.
  האוכלוסייה מיוצגת ע"י מופיע של המחלקה CVRPPopulation, אשר מכילה בין היתר את השדות: רשימת הפתרונות, רשימות של הפיטניס הטוב ביותר, והפיטניס הממוצע בכל איטרציה. בנוסף לכך היא מכילה פרטים המחולצים מקובץ הקלט כגון תכולת הרכבים ומספרים, קואורדינטות הערים, הביקוש של הערים, מטריצה דו-ממדית השומרת את המרחקים בין הערים השונים ומחושבות בעת אתחול המופע ומשמשת חישובים שונים במהלך ריצת האלגוריתם.
- **בעיית פונקציית Ackley**: הפתרון\הפרט מיוצג ע"י מופע של המחלקה AckleyIndividual, המכילה שדות של רשימת מקדמי הממדים של הפונקציה,

הפיטניס, ומופע האוכלוסייה אליה משתייך הפתרון. הפיטניס מחושב על פי הנוסחה הנתונה. האוכלוסייה מיוצגת ע"י מופע של המחלקה AckleyPopulation המכילה בנוסף לרשימות הפתרונות, הפיטניס הטוב ביותר והפיטניס הממוצע בכל איטרציה, את הערכים הנתונים בסעיף של הפרמטרים a,b,c, הממד, והחסמים התחתון והעליון.

בנוסף כל אחד מהאלגוריתמים מיוצג ע"י מחלקה נפרדת. המכילה בתוכה שדה לאוכלוסייה עליה תעבוד, פונקציה ראשית הנקראת solve המריצה את האלגוריתם ומדפיסה את הפלט הסופי שלו, ופונקציות עזר אשר נקראות ע"י solve. כמו כן, פונקציות רבות שימוש, המשמשות יותר מאלגוריתם אחד, כמו הפונקציה המדפיסה את התוצאה הסופית, נמצאות מחוץ למחלקות.

ערכי הפרמטרים הייחודיים של האלגוריתמים השונים דוגמת הטמפרטורה ההתחלתית, אחוז ההתאדות (evaporation rate), מספר האדיים וכו' נבחרו ע"י המכניזם הבא: עבור כל ערך אפשרי, האלגוריתם הורץ 5 פעמים על 5 אוכלוסיות התחלתיות שונות תוך שימוש בערך הזה, והערך שנתן את התוצאות הכי טובות מבחינת הפיטניס הטוב ביותר\ממוצע הפיטניסים הטוב ביותר הוא שנבחר. הרבה פעמים שני פרמטרים נבחנו ביחד (כמו אחוז ההגירה ומרווח ההגירה באלגוריתם הגנטי) וההשוואה הייתה בין הקומבינציות של זוגות הערכים של שני הפרמטרים. בחלק מהפעמים התוצאות היו שונות עבור שתי הבעיות ולכן לכל בעיה אומצה האופציה הטובה בשבילה. כמו כן, לפעמים האופציה שנתנה את הפיטניס הטוב ביותר הייתה שונה מזאת שהניבה את ממוצע הפיטניסים הטוב ביותר והיה צורך לבחור אחד מהם, לא הייתה לנו דרך ספציפית להתמודדות עם מקרים כאלה ובחירת אחד מבין השניים הייתה תלוית סיטואציה. בפרקים שונים בהמשך אנחנו מציגים דיאגרמת עמודות של התוצאות שהתקבלו עבור ערכי הפרמטרים הרלוונטיים. יש לציין שההרצות הנוגעות להשוואות נעשו תוך שימוש בקובץ A-n80-k10 מפני שלדעתנו הוא בגודל ובמורכבות שמאפשרות הבחנה בהבדלים.

מלבד האלגוריתמים Multi-Stage Heuristics ו- Branch and Bound, כל האלגוריתמים רצים עד שאחד משלושת התנאים מתקיים: התכנסות לאופטימום לוקאלי, סיום פרק הזמן המקסימלי הניתן לאלגוריתם כארגומנט, הגעה למספר האיטרציות המקסימלי. כמו כן, מלבד ה- ACO שהוא חלק מה- ILS, העובד אך ורק לבעיית ניתוב הרכבים, כל האלגוריתמים תקפים לשתי הבעיות, לפעמים עם מימושים שונים כתלות בבעיה, שינויים שיכולים להיות קטנים דוגמת שורה שונה לכל אלגוריתם בתוך אותה פונקציה ועד שינויים גדולים כגון פונקציה נפרדת לכל אלגוריתם שמממשת את הדברים תוך שימוש בלוגיקה שונה.

סעיף 1

האלגוריתם מומש ע"י המחלקה MSHeuristicsAlgorithm הפועלת בשני שלבים עיקריים: יצירת אוכלוסייה (אוסף פתרונות) התחלתיים, ואופטימיזציה של הפרטים.

עבור בעיית CVRP, השלב הראשון מתבצע ע"י השימוש באלגוריתם הבנוי על בסיס אלגוריתם K-means אך מותאם לבעיה שלנו, כאשר k הוא מספר הרכבים בבעיה. הערים באלגוריתם שלנו מחולקים למספר של קבוצות שיכול להגיע עד k, כאשר כל קבוצה מייצגת רכב\מסלול, ועל סמך כך הערים הנמצאים באותה קבוצה נכללים באותו מסלול. האלגוריתם משייך את העיר לקבוצה מסוימת על בסיס המרחק שלה ממרכז הקבוצה תוך הלקיחה בחשבון של קיבולת הרכב המקסימלית. האלגוריתם מתחיל במסלול אחד המגיע לעיר אחת הנבחרת באופן אקראי, ולכל עיר הוא משייך אותה למסלול קיים או למסלול חדש שבו יהיה את העיר בלבד. ההחלטה על פתיחת מסלול חדש לעיר מסוימת מתקבלת אם המרחק של העיר מהמחסן קטן מכל מרחק ממרכז של אחת הקבוצות (המסלולים) הקיימות אשר הקיבולת שלהם מאפשרת הכנסת הרכב, ובתנאי שמספר המסלול עוד לא הגיע ל- k.
האתחול מתבסס על ההיוריסטיקה שערים קרובות אחת לשנייה כדאי וחסכני שיהיו באותו מסלול. בכדי להתמודד עם מצבים קיצוניים בהם הפונקציה לא מצליחה לייצר פרטים, אם 3 ניסיונות ליצירת פרט נכשלים, פונקציה אחרת המשתמשת באלגוריתם First-Fit וממלאת את המסלולים על פי הביקוש שלהם נקראת, הפונקציה מהווה חגורת ביטחון ומטרתה למנוע מצב בו האלגוריתם לא מצליח לאתחל אוכלוסייה, אולם הפתרונות שהיא מייצרת רנדומליים, והתקווה היא שהשלבים הבאים באלגוריתם ישפרו את אותם פתרונות.

אחרי קביעת הערים המשתתפות בכל מסלול, מטרת השלב השני של האלגוריתם היא לסדר את הערים במסלול בצורה הכי אופטימלית, ועל מנת לעשות זאת הוא משתמש בהיוריסטיקה של "הלקוח הקרוב ביותר" כך שבכל שלב הבחר הלקוח הקרוב ביותר ללקוח הנוכחי. הלקוח הראשון הוא הכי קרוב למחסן מבין הערים באותו מסלול.

מנגד, **עבור בעיית פונקציית ה- Ackley**, האתחול הוא אקראי. הוא נבחר להיות כזה בשל התוצאות הטובות שהוא הניב לעומת הדרך האחרת שנבחנה שדאגה לגיוון וחילקה את המרחב לתתי מרחב כמספר הפרטים\הפתרונות ובוחרת פרט מכל תת-מרחב באופן רנדומלי. השלב השני שואף לשפר את הפתרונות ולשם כך נעשה שימוש בחיפוש לוקאלי ע"י יצירת 1000 עותקים של הווקטור ולהוסיף לכל ממד בכל וקטור רעש גאוסיאני, ולבסוף הטוב מבין הווקטורים מחליף את הווקטור הנוכחי.

הסבר לגבי ההיורסטיקות בהם השתמשנו בסעיף, ביחד עם ניתוח הסיבוכיות שלהן, נכללים בהסבר לגבי כלל ההיורסטיקות שנעשה בהם שימוש במעבדה בחלק האחרון של הדו"ח.

סעיף 2

האלגוריתם ממומש ע"י הפונקציה ILSAlgorithm, אשר מאתחלת אוכלוסייה (אוסף פתרונות) באותה שיטה שהוסברה בסעיף הקודם (ע"י האלגוריתם של ה- K-means במקרה של בעיית ה- CVRP, ובאופן אקראי במקרה של בעיית Ackley). האלגוריתם לאחר מכן מוצא בכל איטרציה שכן לכל אחד מהפתרונות, ומאמץ אותו במקרה שהוא טוב מהנוכחי, אחרת הוא פועל בהתאם למיתא היוריסטיקה שנבחרה. שיטת ה- ACO פועלת באופן שונה ועליה יוסבר בנפרד בהמשך.

מציאת השכן מתבצעת ע"י הפונקציה find_neighbor הפועלת תוך שימוש בשיטות שונות לכל אחת משתי הבעיות:

בעיית ה- CVRP משתמשת בשיטות הבאות למציאת שכן:

- 2-opt: בוחרת קטע אקראי מתוך מסלול אקראי והופכת אותו.
- relocate: בוחרת עיר באופן אקראי, מוציאה אותה מהמסלול אליו היא שייכת ומכניסה אותה למסלול אקראי אחר. אם המסלול החדש עדיין עומד במגבלת התכולה של הרכב היא מחזירה את השכן.
- reposition: בוחרת עיר ומשנה את המיקום שלה באופן אקראי בתוך המסלול אליו היא שייכת.
- swap: מחליפה בין שתי ערים באופן אקראי, נבדקת תקינות המסלולים לאחר מכן, ואם הם עומדים במגבלת הרכבים השכן מוחזר.
- shuffle: לוקחת מסלול אקראי ומערבבת את הסדר של הערים בו.

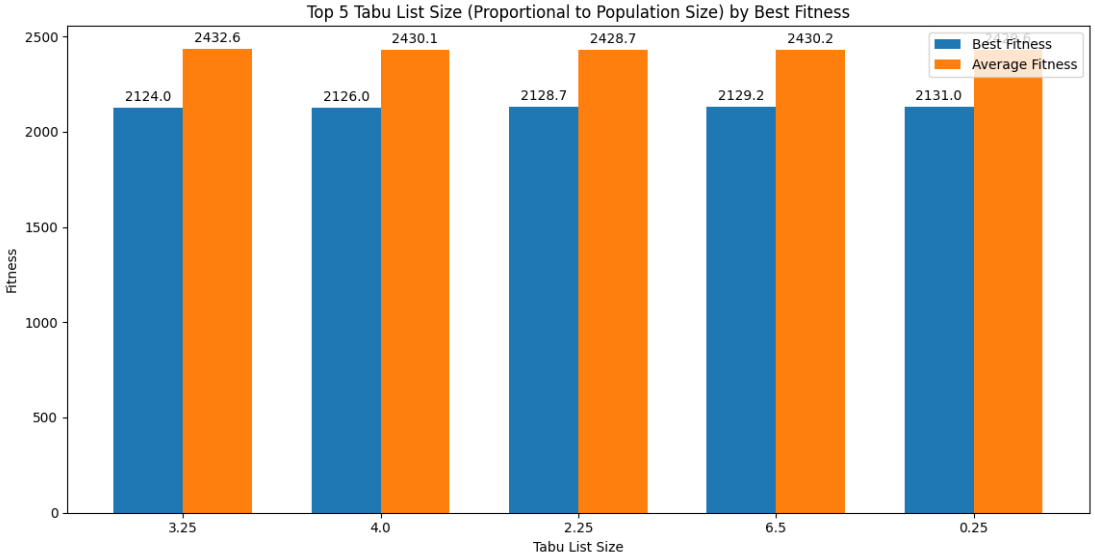
עבור בעיית Ackley, השימוש היה בשיטות הבאות:

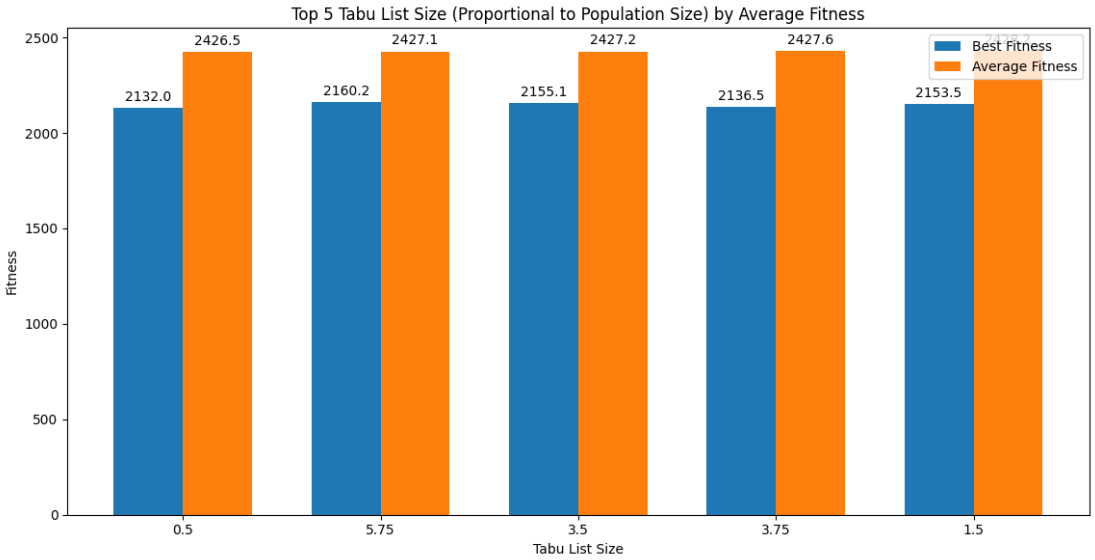
- shift one: הוספת רעש, שהוא ערך שבין 0.1- ל- 0.1, לממד אקראי של הווקטור.
- shift all: הוספת רעש, וקטור שהערכים שלו בין 0.05- ל- 0.05, לווקטור שנבחר באופן אקראי (כל הממדים מושפעים).
- set random: שינוי הערך של אחד הממדים בערך רנדומלי.

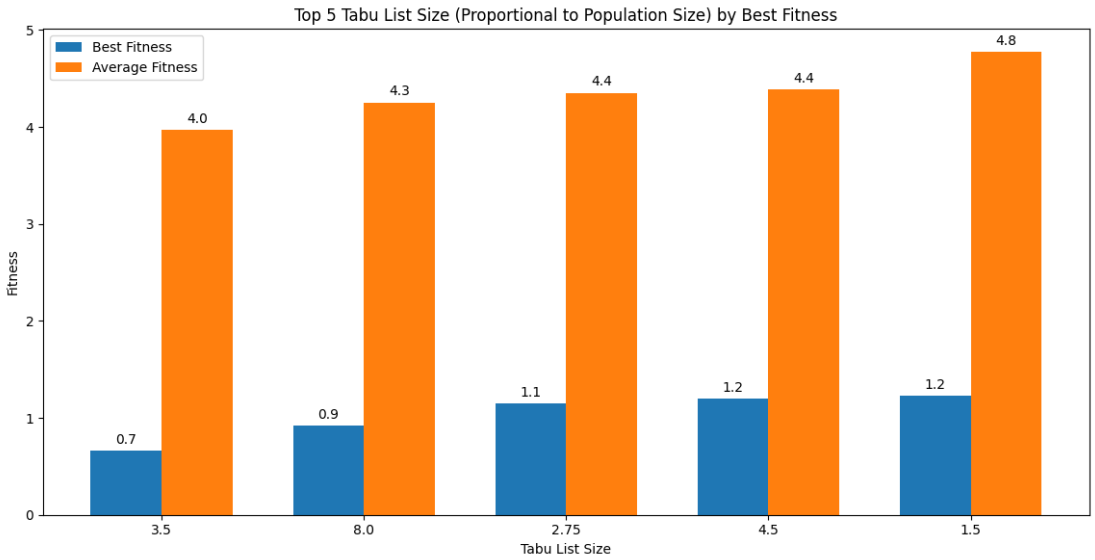
במקרה שהמיטה היוריסטיקה היא Tabu Search, האלגוריתם מאתחל רשימת טאבו בגודל פי 3.5 מגודל האוכלוסייה, ומתחזק אותה במהלך האלגוריתם, בכך שהוא מוסיף פתרון חדש לסופה, ומוציא פתרון מהתחלתה במקרה שהיא מלאה. כמו כן, ובשל צרכי יעילות, נעשה שימוש במבנה נתונים מסוג set על מנת למצוא אם פתרון מסוים קיים או שלא והוא מעודכן בהתאם לרשימה, ויחד מרכיבים את טבלת הטאבו. לבחירת גודל הטבלה נבחנו כל הגדלים

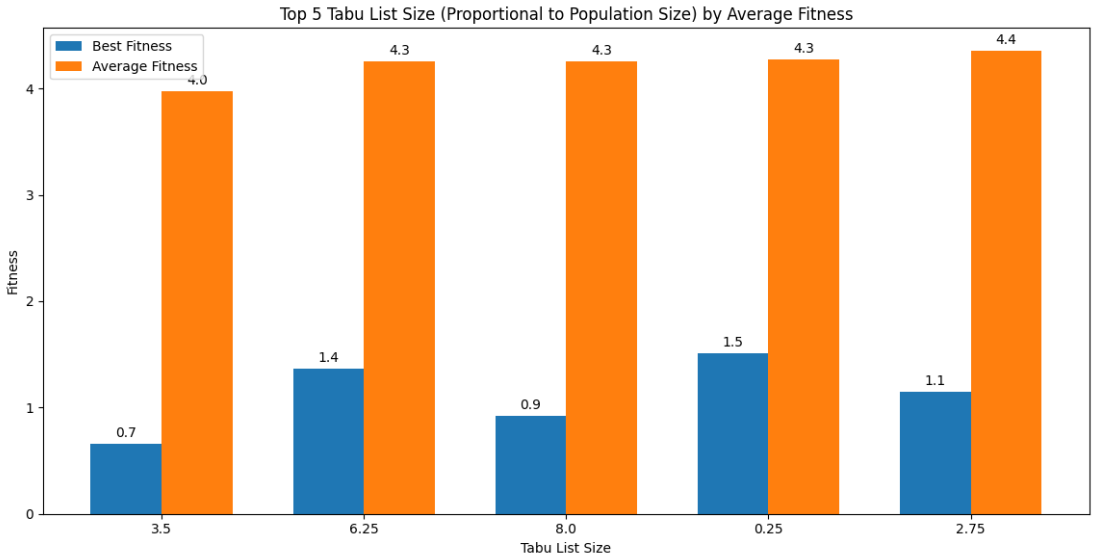
האפשריים בין 0.25*גודל האוכלוסייה ל- 8* גודל האוכלוסייה בקפיצות של 0.25. התוצאות במקרה ה- CVRP (שתי הדיאגרמות הראשונות) היו מאד קרובות וההבדלים לא היו משמעותיים. מנגד, כן נצפו הבדלים במקרה של בעיית Ackley (שתי הדיאגרמות האחרונות). הערך נבחר להיות 3.5* גודל האוכלוסייה נבחר בשל היותו מופיע ב- 3 מ- 4 הדיאגרמות, ובמיוחד שהוא נתן התוצאות הכי טובות במקרה של בעיית Ackley.

האלגוריתם מחלץ לכל פרט את השכנים שלו באמצעות כל אחת מהשיטות ובוחר מתוכם את הכי טוב בתנאי שהוא לא נמצא בתוך הטבלה.

