**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 lab2.py <time\_limit> <problem> <file\_path>

python lab2.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 הרכיבים.

דוגמאות לפקודות הרצה חוקיות:

python lab2.py 300 CVRP P-n16-k8.vrp.txt

python lab2.py 500 ACKLEY

**הרצת קובץ ה- EXE**

קוד ה- EXE רץ באמצעות שתי הפקודה הבאה:

lab2.exe <time\_limit> <problem\_type> <file\_path>

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

**ייצוג הפרטים והאוכלוסייה**

* **בעיית ה- CVRP:** הפרט (הפתרון) מיוצג ע"י מופע של המחלקה CVRPIndividual המכילה 3 שדות: מסלולים\רכבים, הפיטניס, והאוכלוסייה (אוסף הפתרונות) אליה משתייך הפתרון. שדה המסלולים הוא רשימה המכילה רשימות כאשר הרשימה הפנימית מכילה את האינדקסים של הערים במסלול בסדר. 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 המחושבות 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, המימד, 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.

הפרמטרים של האלגוריתמים שונים נבחרו ע"י בחינת אופציות שונות, שהשוו ערכים שונים של פרטים, והרבה פעמים קומבינציות שונות. כל אופציה נבחנה על 5 הרצות נפרדות שהתמצעו.

סעיף 1

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