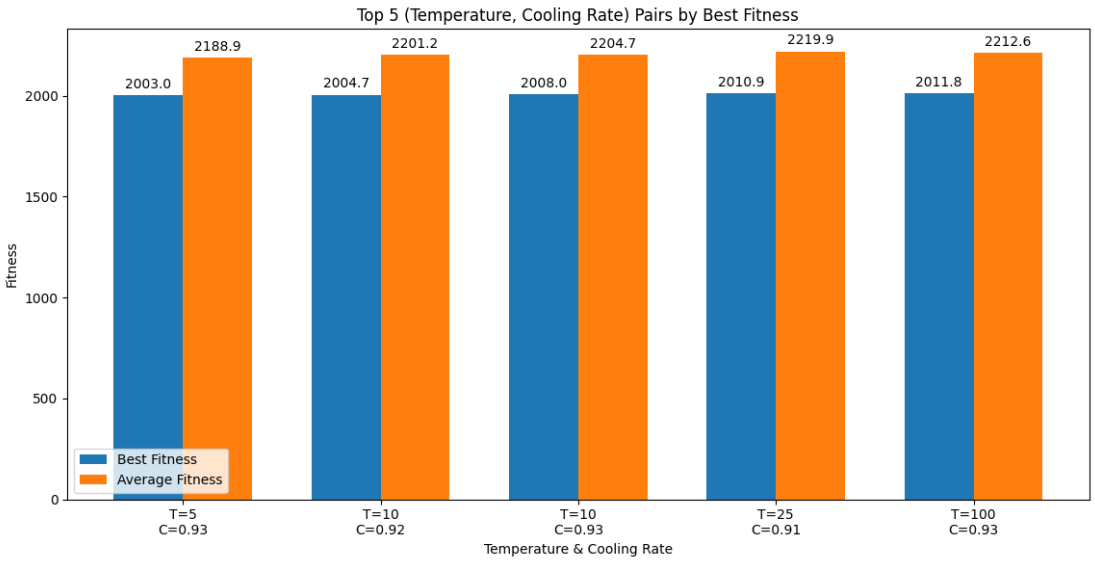


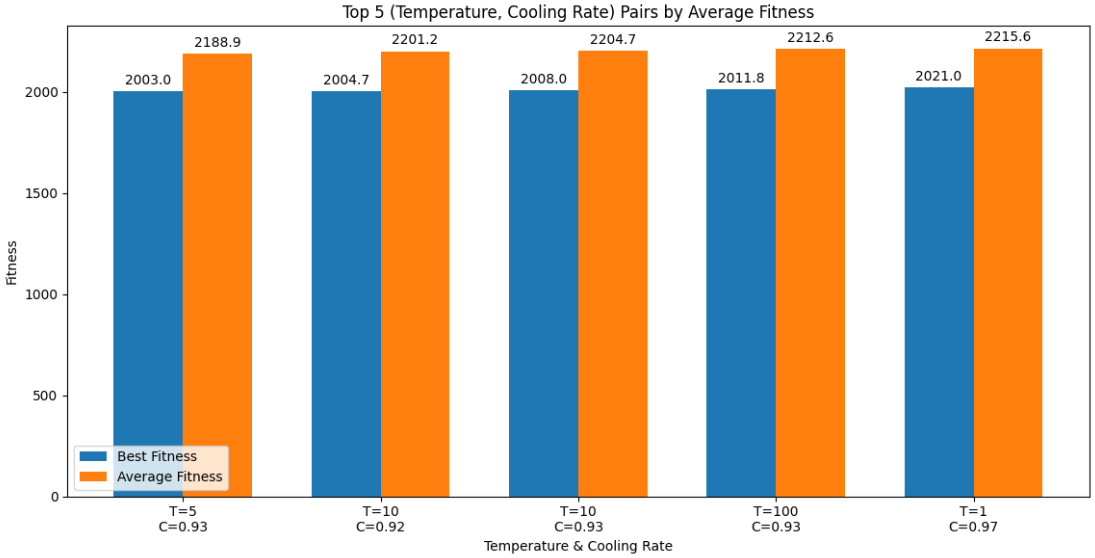


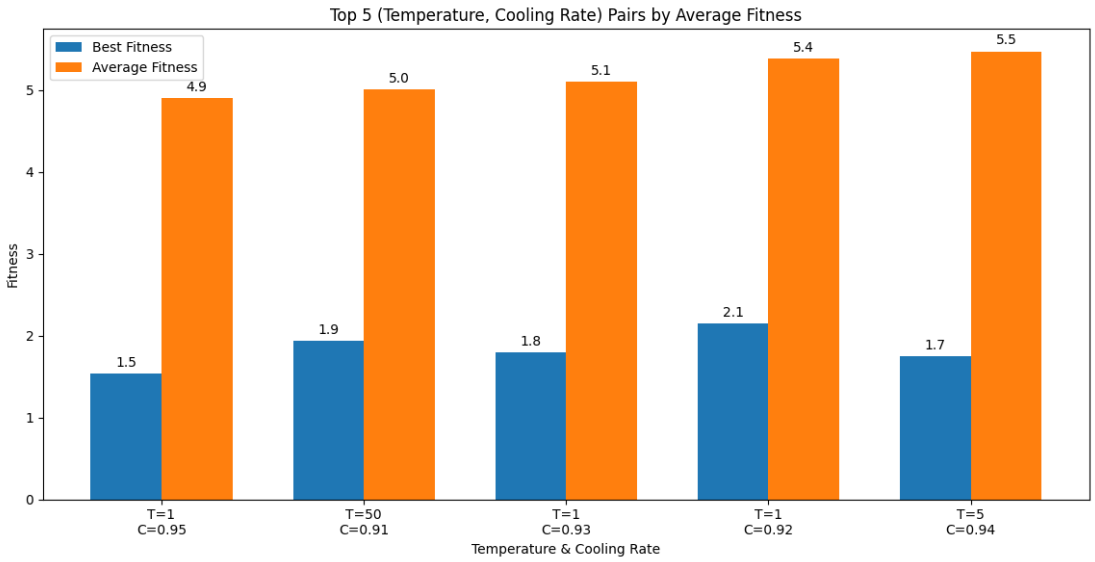


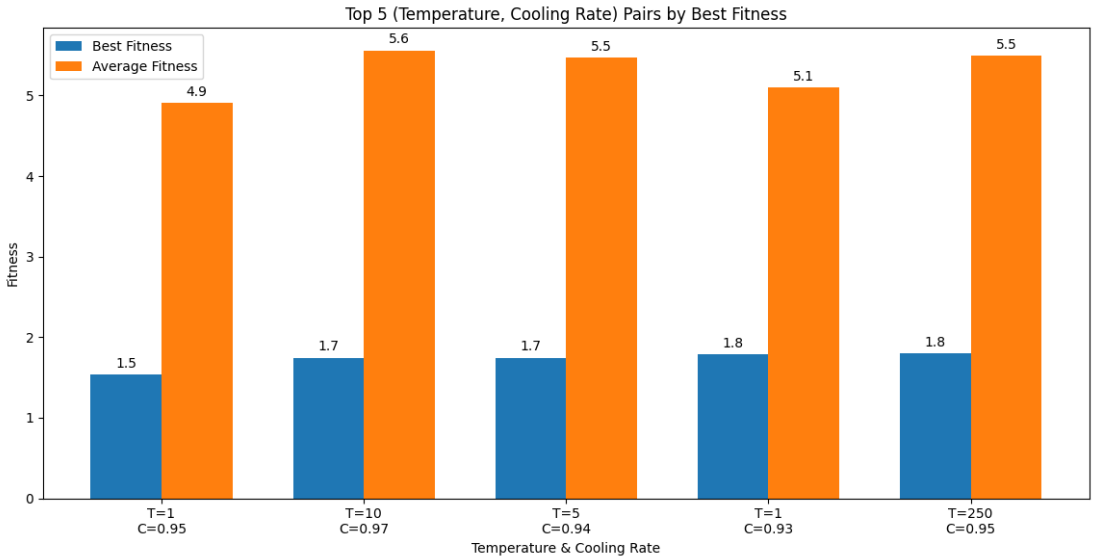


ה- Simulated Annealing במקום זאת בוחרת שיטת שכנות באופן אקראי ומחלצת את השכן על פיה, אם הוא יותר טוב מהפרט הנוכחי היא מאמצת אותו במקומו, אחרת, היא מאמצת אותו בהסתברות שהשיטה Simulated Annealing קובעת. הערכים של ה- Temperature ההתחלתי וה- Cooling rate של השיטה נבחרו להיות שונים בהתאם לבעיה על סמך תוצאות ההשוואה המצורפות להלן:

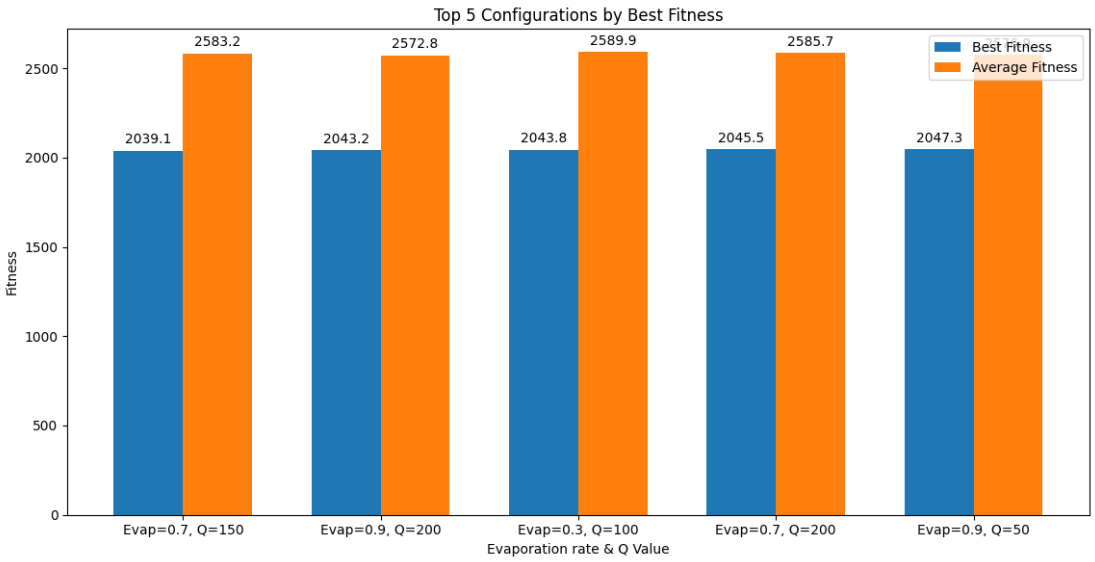

Top 5 (Temperature, Cooling Rate) Pairs by Best Fitness
Fitness
0
500
1000
1500
2000
2003.0
2188.9
2004.7
2201.2
2008.0
2204.7
2010.9
2219.9
2011.8
2212.6
Best Fitness
Average Fitness
T=5
C=0.93
T=10
C=0.92
T=10
C=0.93
T=25
C=0.91
T=100
C=0.93
Temperature & Cooling Rate

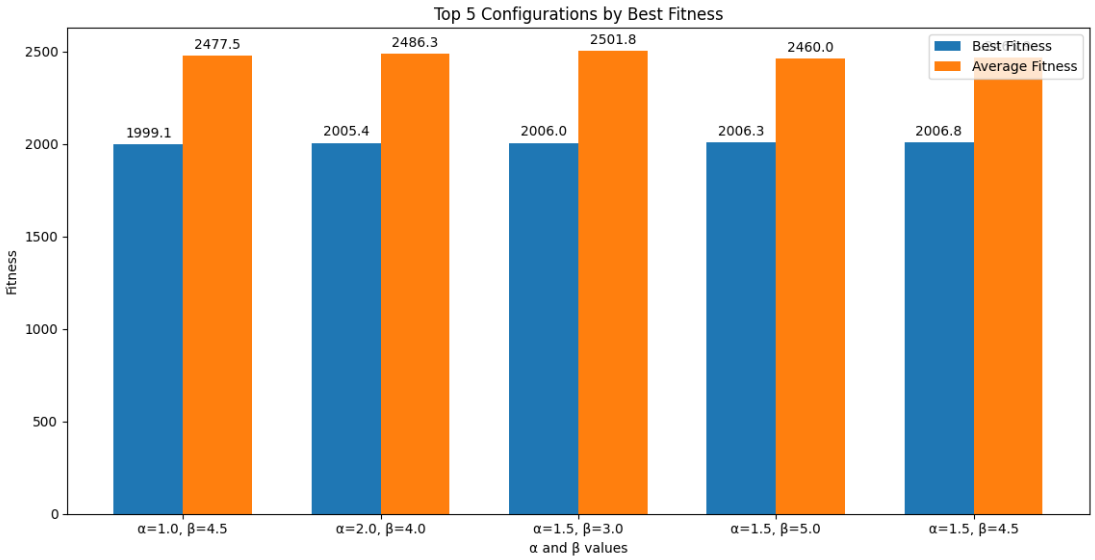

Top 5 (Temperature, Cooling Rate) Pairs by Average Fitness
Fitness
0
500
1000
1500
2000
2003.0
2188.9
2004.7
2201.2
2008.0
2204.7
2011.8
2212.6
2021.0
2215.6
Best Fitness
Average Fitness
T=5
C=0.93
T=10
C=0.92
T=10
C=0.93
T=100
C=0.93
T=1
C=0.97
Temperature & Cooling Rate





היוריסטיקת Discrete Ant Colony Optimization פעולת באופן שונה. כאשר היא מאתחלת את ה- Pheromone בין כל שתי ערים להיות 1.0 ומעדכנת אותם אחרי כל איטרציה. בכל איטרציה היא בונה מסלולים מחדש ע"י בחירת עיר התחלתית במסלול באופן אקראי וכל עיר אחרת מתווספת בהתאם לאם עם הוספתה עדיין המסלול יעמוד בתנאי התכולה של הרכב ובאופן יחסי לערך ה- Pheromone שלה מתוך ה- Pheromones של שאר הערים העומדים בתנאי בהתאם לנוסחה. בחירת ערכי הפרמטרים השונים בהם ההיורסטיקה משתמשת (Evaporation rate, Q, α, β) היו בהתאם להשוואה שנעשתה ותוצאותיה מצורפות להלן:





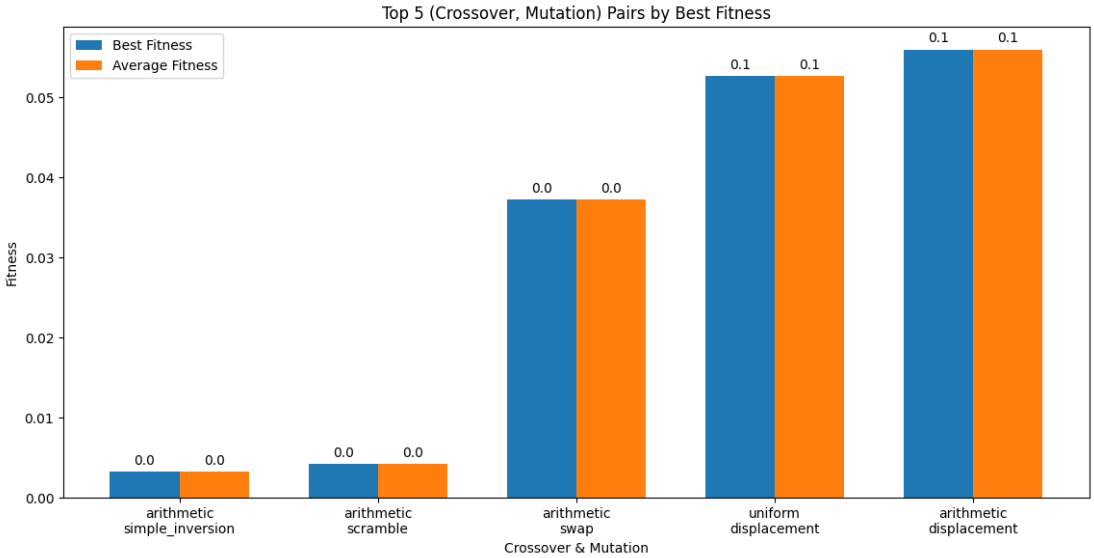
סעיף 3

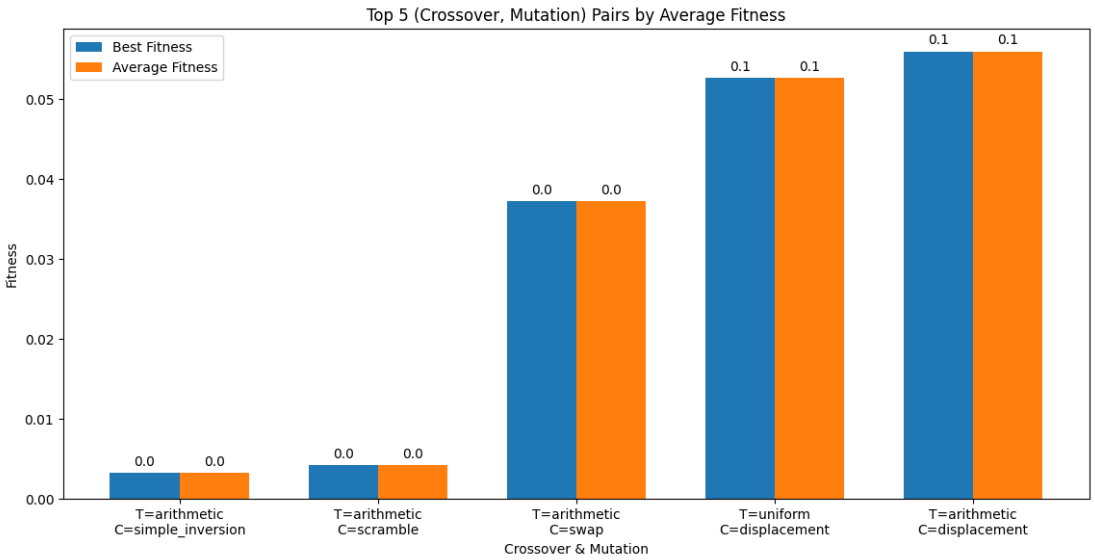
האלגוריתם ממומש באמצעות המחלקה GAAlgorithm. היא מתחילה בכך שהיא מאתחלת את האוכלוסייה באותה דרך אתחול שהוסברה מקודם בסעיף א', ומחלקת את האוכלוסייה לאיים שוות בגודלם, בכל שלב ושלב כל אי מתנהג כאוכלוסייה בשל עצמה בכך שהיא שומרת את האליטה לשלב הבא, ויוצרת פרטים חדשים כשיחלוף של שני הורים, שיכולים לעבור מוטציה בהסתברות שנקבעה להיות 0.25. במרווח של מספר קבוע של דורות מתבצעת הגירה של אחוז מסוים מאוכלוסיית כל אי לאי שבא אחריה. המהגרים נלקחים כך שחצי מהם הם הפרטים הטובים ביותר באי שלהם, והחצי האחר באופן רנדומלי.

רוב הפונקציות לקוחות מהמימושים שלנו במעבדות הראשונה והשנייה. עבור בעיית ה-CVRP ובשל היותה ניתנת לייצוג ע"י פרמוטציה, בדומה לבעיה שהתמודדנו איתה במעבדה הקודמת, שיטות בחירת ההורים, השיחלוף, והמוטציה, כמו גם הערכים של המשתנים הקשורים בשיטות הללו נבחרו להיות כאלה שהראו את התוצאות הכי טובות במעבדה הקודמת. לעומת זאת, בבעיית Ackley, ובשל היותה לא דומה לבעיות שהתמודדנו איתם בעבר היה צורך בלהשוות את הביצועים של הקונפיגרציות השונות על הקלט שלה. בעיית Ackley השתמשה באותם שיטות מוטציה (displacement, swap, insertion, simple inversion, inversion, scramble), אולם השתמשה רק באחת משיטות השיחלוף של בעיית CVRP שהיא ה- Order Crossover. השיטות האחרות לדעתנו לא תואמות לשימוש של הבעיה. בשל כך הוספנו שתי שיטות שיחלוף נוספות הייחודיות לבעיית Ackley והם:

- Arithmetic Crossover: השיטה בוחרת ערך α אקראי בטווח (0,1), ויוצרת פרט חדש כצירוף של α* ההורה הראשון ו- (1-α)*ההורה השני.
- Uniform Crossover: השיטה בונה וקטור חדש כאשר כל ערך של ממד בו לקוח מאחד ההורים באופן אקראי.

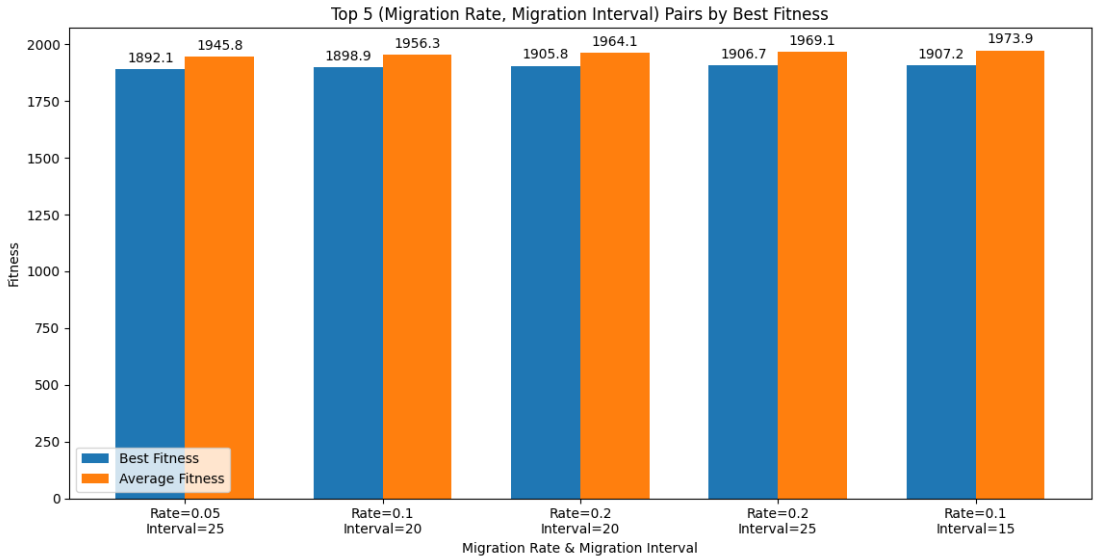
18 הזוגות (3 שיטות השיחלוף ו- 6 שיטות המוטציות) נבחנו עבור בעיית Ackley ועל סמך התוצאות נבחרו הזוג (Arithmetic, Simple Inversion) להיות שיטות השיחלוף והמוטציה בהם האלגוריתם משתמש במקרה של הבעיה:

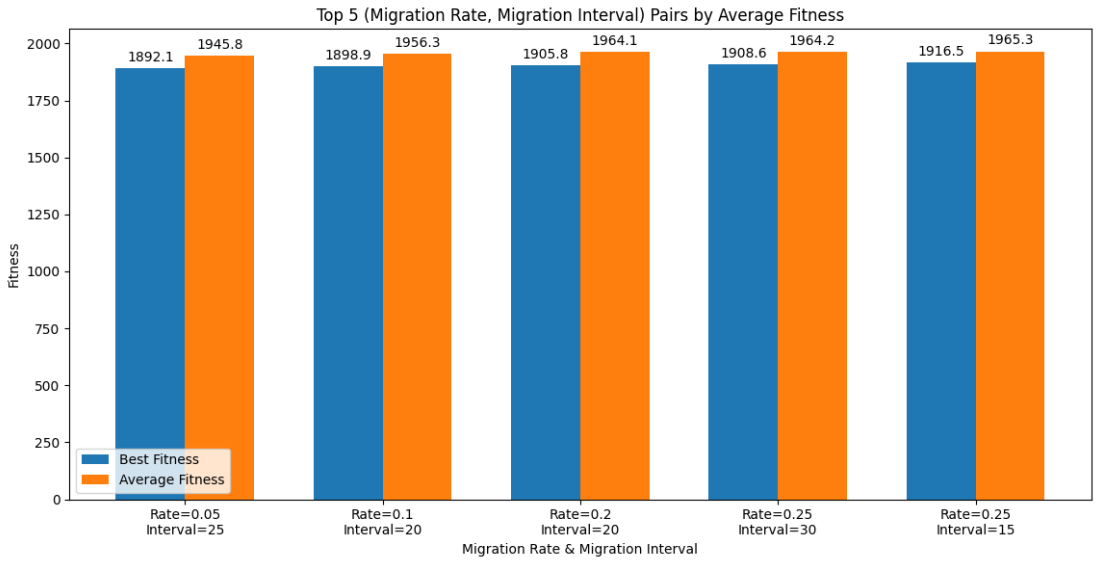




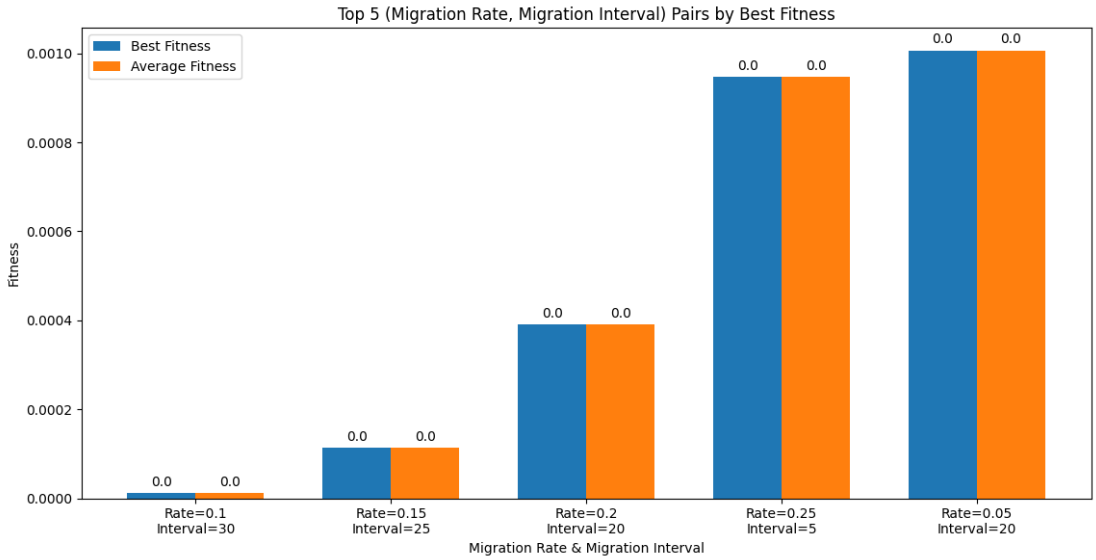
משתנים שהיה חשוב לעשות להם Tuning לשתי הבעיות הם המשתנים הקשורים במודל האיים: מספר האיים, אחוז ההגירה, מרווח ההגירה, ועל כן היה צורך בלהשוות ביניהם. התוצאות של ההשוואה בין זוגות שונים של **אחוז ההגירה ומרווח ההגירה** הראו את התוצאות המצורפות למטה עבור כל אחד מהבעיות, כאשר הערכים שנבחנו עבור אחוז ההגירה היו 0.05-0.25 בקפיצות של 0.05, ועבור מרווח ההגירה 5-30 דורות בקפיצות של 5.

עבור בעיית CVRP:

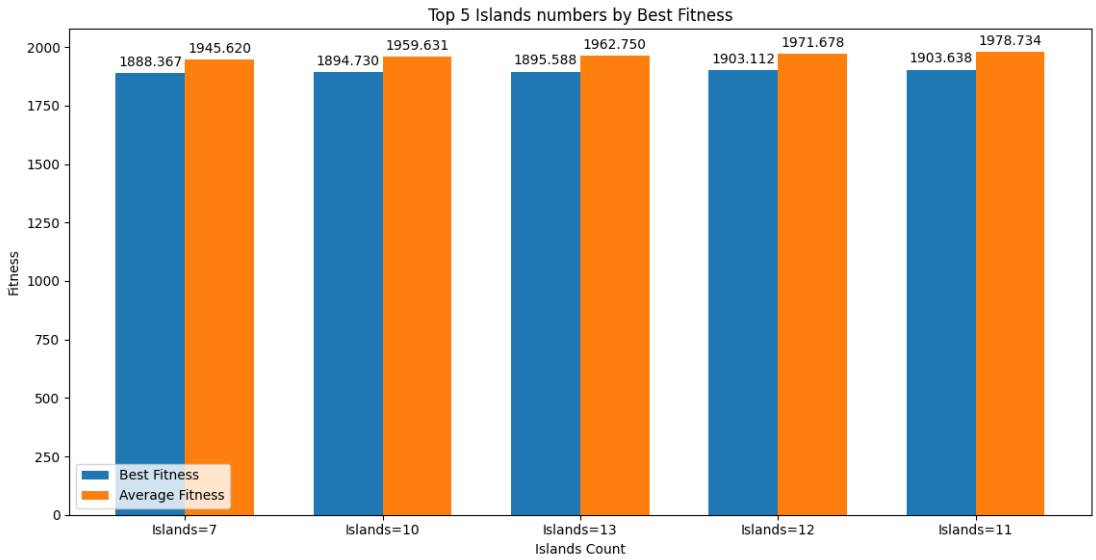


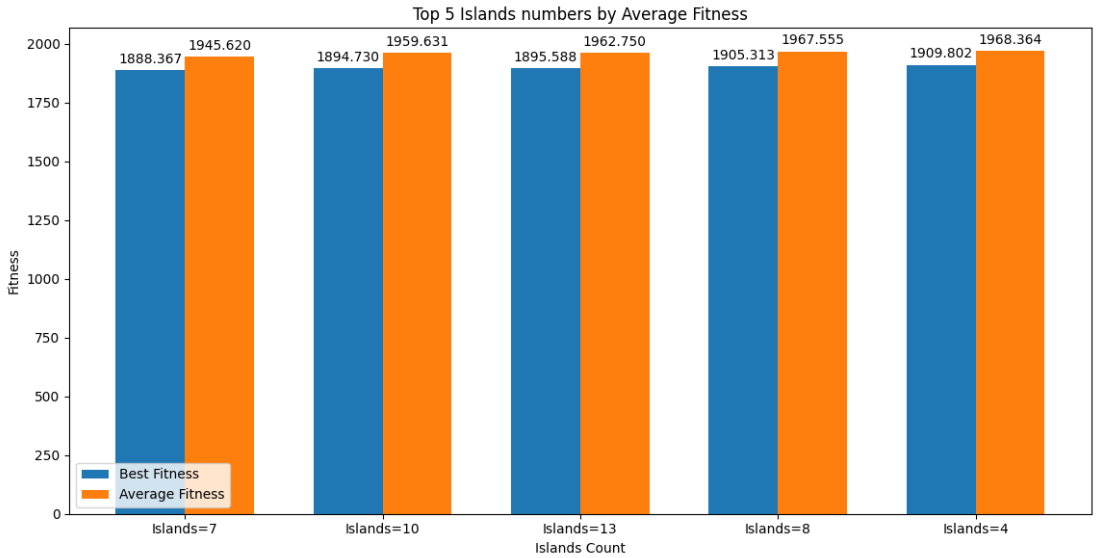


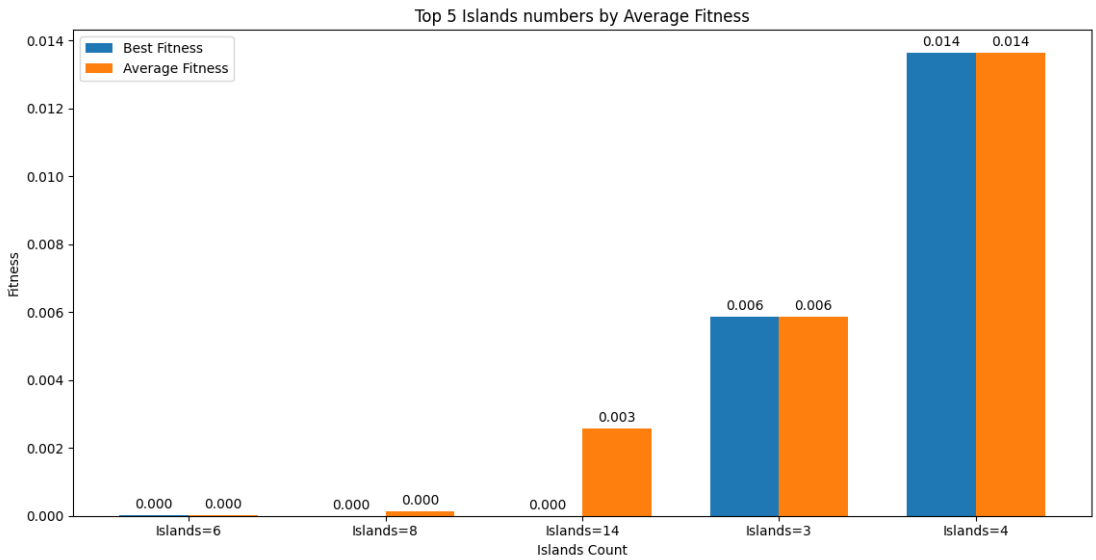
עבור בעיית Ackley:



ההשוואה בין מספר האיים לקחה בחשבון את מספר האיים שבין 2 עד 15 והתוצאות לשתי הבעיות היו כמוצג הדיאגרמות למטה כאשר שתי הראשונות הן לבעיית ה- CVRP והאחרונה לבעיית Ackley. בבעיית Ackley ובשל כך שבהינתן זמן מספיק גדול התוצאות הכי טובות מגיעות לפיטניס קטן מדי, שלא מאפשר להשוות בין האופציות השונות, הגבלנו את הזמן ל- 45 שניות ובכך פקטור הביצוע תחת אילוץ זמן קיבל משקל משמעותי בבחירת הערכים. על סמך התוצאות קבענו את מספר האיים להיות 7 במקרה שהבעיה היא CVRP ו- 6 אם היא Ackley.


Top 5 Islands numbers by Best Fitness
Fitness
Islands Count
Best Fitness
Average Fitness
Islands=7
Islands=10
Islands=13
Islands=12
Islands=11
1888.367
1945.620
1894.730
1959.631
1895.588
1962.750
1903.112
1971.678
1903.638
1978.734


Top 5 Islands numbers by Average Fitness
Fitness
Islands Count
Best Fitness
Average Fitness
Islands=7
Islands=10
Islands=13
Islands=8
Islands=4
1888.367
1945.620
1894.730
1959.631
1895.588
1962.750
1905.313
1967.555
1909.802
1968.364


Top 5 Islands numbers by Average Fitness
Fitness
Islands Count
Best Fitness
Average Fitness
Islands=6
Islands=8
Islands=14
Islands=3
Islands=4
0.000
0.000
0.000
0.000
0.000
0.003
0.006
0.006
0.014
0.014

סעיף 4

האלגוריתם מיוצג ע"י המחלקה ALNSAlgorithm שעובדת בצורה הבאה:

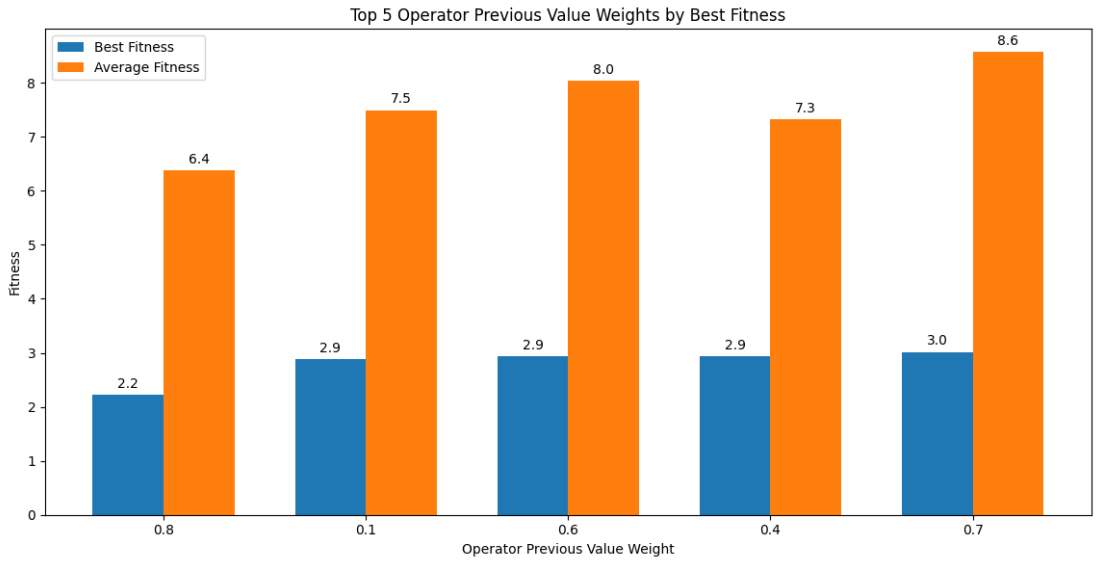
1. יצירת אוסף פתרונות התחלתי.
2. בכל איטרציה מבוצע הבא:
   a. לכל פתרון נמצא שכן לפי אופרטור כלשהו, ונבדק אם הפיטניס שלו קטן מהפיטניס של הפתרון. אם כן, הוא מחליף אותו, אחרת מחליף אותו בהסתברות כלשהי.
   b. המשקלים של האופרטורים מעודכנים.

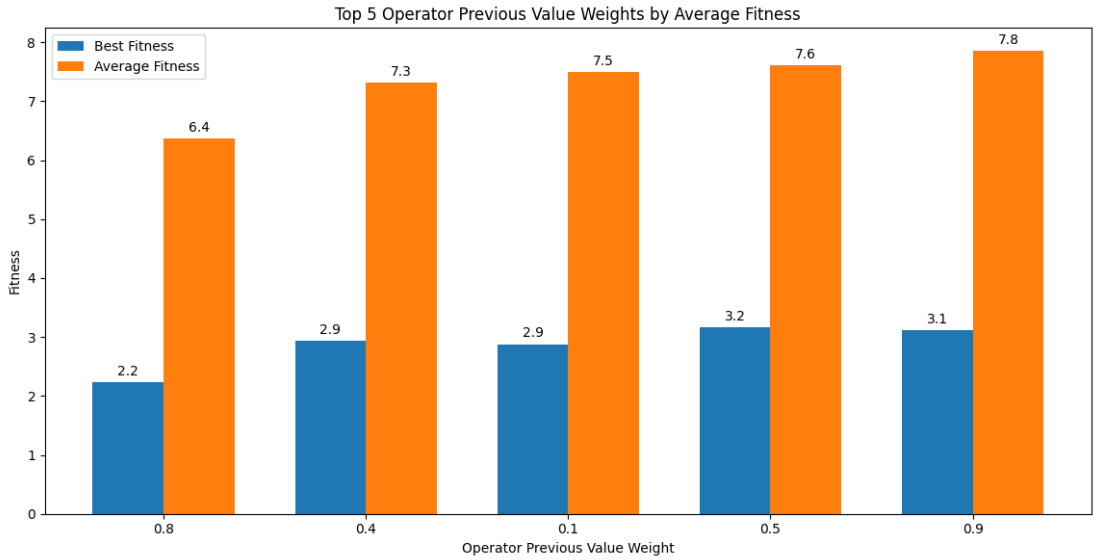
אתחול האוכלוסייה מתבצע באותה דרך שהוסברה בסעיף 1 בשיטה שמתבסס על הרעיון של ה- K-means.

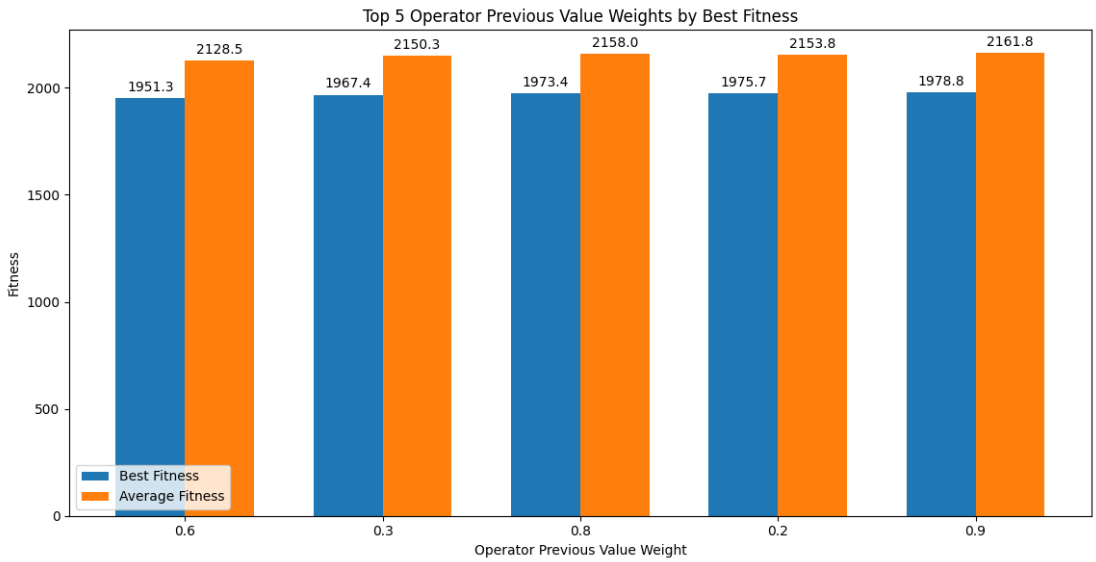
על מנת למצוא שכנים, האלגוריתם משתמש באותם אופרטורים שהשתמשנו בהם בסעיף 2, כאשר עבור בעיית CVRP האופרטורים הם 2-opt ההופכת קטע אקראי הלקוח מתוך מסלול אקראי, relocate הלוקחת לקוח אקראי ומכניסה אותו למקום כלשהו במסלול כלשהו (כולל בדיקת תקינות), reposition הדומה לקודם אך מקומו החדש של הלקוח יהיה בתוך אותו מסלול, swap המחליפה בין שני לקוחות (כולל בדיקת תקינות), ו- shuffle הבוחר מסלול אקראי ומערבב את סדר הלקוחות בו, ועבור בעיית Ackley האופרטורים הם: shift one, המוסיפה "רעש" שהוא ערך אקראי לאחד הממדים, shift all המוסיפה רעש שהוא וקטור עם ערכים אקראיים לווקטור, ו- set random המחליפה את הערך של אחד הממדים בערך כלשהו. בחירת האופרטור תהיה באופן הסתברותי ביחס למשקל שלו.

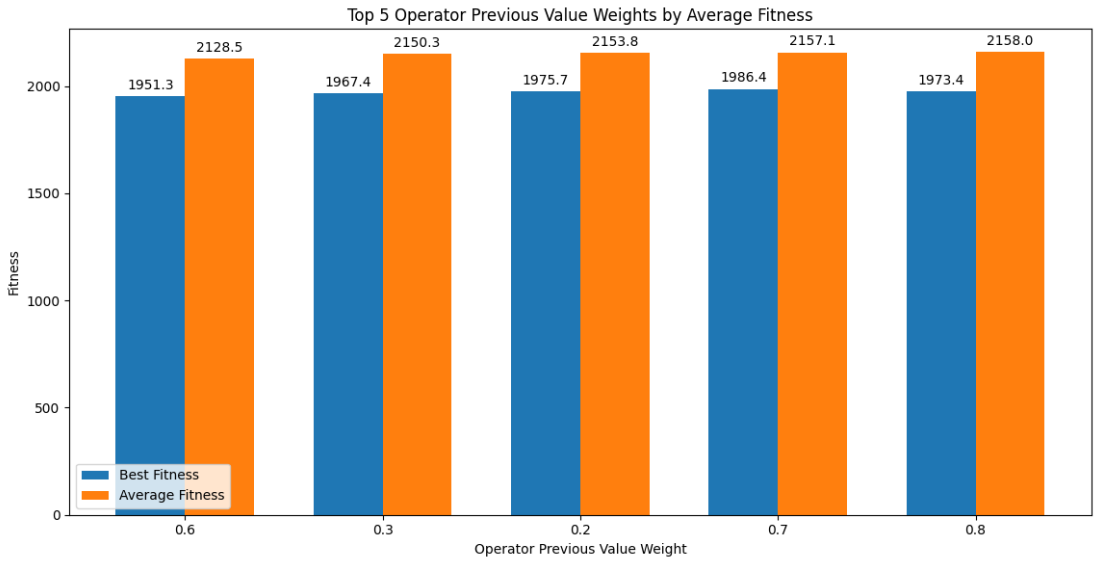
במידה והפיטניס של השכן המתקבל לא טוב מזה של הפתרון הנוכחי, משתמשים בשיטת Simulated Annealing, שפותחה והוסברה באלגוריתם ILS בסעיף ב', על מנת להחליט אם לאמץ אותו בכל זאת או שלא.

המשקל של האופרטור מעודכן בכל איטרציה כצירוף של 0.6 * המשקל הקודם שלו ו- 0.4 * מידת ההצלחה שלו באיטרציה הנוכחית בבעיית ניתוב הרכבים, האחוזים משתנים ל- 0.8 ו- 0.2 בהתאמה במקרה שהבעיה היא Ackley, כאשר מידת ההצלחה מחושבת כמספר הפעמים שבהם השימוש באופרטור הניב שכן בעל פיטניס טוב יותר מחולק במספר הפעמים הכללי בהם נעשה שימוש באופרטור. בגלל שהבחנו בהשפעה הגדולה של החלוקה הזאת על ביצועי האלגוריתם, הוחלט לעשות לה tuning שהשווה בין חלוקות שונות מכפילויות של 0.1 וסכומם 1 (למשל, אופציות אחרות היו הזוגות (0.1,0.9) ו- (0.3, 0.7)):


Top 5 Operator Previous Value Weights by Best Fitness
Best Fitness
Average Fitness
Fitness
Operator Previous Value Weight
0.8
0.1
0.6
0.4
0.7
2.2
6.4
2.9
7.5
2.9
8.0
2.9
7.3
3.0
8.6


Top 5 Operator Previous Value Weights by Average Fitness
Best Fitness
Average Fitness
Fitness
Operator Previous Value Weight
0.8
0.4
0.1
0.5
0.9
2.2
6.4
2.9
7.3
2.9
7.5
3.2
7.6
3.1
7.8


Top 5 Operator Previous Value Weights by Best Fitness
Best Fitness
Average Fitness
Fitness
Operator Previous Value Weight
0.6
0.3
0.8
0.2
0.9
1951.3
2128.5
1967.4
2150.3
1973.4
2158.0
1975.7
2153.8
1978.8
2161.8



סעיף 5

האלגוריתם ממומש ע"י המחלקה BranchAndBoundAlgorithm, המשתמשת בעץ חיפוש לבניית הפתרון באופן הדרגתי. האלגוריתם הזה בשונה מהאלגוריתמים הקודמים ממומש בצורה שונה לכל אחת מהבעיות, אך תוך שימוש באותה לוגיקה של פירוק לבעיות קטנות ובנייה הדרגתית של הפתרון.

בבעיית ניתוב הרכבים, הצומת בעץ מייצג פתרון חלקי המכיל את המסלולים שנבנו עד כה ,כולל המסלול הנוכחי שבתהליך בניה, את הלקוחות הנכללים במסלולים הללו, את הלקוחות שנשארו, וחסם תחתון על העלות הסופית של הפתרון. מכאן שעלה הוא מצב בו כל הלקוחות נכללים במסלולים, והעלות המשוערת היא כעלות האמיתית של הפתרון. כמו כן משתמשים בערימת מינימום העוזרת בלבחור את המצב הכי מבטיח על מנת להתקדם ממנו, שהוא המצב בעל החסם התחתון הכי נמוך ברגע כלשהו.

שלבי האלגוריתם הם כבא:

1. **מאתחלים ערימת מינימום עם המצב ההתחלתי** שהוא מצב בו אין מסלולים, כל הלקוחות עוד לא שובצו, והעלות של המסלול עד עכשיו היא 0.
2. כל עוד הערימה לא ריקה, **שולפים את המצב הכי מבטיח מערימת המינימום**:
   a. אם כל הלקוחות של המצב שובצו, אז יש פתרון מלא, ולכן מחשבים את העלות המלאה שלו, ואם הוא יותר טוב מהטוב ביותר שנמצא עד כה הופכים אותו להיות הטוב ביותר.
   b. אם יש לקוחות שנשארו, מתנהגים לפי אם יש מסלול פתוח:

<u>אם יש מסלול פתוח</u>: מסתכלים על k (3) הלקוחות הקרובים ביותר ללקוח האחרון במסלול ולכל אחד בודקים אם ניתן להוסיף אותו אחריו (אם קיבולת הרכב מאפשרת), אם כן, מוסיפים אותו, אחרת סוגרים את המסלול הנוכחי.

<u>אם אין מסלול פתוח</u>: מוסיפים מסלול חדש עם הלקוח הבא בתור (שהוא הכי קרוב למחסן מבין כל הלקוחות שנשארו).

c. **מחשבים את החסם התחתון החדש של הפתרון**, ואם הוא טוב מספיק (קטן מהפתרון המלא הטוב ביותר שנמצא עד עכשיו) **שומרים אותו בערימה**.

חישוב החסם התחתון מתבצע לפי אם יש מסלול פתוח או שלא. אם יש אז הוא יהיה כסכום של המרחקים של המסלולים כולל הנוכחי, בנוסף לעץ הפורש המינימלי של הקואורדינטות של הערים שנשארו. אם אין מסלול פתוח אז הוא מחושב באותה שיטה, רק שבמקום המסלול הנוכחי שלכאורה נסגר מחושב מסלול דמה שמכיל את העיר שהייתה אמור להיכנס למסלול, דבר זה נעשה בשל כך שבוודאי יהיה מסלול נוסף שיתחיל ויחזור למחסן הכולל את הצומת ההיא.

בשלב b בחלק "אם יש מסלול פתוח" היינו לוקחים בחשבון את כל הלקוחות שנשארו ומנסים לשרשר אותם ללקוח האחרון במסלול, אולם זה עלה בסיבוכיות זמן גבוהה מאד, מכאן בא הרעיון של להסתפק ב- k השכנים הקרובים ביותר, דבר שמפחית סיבוכיות ומפקס את האלגוריתם לכיוון פתרונות בעלי פוטנציאל גדול. הערך של k נקבע אותו להיות 3 כברירת מחדל, שאיזן בין סיבוכיות הזמן לבין איכות הפתרון המתקבל, אולם בהיותו משתנה ניתן לשנות את הערך שלו בהתאם לצורך.

בבעיית Ackley, המצב כולל רשימה של ערכים של הממדים שכבר נבחרו, את מספר הממדים שכבר נבחרו, חסם תחתון על העלות של הווקטור (הפתרון) הסופי, ולן עלה הוא מצב בו מספר הממדים שנבחרו הוא כמספר הממדים בפתרון הסופי (שהוא 10 במקרה שלנו). האלגוריתם הוא כמפורט להלן:

1. **מאתחלים ערימת מינימום עם המצב ההתחלתי** שהוא מצב דמה בו החסם התחתון הוא אפס, ווקטור ריק, ומספר הממדים שנבחרו הוא אפס.
2. כל עוד הערימה לא ריקה, **שולפים את המצב הכי מבטיח**, ומתנהגים לפי אם מספר הממדים שנבחרו הוא כמספר הממדים הרצוי בפתרון הסופי (10 במקרה שלנו):
<u>אם כן</u>, מחשבים את העלות האמיתית של הווקטור, ואם הוא יותר טוב מהטוב ביותר, הופכים אותו להיות הטוב ביותר.

אם לא, מפרקים את הטווח [-32.768, 32.768] ל- 100 תת-טווחים, לוקחים באופן רנדומלי ערך מכל תת-טווח, מוסיפים אותו לווקטור ומחשבים את החסם התחתון על הפתרון הסופי, ואם הוא טוב מספיק (הכי טוב שנמצא עד כה) מוסיפים אותו לערימה.

חישוב החסם התחתון על העלות מתבצע ע"י ריפוד אפסים, כלומר בהנחה שכל שאר הממדים בעלי ערך 0.

היורסטיקות שנעשה בהם שימוש במעבדה

- **היוריסטיקת בניית הפתרון ההתחלתי**: נעשה בה שימוש באמצעות הפונקציה cvrp_generate_assignment המשמשת אלגוריתמים רבים, ביניהם Multi-Stage Heuristics Algorithm לאתחול אוכלוסייה התחלתית. ההיוריסטיקה מבוססת על כך שערים קרובות אחת לשנייה כדאי לבקר ביחד, וזה לא יוסיף הרבה אורך, ולכן הפונקציה, ששואבת את הרעיון שלה מאלגוריתם k-means, מנסה לחלק את הערים לקבוצות שמספרם עד מספר הרכבים על בסיס מיקום גאוגרפי כך שהערים באותה קבוצת יכללו באותו מסלול.
  סיבוכיות: בכל איטרציה יש לנו N לקוחות, ומחשבים את המרחק שלהם מ- K צנטרואידים, וכל חישוב לוקח $O(1)$ זמן. יש לנו 4 איטרציות בסה"כ שזה קבוע. ולכן הסיבוכיות הכוללת היא $O(N * K)$.
- **היוריסטיקת סידור הלקוחות בתוך המסלול**: נעשה בה שימוש באלגוריתם Multi-Stage Heuristics, בשלב שאחרי אתחול האוכלוסייה, במטרה לקבוע את סדר הערים בתוך המסלול. ההיוריסטיקה מתבססת על כך שבכל צעד העיר הסבי ביותר שתהיה הבאה בתור היא הכי קרובה.
  סיבוכיות: $O(N^2)$ כאשר N הוא מספר הערים בבעיה. בכל פעם אנחנו עוברים על הערים, מחשבים את המרחק שלה מהעיר הנוכחית\מהמחסן ולוקחים את הקרובה ביותר. (ספציפית באלגוריתם שלנו, אנחנו מחשבים את כלל המרחקים בין כל עיר ועיר בהתחלה, בונים מטריצה ומשם שולפים ממנה כל פעם את המרחקים שנרצה). המצב הגרוע ביותר הוא כאשר כל הערים נמצאות על אותו מסלול.
- **העץ הפורש מינימלי (MST) בין הקואורדינטות של הערים כחסם תחתון על אורך המסלולים**: באלגוריתם Branch and Bound, במהלך בניית הפתרון, שמתבצעת באופן הדרגתי, אנחנו מחשבים חסם תחתון על אורך המסלולים שעוד לא נבנו ע"י

חישוב העץ הפורש מינימלי בין הערים שעוד לא נכללו בתוך המסלולים הקיימים עד כה. הדבר מאפשר לבדוק אם כדאי להמשיך בחיפוש בתת העץ של המצב או לעשות גיזום ולחסוך במשאבים.

<u>סיבוכיות</u>: משתמשים באלגוריתם Prim, כך שבכל איטרציה אנחנו בוחנים את הערים שלא ביקרנו בהם ולוקחים את המינימלי מביניהם שזה $O(N)$ במצב הגרוע, כאשר N הוא מספר הערים\הקודקודים. מעדכנים את הקשתות שמקשרות הקודקוד אותו עם שכניו שזה גם $O(N)$. ומפני שיש לנו מספר איטרציות כמספר הקודקודים אז הסיבוכיות הכוללת היא $O(N+N)*O(N)=O(N^2)$.

- **בדיקת 3 הערים הקרובות ביותר כפוטנציאלים להיות העיר הבאה במסלול**:
  משתמשים בו באלגוריתם Branch and Bound, במהלך בניית הפתרון שנעשה באופן הדגרתי ובאמצעות עץ, אנחנו מנסים לפתח את הצומת (המייצג מצב) שזה שקול לעיר הבאה בתור אחרי העיר האחרונה במסלול הפתוח, ע"י כך שאנחנו בודקים את האופציות של 3 הערים הקרובות ביותר, על סמך כך שערים קרובות כדאי לבדק ביחד. כמו כן, כחלק מההיורסטיקה, נעשה סידור לערים לפי מרחקם מהמחסן בהתחלת האלגוריתם במטרה שכשנרצה לפתוח מסלול העיר שניקח תהיה הקרובה ביותר למחסן מבין אלה שעוד לא שובצו.
  <u>סיבוכיות:</u> שליפת המרחקים של העיר האחרונה במסלול ב- $O(N)$ כמספר הערים, מיונם מתבצע ב- $O(N*\log(N))$, הדבר נעשה ל- N לקוחות, לכן בסה"כ: $O(N^2*\log(N))$.

<u>ההשוואה בין ביצועי האלגוריתמים על הקלטים</u>

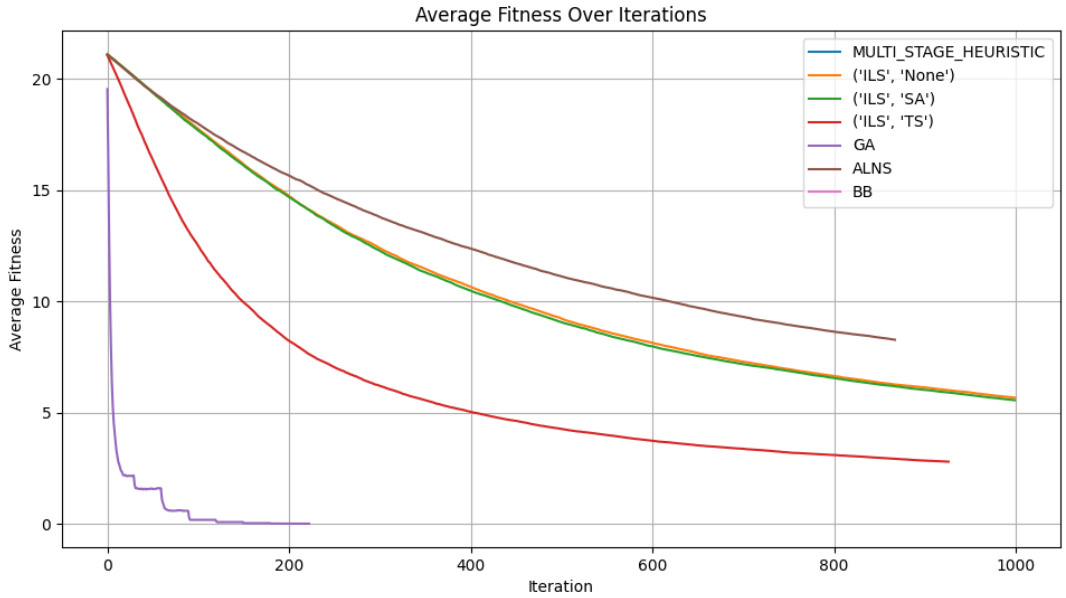
להלן תוצאות ההשוואה בין ביצועי האלגוריתמים על כל אחד מהקבצים הנתונים כקלט, בנוסף לקלט של בעיית Ackley. בחרנו להשוות בין האלגורתמים לפי הפיטניס (העלות) הכי גבוה שהפיק, ממוצע הפיטניסים, השונות בין הפיטניסים, וזמן הביצוע.

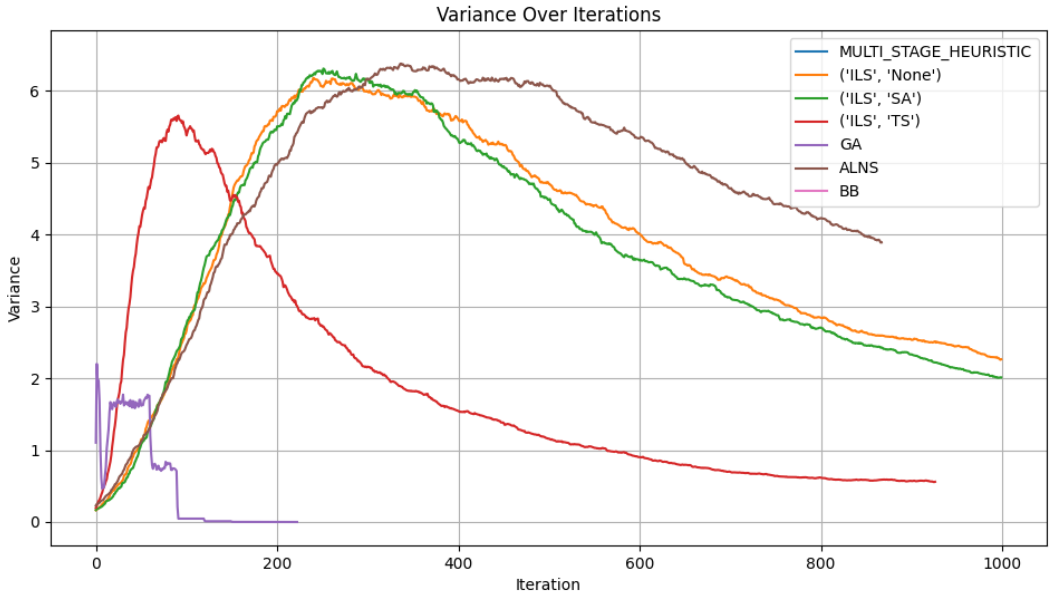
הערות:

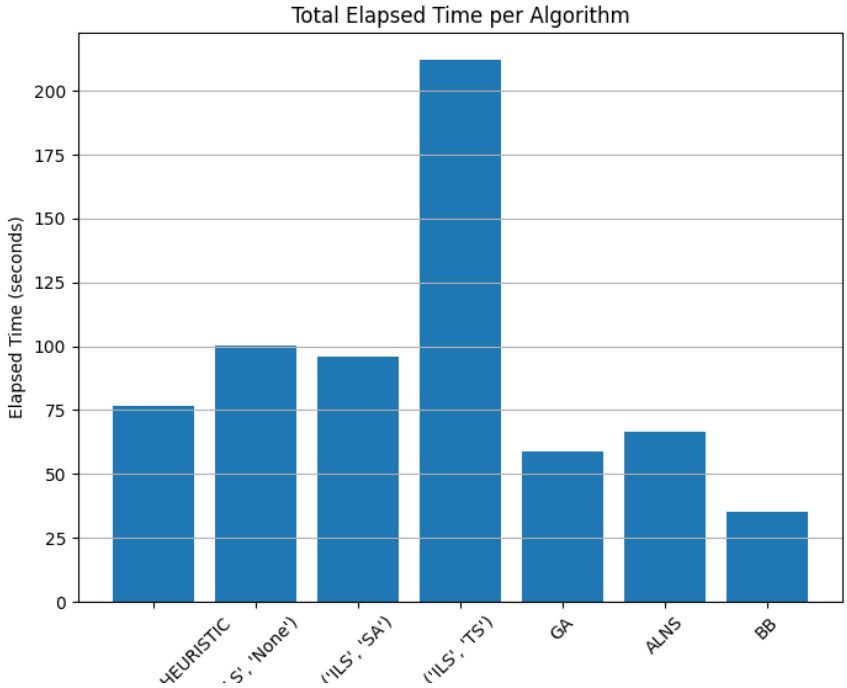
- בשל המבנה הלא איטרטיבי של שני האלגוריתמים Multi-Heuristic ו- Branch and Bound הם לא מופיעים ב- 3 הגרפים הראשונים המראים את הביצוע לאורך האיטרציות.
- בטבלה האחרונה, המספרים של הפיטניס הטוב ביותר, הממוצע, והשונות מתייחסים לערכים בתום ריצת האלגוריתם.

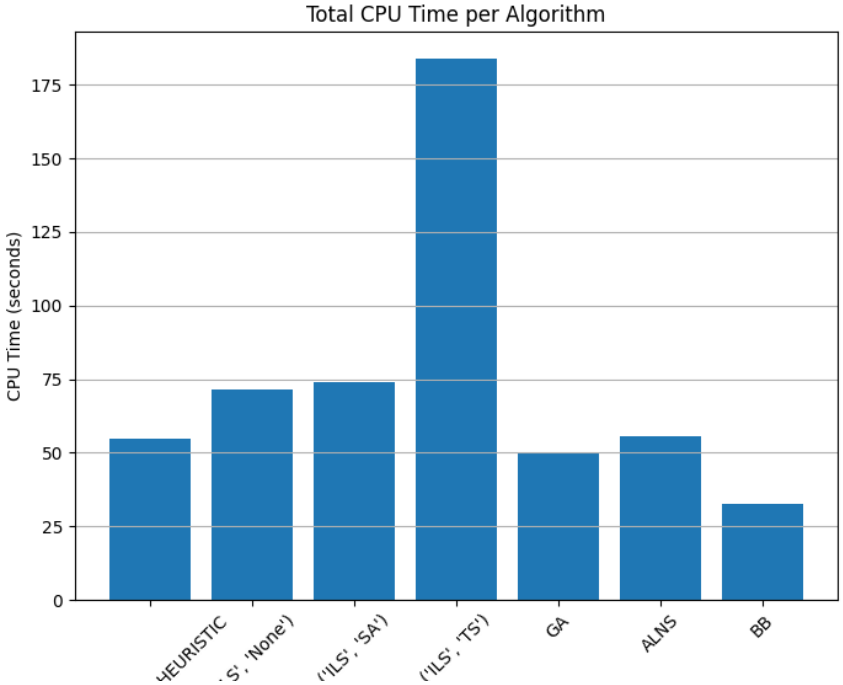
- בשל לחץ הזמן, עבור חלק מהקלטים (החלק מה- 4 בסדר כאן), הורצה גרסה מצומצמת של אלגוריתם Branch and Bound, בה עבור על עיר היא מחפשת את השכן הכי קרוב בלבד, במקום 3, לכן התוצאות לא בהכרח משקפות את ביצועי האלגוריתם שבפועל יותר טובים ולא את זמן הריצה שבפועל יותר ארוך.

בעיית Ackley:



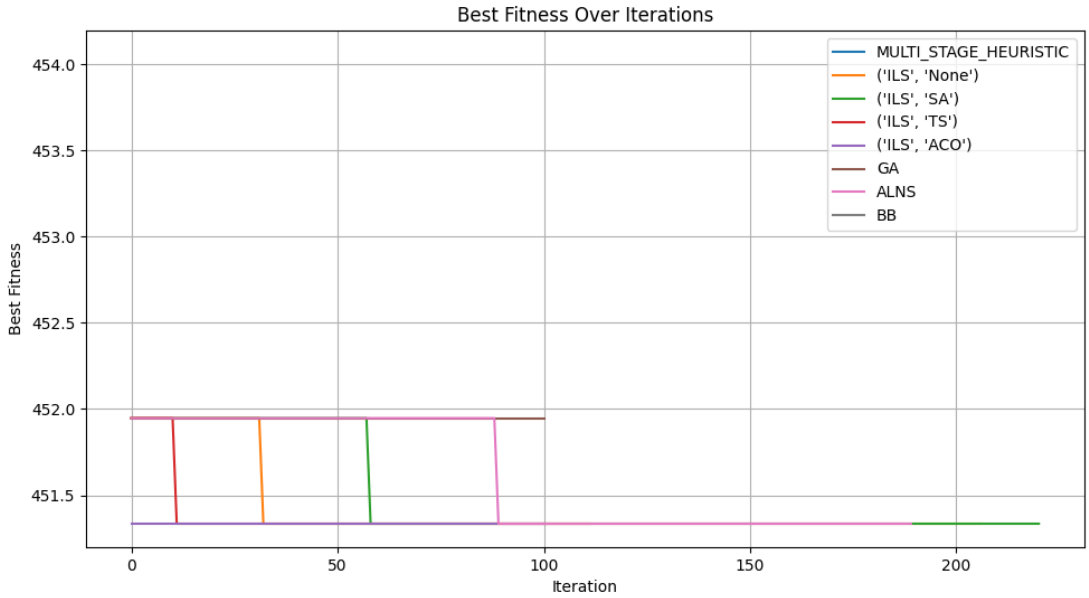

Variance Over Iterations
MULTI_STAGE_HEURISTIC
('ILS', 'None')
('ILS', 'SA')
('ILS', 'TS')
GA
ALNS
BB
Variance
Iteration
0
1
2
3
4
5
6
0
200
400
600
800
1000

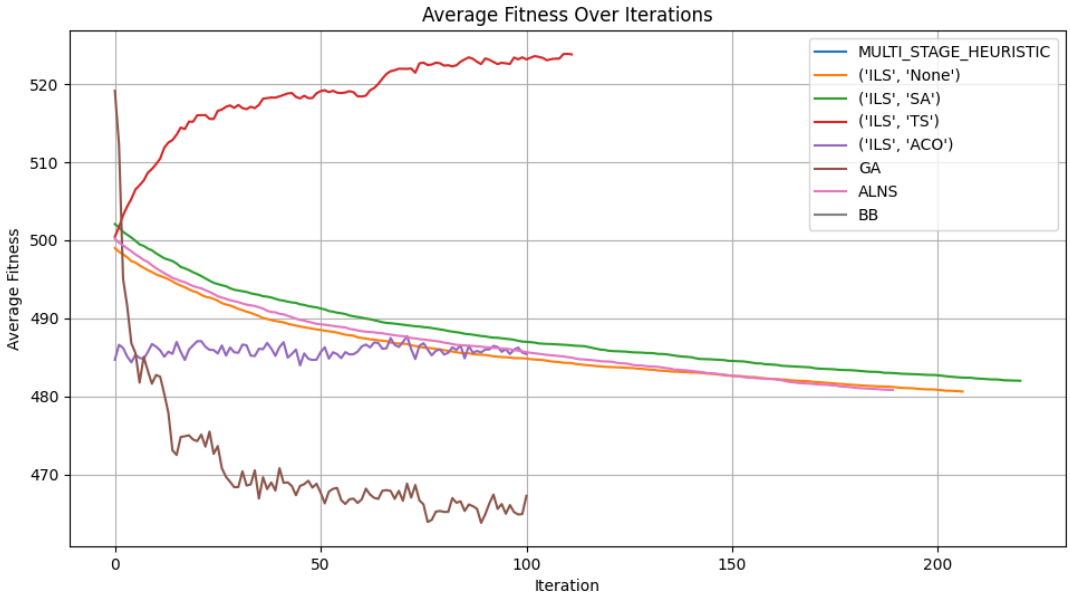

Total Elapsed Time per Algorithm
Elapsed Time (seconds)
0
25
50
75
100
125
150
175
200
HEURISTIC
S', 'None')
('ILS', 'SA')
('ILS', 'TS')
GA
ALNS
BB

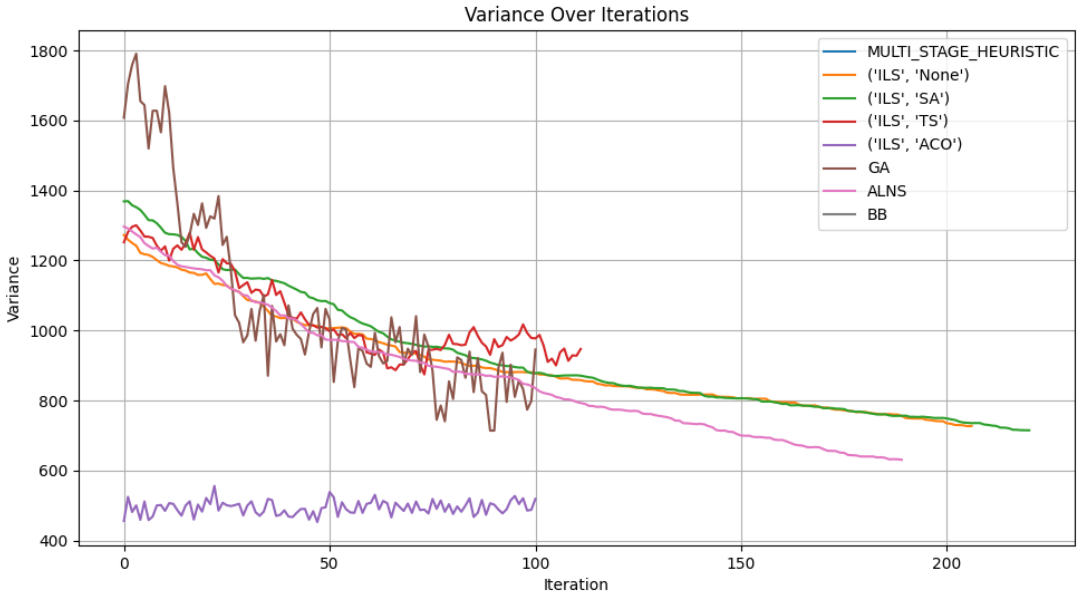


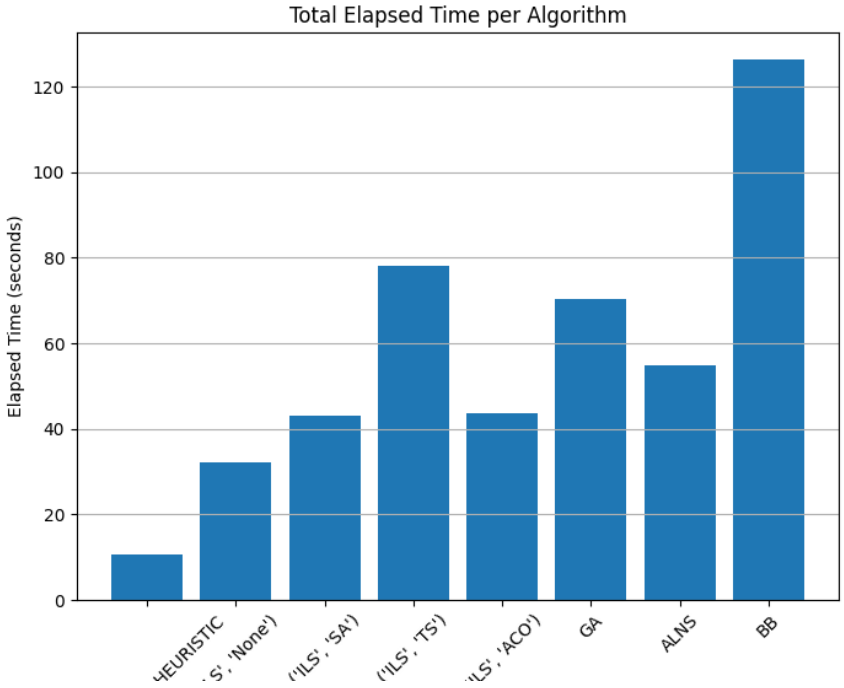
Algorithm	Best Fitness	Average Fitness	Variance	Elapsed Time	CPU Time
MULTI_STAGE_HEURISTIC	2.88	19.73	6.52	76.44	54.53
('ILS', 'None')	1.65	5.67	2.26	100.26	71.51
('ILS', 'SA')	1.74	5.55	2.01	96.04	74.13
('ILS', 'TS')	0.11	2.79	0.56	212.14	183.85
GA	0.00	0.00	0.00	58.84	50.15
ALNS	2.32	8.27	3.89	66.49	55.58
BB	2.18	-	-	35.00	32.72

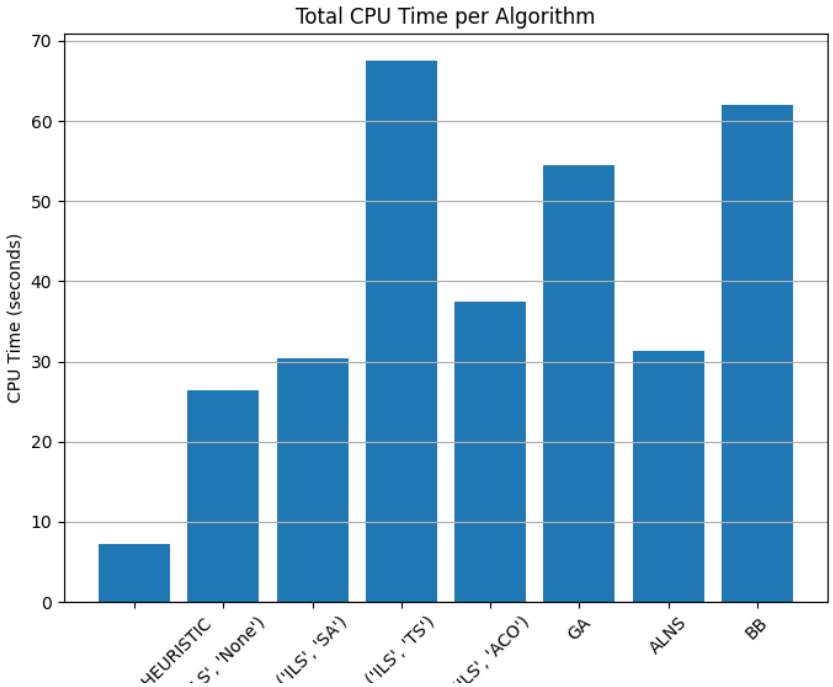
:P-n16-k8


Best Fitness Over Iterations
Best Fitness
Iteration
454.0
453.5
453.0
452.5
452.0
451.5
0
50
100
150
200
MULTI_STAGE_HEURISTIC
('ILS', 'None')
('ILS', 'SA')
('ILS', 'TS')
('ILS', 'ACO')
GA
ALNS
BB


Average Fitness Over Iterations
Average Fitness
Iteration
520
510
500
490
480
470
0
50
100
150
200
MULTI_STAGE_HEURISTIC
('ILS', 'None')
('ILS', 'SA')
('ILS', 'TS')
('ILS', 'ACO')
GA
ALNS
BB

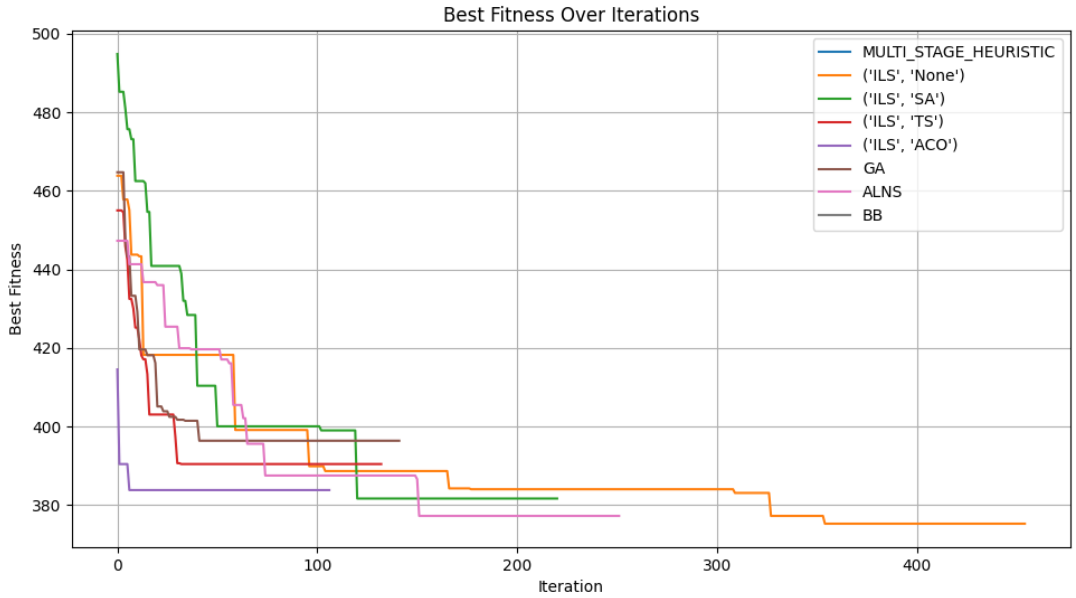

Variance Over Iterations
MULTI_STAGE_HEURISTIC
('ILS', 'None')
('ILS', 'SA')
('ILS', 'TS')
('ILS', 'ACO')
GA
ALNS
BB
Variance
Iteration
1800
1600
1400
1200
1000
800
600
400
0
50
100
150
200

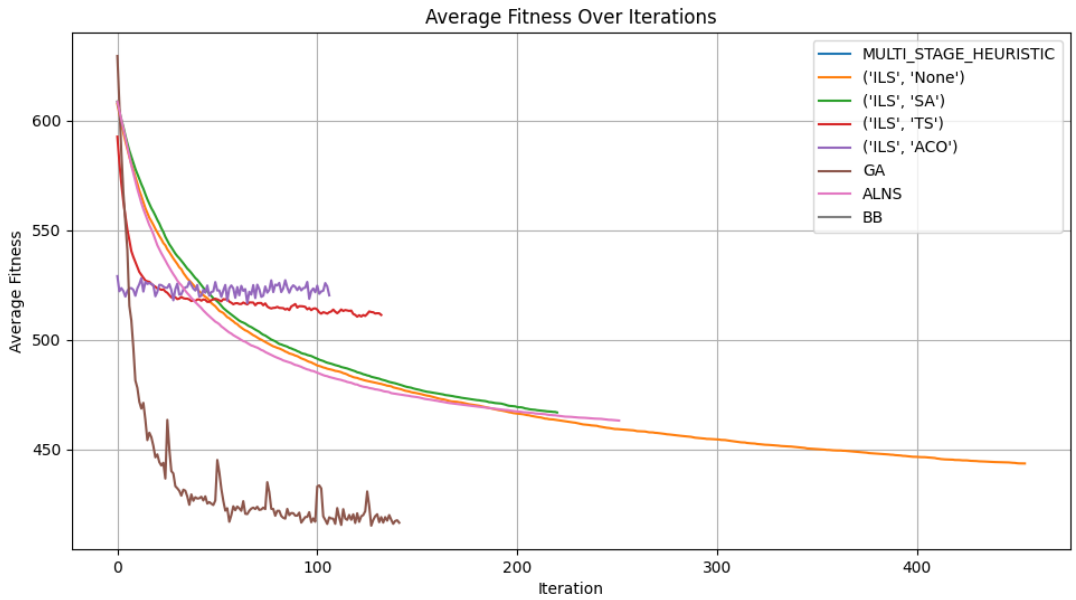

Total Elapsed Time per Algorithm
Elapsed Time (seconds)
120
100
80
60
40
20
0
GA
ALNS
BB

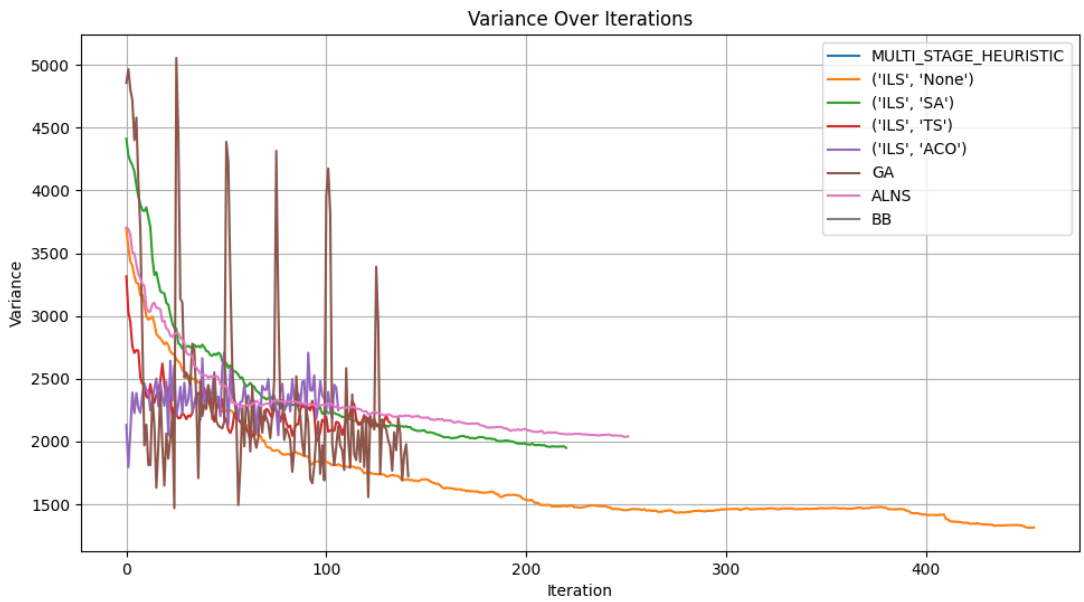


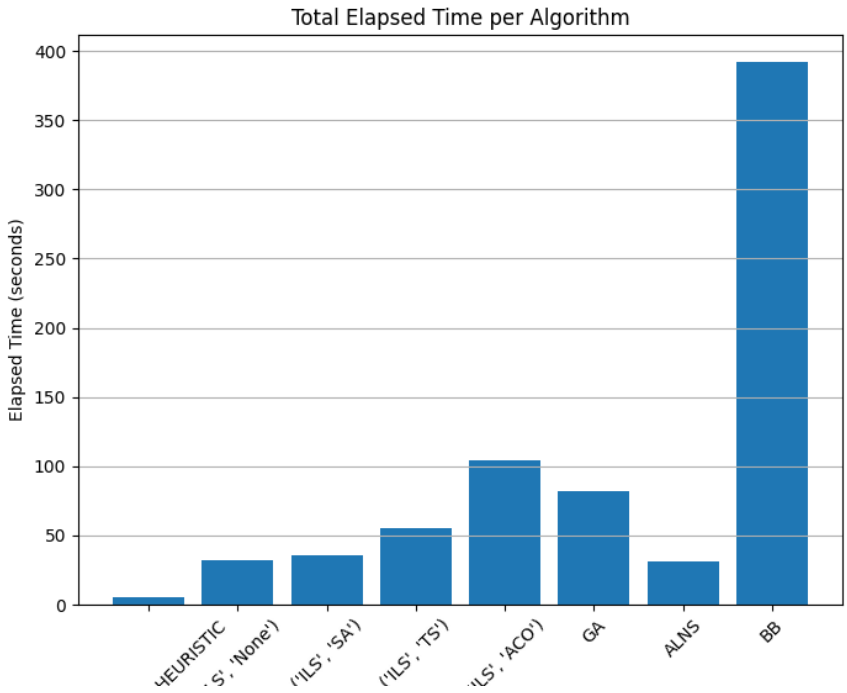
Algorithm	Best Fitness	Average Fitness	Variance	Elapsed Time	CPU Time
MULTI_STAGE_HEURISTIC	454.06	496.13	1176.85	10.60	7.18
('ILS', 'None')	451.34	480.62	727.55	32.08	26.33
('ILS', 'SA')	451.34	482.00	715.11	43.09	30.41
('ILS', 'TS')	451.34	523.81	947.27	78.07	67.54
('ILS', 'ACO')	451.34	485.42	519.17	43.70	37.41
GA	451.95	467.25	945.75	70.38	54.52
ALNS	451.34	480.82	630.99	54.79	31.28
BB	451.34	-	-	126.38	62.00

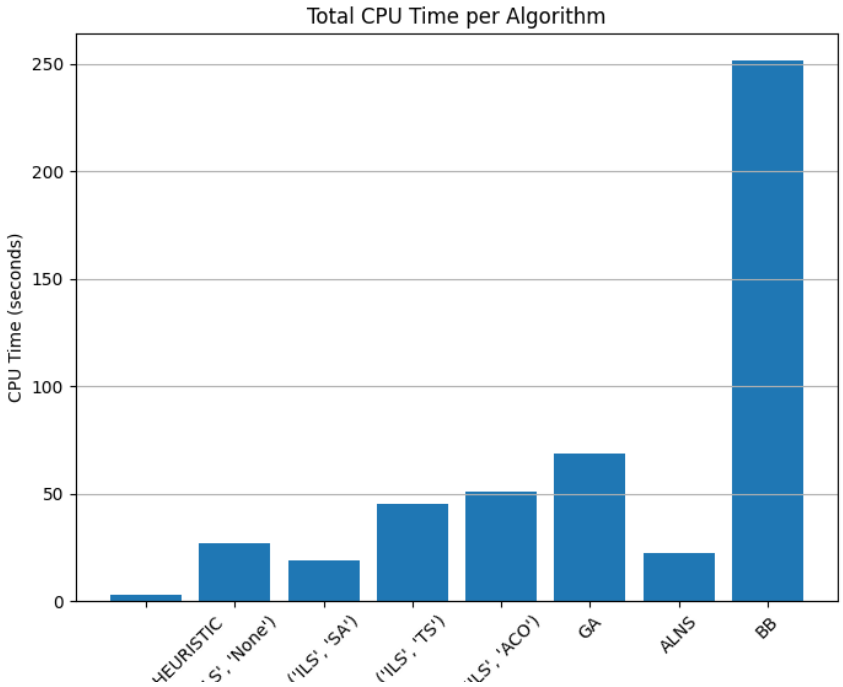
:E-n22-k4


Best Fitness Over Iterations
Best Fitness
Iteration
500
480
460
440
420
400
380
0
100
200
300
400
MULTI_STAGE_HEURISTIC
('ILS', 'None')
('ILS', 'SA')
('ILS', 'TS')
('ILS', 'ACO')
GA
ALNS
BB


Average Fitness Over Iterations
Average Fitness
Iteration
600
550
500
450
0
100
200
300
400
MULTI_STAGE_HEURISTIC
('ILS', 'None')
('ILS', 'SA')
('ILS', 'TS')
('ILS', 'ACO')
GA
ALNS
BB

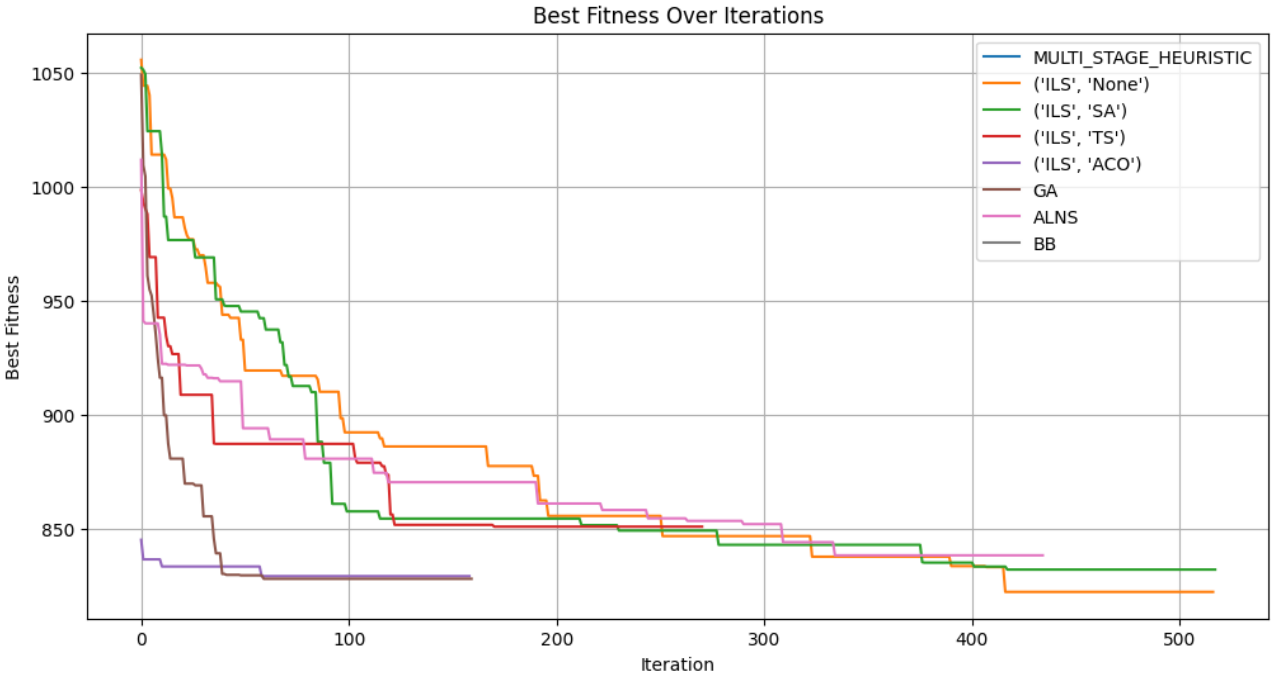

Variance Over Iterations
MULTI_STAGE_HEURISTIC
('ILS', 'None')
('ILS', 'SA')
('ILS', 'TS')
('ILS', 'ACO')
GA
ALNS
BB
Variance
Iteration
5000
4500
4000
3500
3000
2500
2000
1500
0
100
200
300
400

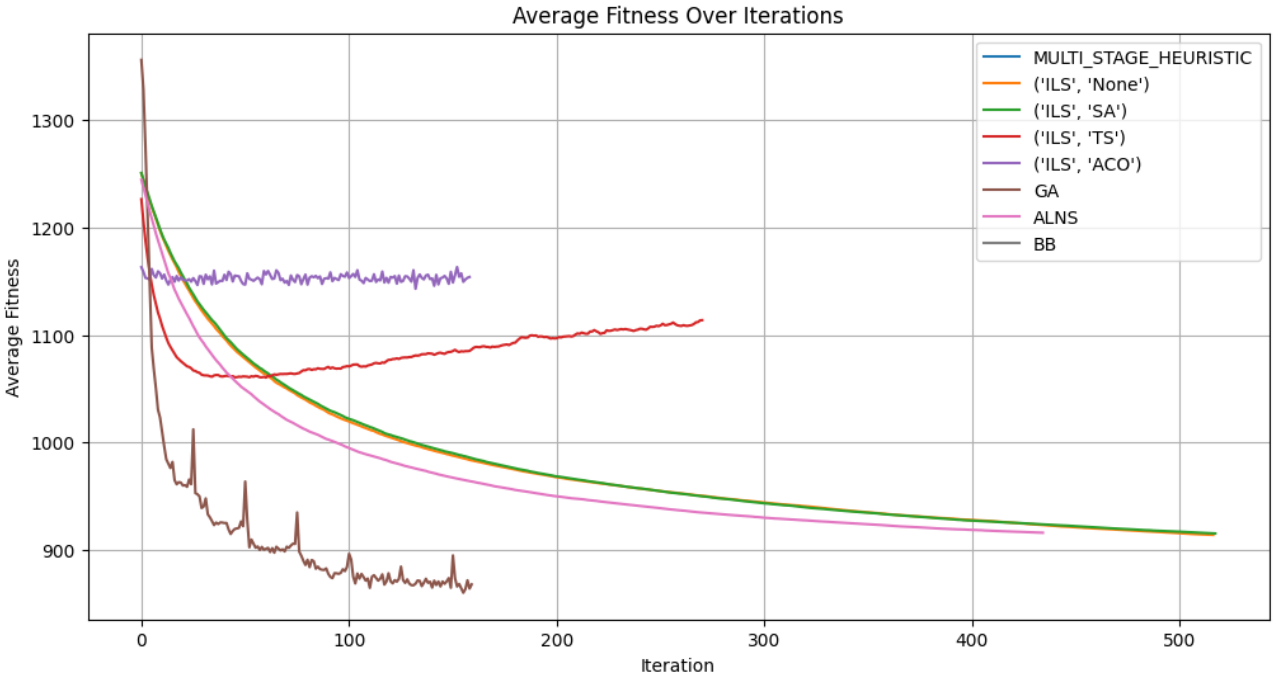

Total Elapsed Time per Algorithm
Elapsed Time (seconds)
400
350
300
250
200
150
100
50
0
GA
ALNS
BB

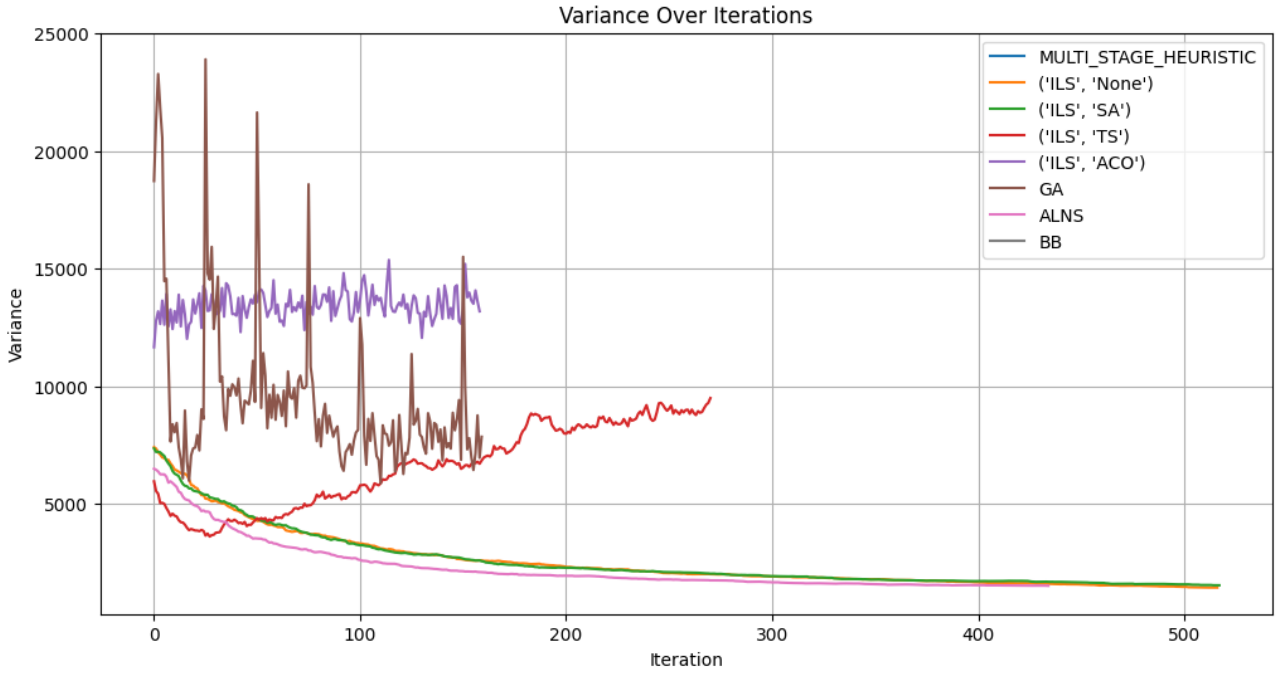


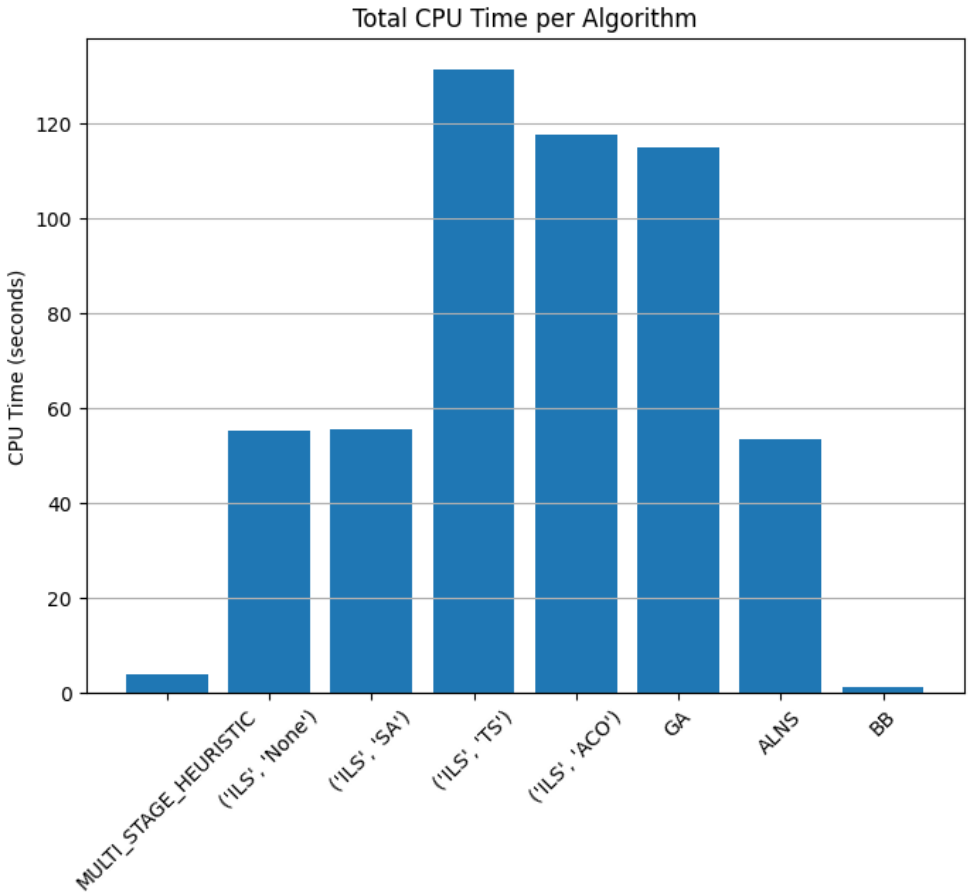
Algorithm	Best Fitness	Average Fitness	Variance	Elapsed Time	CPU Time
MULTI_STAGE_HEURISTIC	402.91	489.87	2375.57	4.95	2.65
('ILS', 'None')	375.28	443.62	1315.79	32.03	27.06
('ILS', 'SA')	381.69	466.88	1949.70	35.46	19.01
('ILS', 'TS')	390.47	511.37	2150.96	55.47	45.10
('ILS', 'ACO')	383.84	520.32	2251.34	104.08	50.86
GA	396.40	416.61	1724.60	81.89	68.77
ALNS	377.26	463.20	2041.34	31.37	22.15
BB	379.41	-	-	392.02	251.68

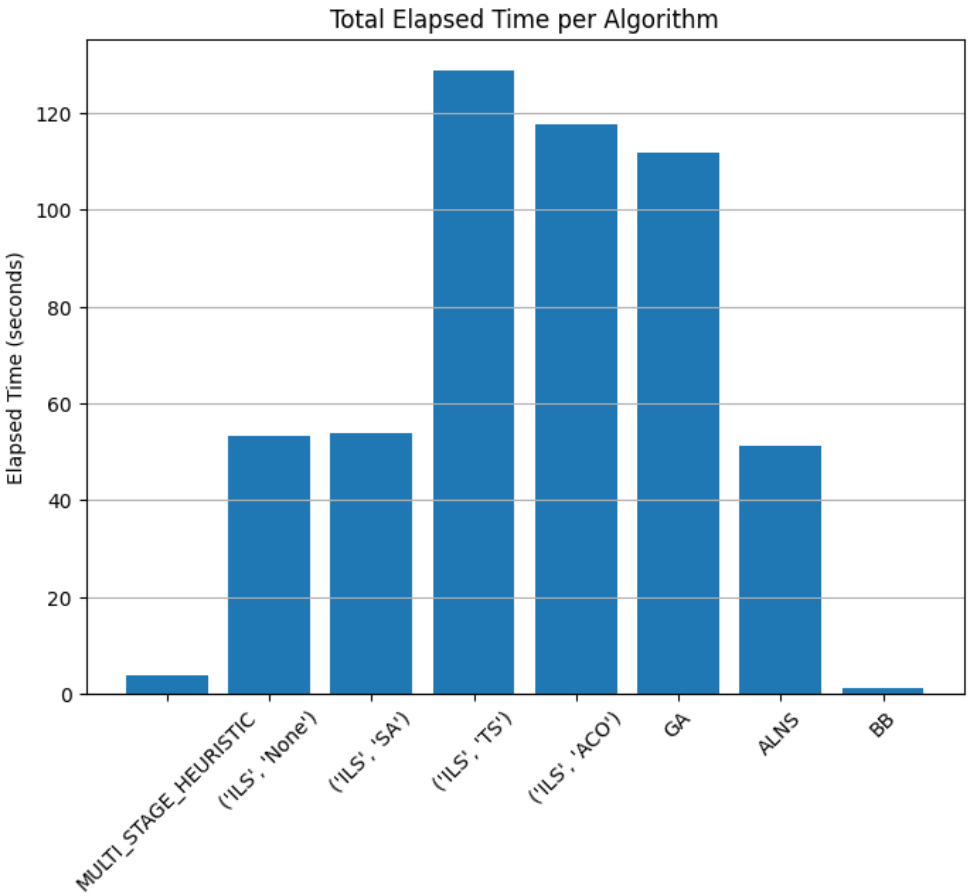
:A-n32-k5


Best Fitness Over Iterations
Best Fitness
Iteration
1050
1000
950
900
850
0
100
200
300
400
500
MULTI_STAGE_HEURISTIC
('ILS', 'None')
('ILS', 'SA')
('ILS', 'TS')
('ILS', 'ACO')
GA
ALNS
BB


Average Fitness Over Iterations
Average Fitness
Iteration
1300
1200
1100
1000
900
0
100
200
300
400
500
MULTI_STAGE_HEURISTIC
('ILS', 'None')
('ILS', 'SA')
('ILS', 'TS')
('ILS', 'ACO')
GA
ALNS
BB

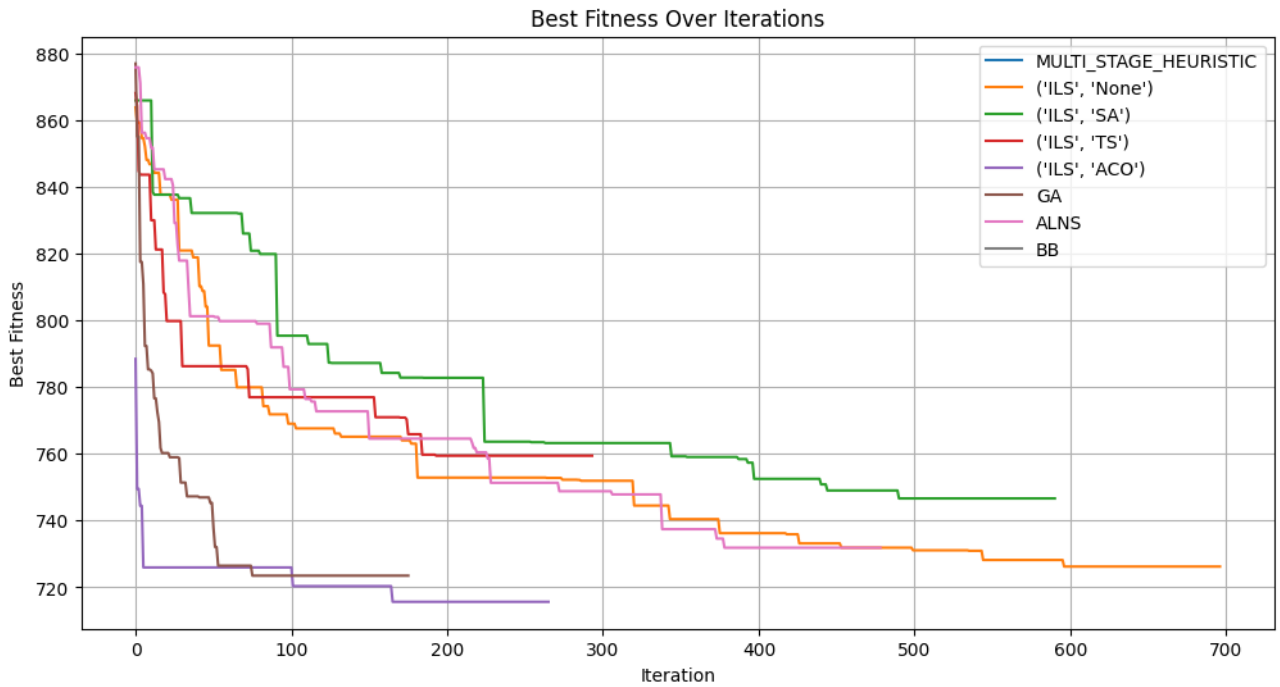

Variance Over Iterations
Variance
Iteration
25000
20000
15000
10000
5000
0
100
200
300
400
500
MULTI_STAGE_HEURISTIC
('ILS', 'None')
('ILS', 'SA')
('ILS', 'TS')
('ILS', 'ACO')
GA
ALNS
BB

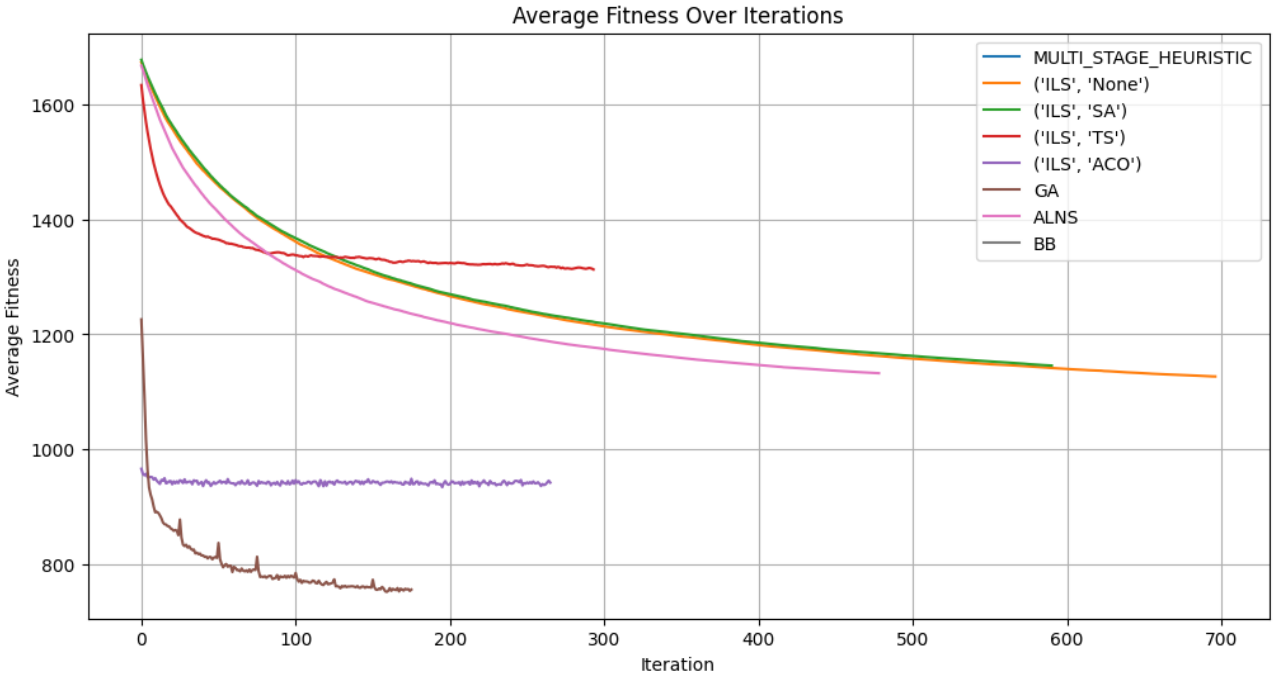

Total CPU Time per Algorithm
CPU Time (seconds)
0
20
40
60
80
100
120
MULTI_STAGE_HEURISTIC
('ILS', 'None')
('ILS', 'SA')
('ILS', 'TS')
('ILS', 'ACO')
GA
ALNS
BB

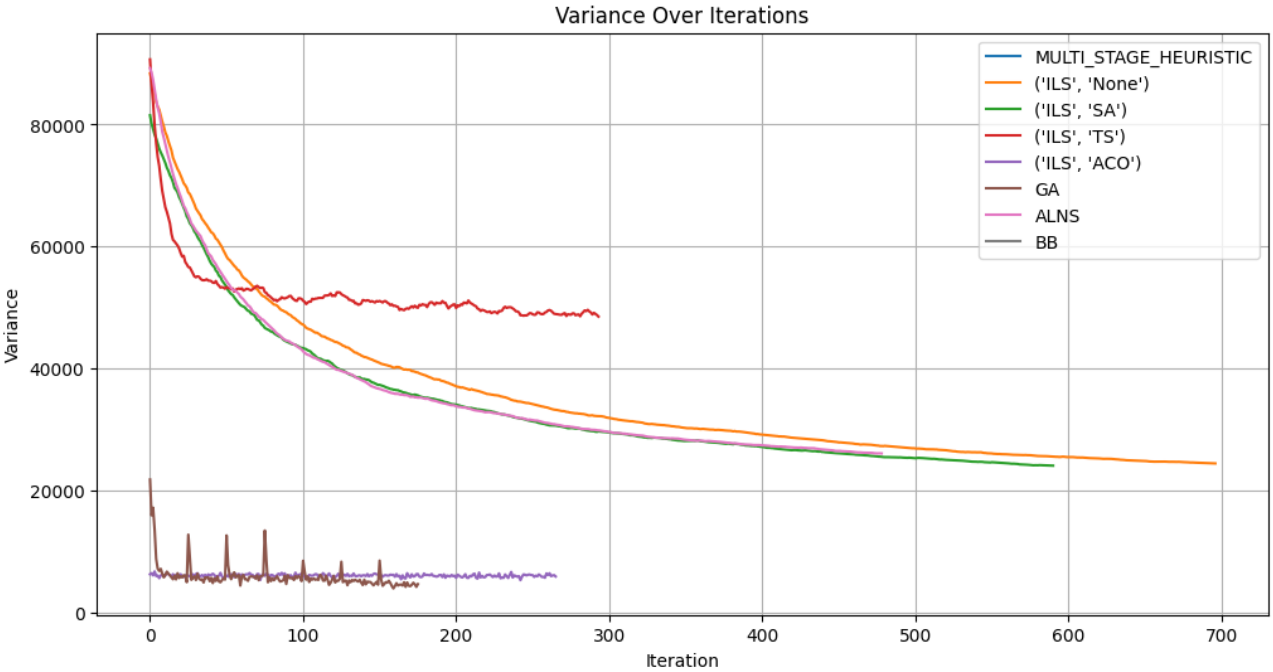

Total Elapsed Time per Algorithm
Elapsed Time (seconds)
0
20
40
60
80
100
120
MULTI_STAGE_HEURISTIC
('ILS', 'None')
('ILS', 'SA')
('ILS', 'TS')
('ILS', 'ACO')
GA
ALNS
BB

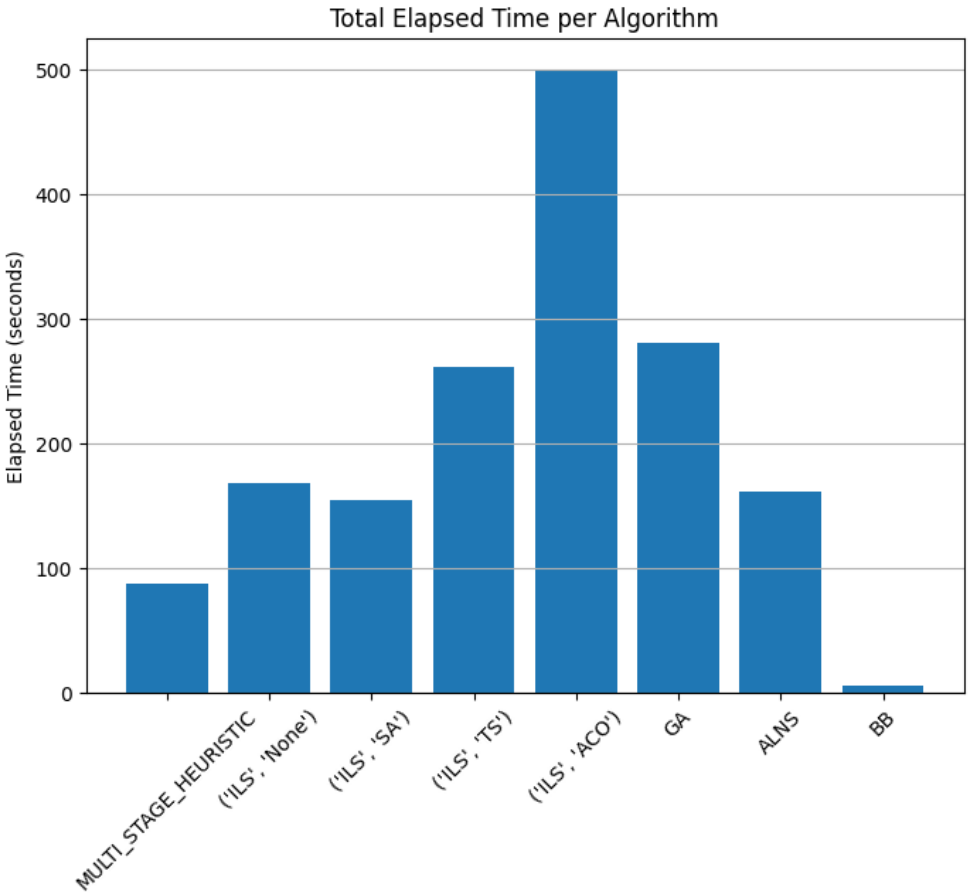
Algorithm	Best Fitness	Average Fitness	Variance	Elapsed Time	CPU Time
MULTI_STAGE_HEURISTIC	871.02	967.07	2372.34	3.77	3.83
('ILS', 'None')	822.46	913.77	1438.66	53.11	55.08
('ILS', 'SA')	832.23	915.11	1536.80	53.75	55.44
('ILS', 'TS')	851.04	1113.80	9494.83	128.85	131.43
('ILS', 'ACO')	829.36	1154.03	13185.84	117.57	117.77
GA	828.19	867.63	7851.96	111.69	115.06
ALNS	838.48	915.78	1515.96	51.32	53.52
BB	906.08	-	-	1.12	1.13

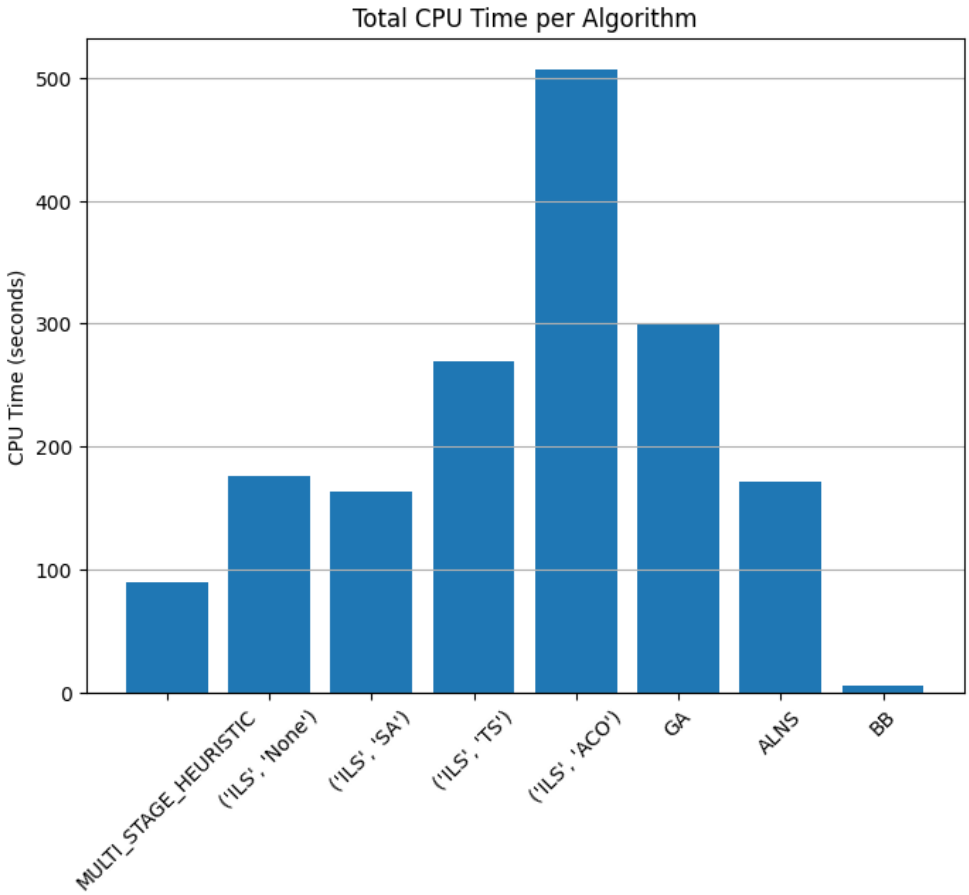
:B-n45-k6




Average Fitness Over Iterations
Average Fitness
Iteration
MULTI_STAGE_HEURISTIC
('ILS', 'None')
('ILS', 'SA')
('ILS', 'TS')
('ILS', 'ACO')
GA
ALNS
BB

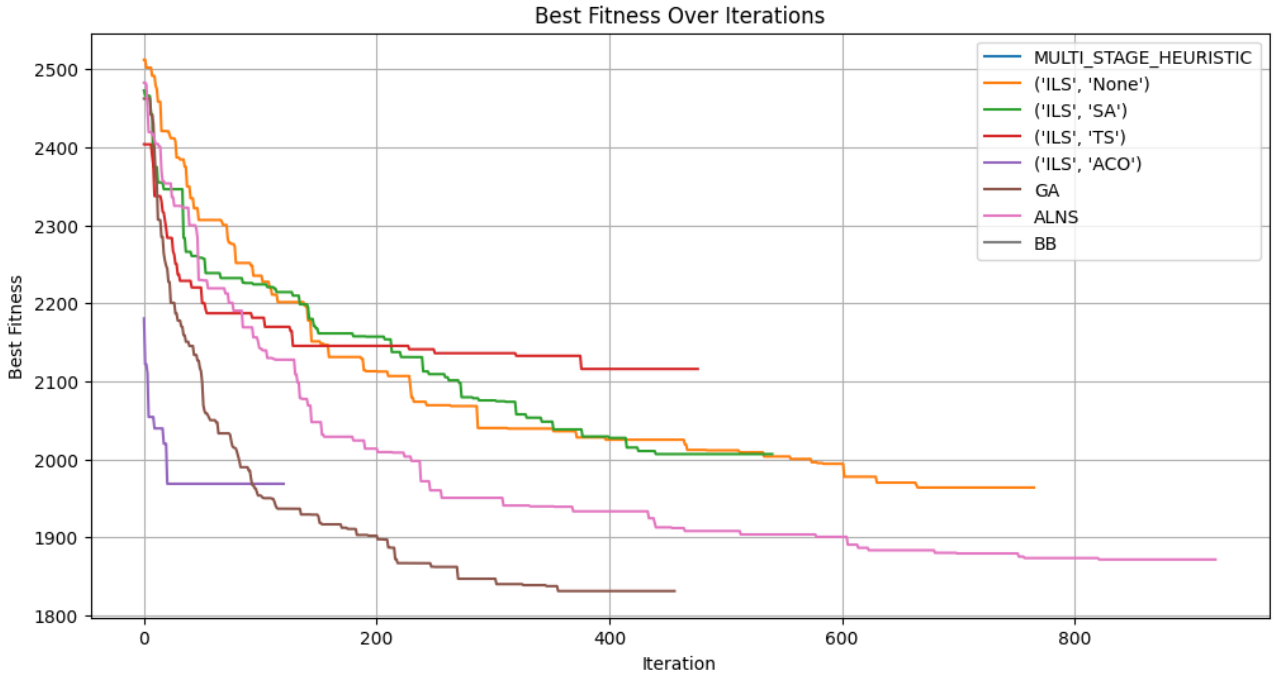

Variance Over Iterations
Variance
Iteration
MULTI_STAGE_HEURISTIC
('ILS', 'None')
('ILS', 'SA')
('ILS', 'TS')
('ILS', 'ACO')
GA
ALNS
BB

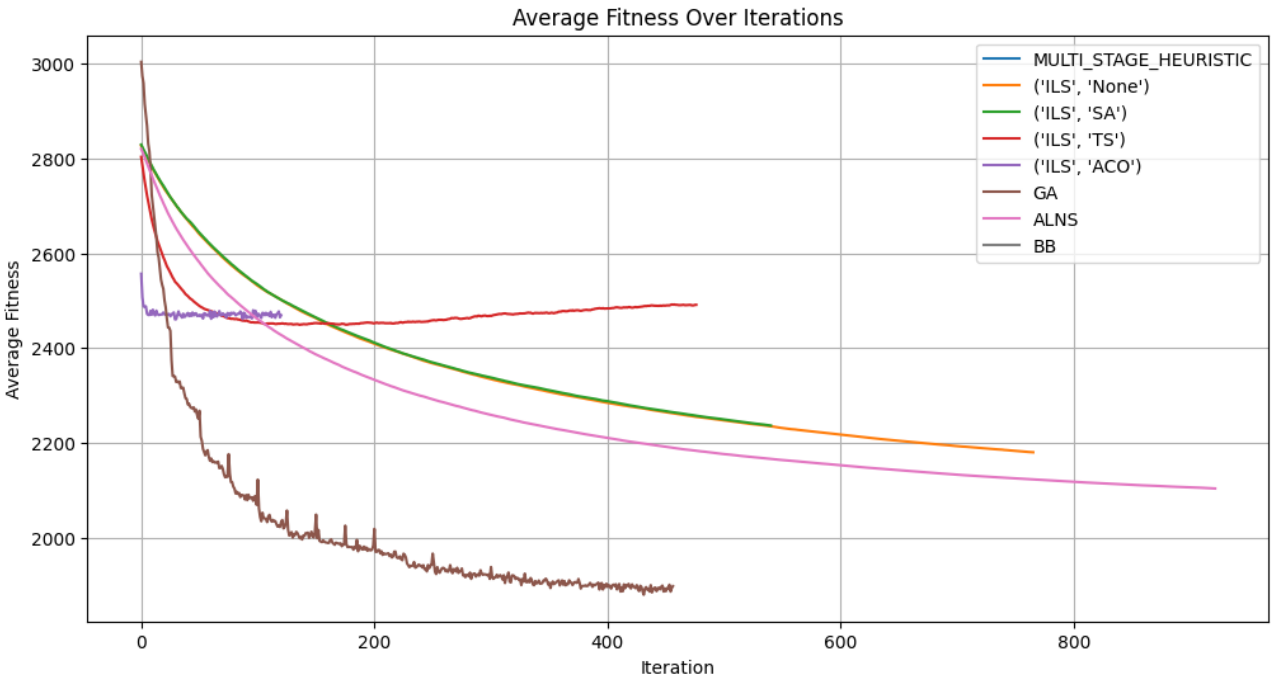

Total Elapsed Time per Algorithm
Elapsed Time (seconds)
0
100
200
300
400
500
MULTI_STAGE_HEURISTIC
('ILS', 'None')
('ILS', 'SA')
('ILS', 'TS')
('ILS', 'ACO')
GA
ALNS
BB

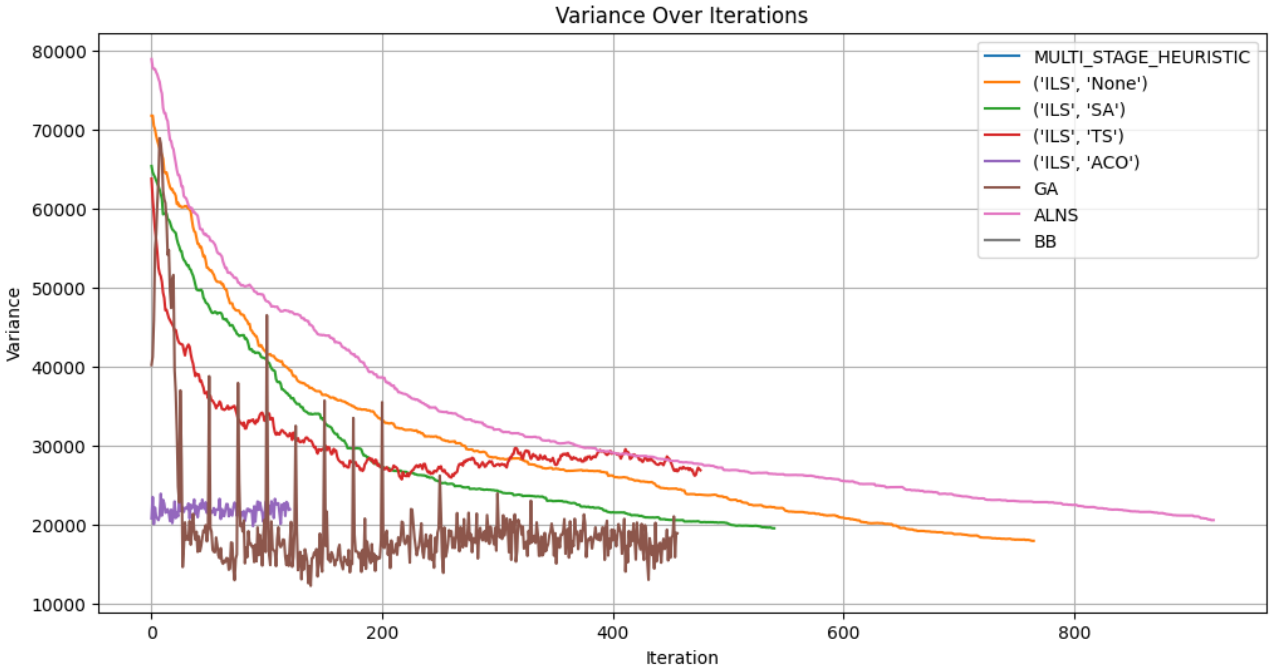


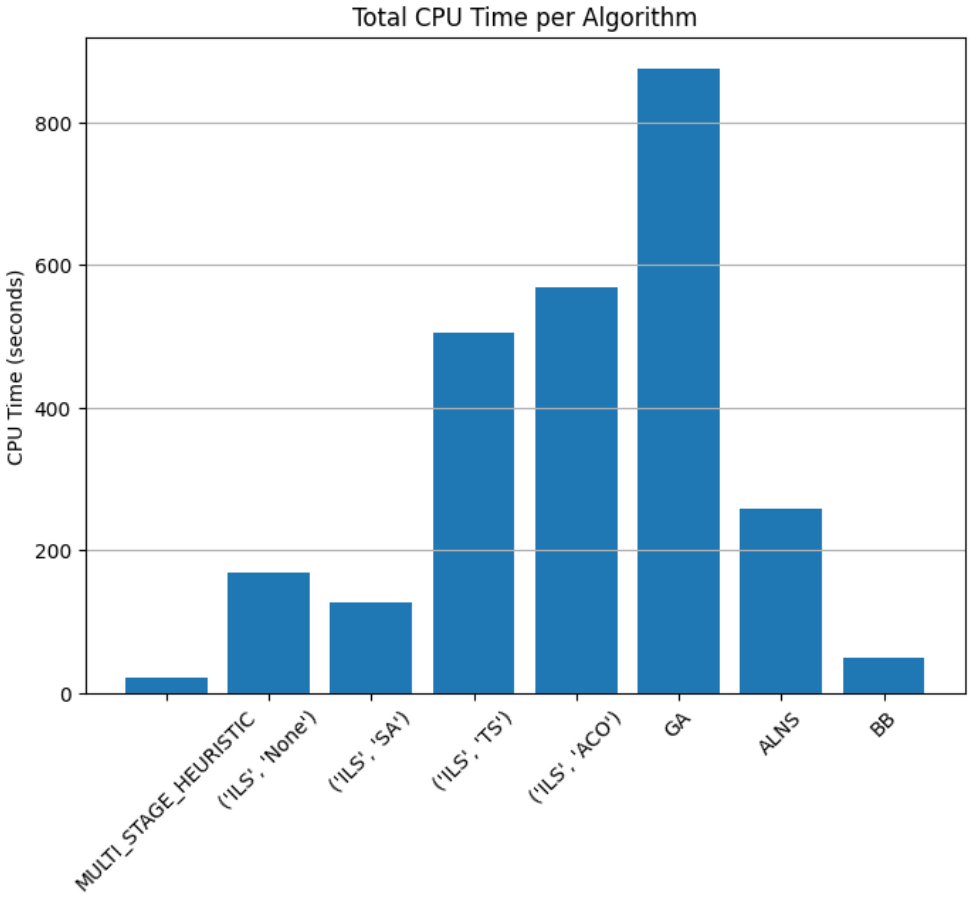
Algorithm	Best Fitness	Average Fitness	Variance	Elapsed Time	CPU Time
MULTI_STAGE_HEURISTIC	726.56	1176.34	30767.51	86.99	89.89
('ILS', 'None')	726.22	1126.85	24436.07	167.53	175.86
('ILS', 'SA')	746.63	1145.79	24070.25	153.99	163.03
('ILS', 'TS')	759.42	1312.73	48510.31	261.52	269.68
('ILS', 'ACO')	715.66	942.43	5898.85	500.09	506.90
GA	723.50	756.64	4647.03	280.99	300.00
ALNS	731.87	1132.50	26072.01	161.63	170.99
BB	781.36	-	-	5.25	5.76

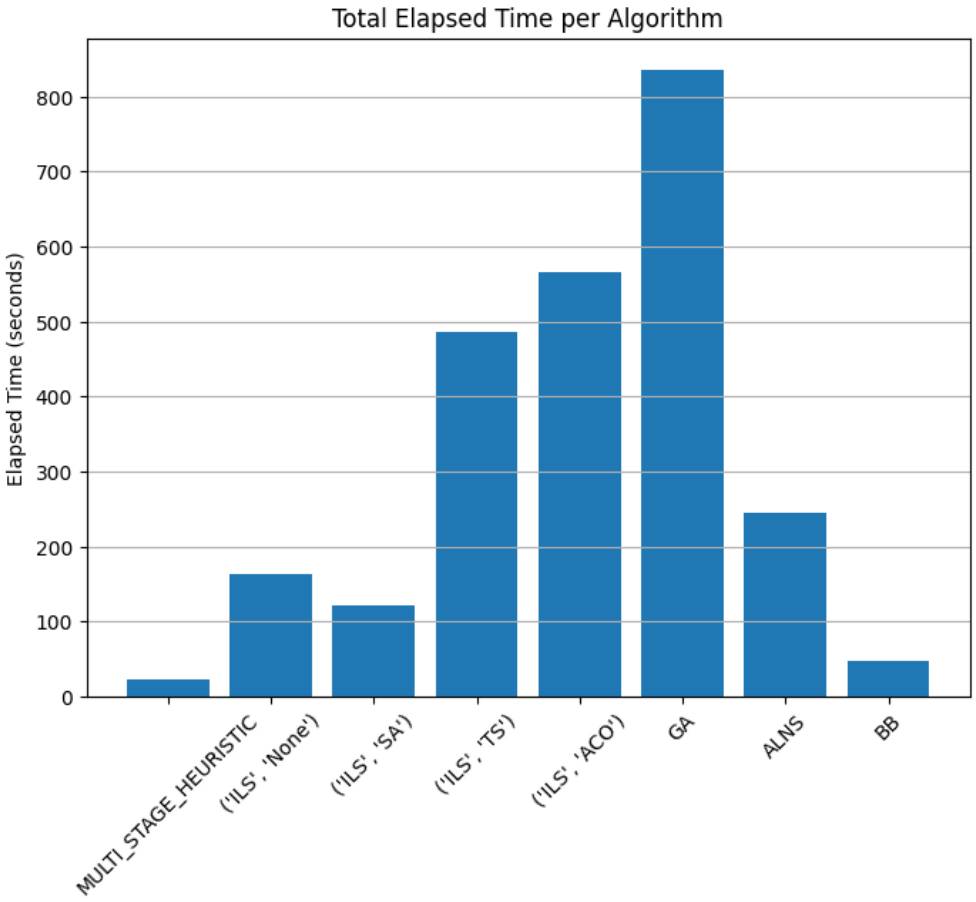
:**A-n80-k10**


Best Fitness Over Iterations
Best Fitness
Iteration
2500
2400
2300
2200
2100
2000
1900
1800
0
200
400
600
800
MULTI_STAGE_HEURISTIC
('ILS', 'None')
('ILS', 'SA')
('ILS', 'TS')
('ILS', 'ACO')
GA
ALNS
BB


Average Fitness Over Iterations
Average Fitness
Iteration
3000
2800
2600
2400
2200
2000
0
200
400
600
800
MULTI_STAGE_HEURISTIC
('ILS', 'None')
('ILS', 'SA')
('ILS', 'TS')
('ILS', 'ACO')
GA
ALNS
BB


Variance Over Iterations
Variance
Iteration
80000
70000
60000
50000
40000
30000
20000
10000
0
200
400
600
800
MULTI_STAGE_HEURISTIC
('ILS', 'None')
('ILS', 'SA')
('ILS', 'TS')
('ILS', 'ACO')
GA
ALNS
BB


Total CPU Time per Algorithm
CPU Time (seconds)
800
600
400
200
0
MULTI_STAGE_HEURISTIC
('ILS', 'None')
('ILS', 'SA')
('ILS', 'TS')
('ILS', 'ACO')
GA
ALNS
BB



Algorithm	Best Fitness	Average Fitness	Variance	Elapsed Time	CPU Time
MULTI_STAGE_HEURISTIC	1962.96	2173.10	21282.96	22.27	21.90
('ILS', 'None')	1963.94	2179.98	17913.39	162.88	169.14
('ILS', 'SA')	2006.83	2236.41	19510.05	122.16	127.46
('ILS', 'TS')	2115.95	2491.19	26853.80	485.89	505.05
('ILS', 'ACO')	1968.76	2469.26	21896.79	566.53	568.74
GA	1831.30	1897.45	18885.13	836.00	875.29
ALNS	1871.71	2103.57	20539.82	244.59	258.15
BB	2048.17	-	-	46.56	49.24

ברוב המקרים אלגוריתם ה- Multi-Stage Heuristics היה החלש ביותר במונחים של הפיטניס הטוב ביותר והפיטניס הממוצע, אולם הפגין יעילות מבחינת סיבוכיות זמן. כמו כן, ניתן לשים לב לשונות הגבוהה בין הפתרונות שלו בתום ריצתו וזה ככל הנראה בשל אופיו הלא איטרטיבי ואי הכלתו למנגנון אינטראקציה בין הפתרונות כמו במקרה האלגוריתם הגנטי ואלגוריתם ILS למשל. למרות שלא נדרש מאיתנו, בחרנו לכלול את גרסת ILS שלא משתמשת במיטא-היוריסטיקה כלשהי בשל כך שהבחנו בתוצאות הטובות שהוא משיג. דבר זה נכיר יותר בבעיות קטנות (מבחינת מספר הערים ומספר המסלולים) לעומת בעיות בעלות

סדר גדול, שם המיטא-היורסטיקה מתגלה כבעל ערך רב בהיחלצות ממינימום לוקאלי. כמו כן, אפשר להבחין, באופן הגיוני, לקשר הופכי בין מספר האיטרציות והשונות, ככל שמספר האיטרציות שהאלגוריתם דורש גדול יותר, השונות בין פתרונותיו קטנה. אלגוריתם ה-Branch and Bound דרש בד"כ זמן גדול יותר בהרבה משאר האלגוריתמים, דבר הנובע מאופן המימוש שלו, וזה למרות השיפורים שבוצעו בו, בכך שהוא לוקח בחשבון את 3 השכנים הקרובים ביותר במקום את כל שאר הערים, ואת הגיזום שהוא עושה למצבים לא מבטיחים. אלגוריתם ה- ALNS הציג יכולת טובה בשדרוג הפתרון הטוב ביותר לאורך האיטרציות תוך שמירה על שונות מתונה, דבר המעיד על איזון טוב בין Exploration ל-Exploitation. אלגוריתם שהציג שונות גבוהה במיוחד היה האלגוריתם הגנטי, עדות להצלחתו של מנגנון האיים, אולם נראה שזה בא על חשבון התכנסות מהירה ברוב המקרים.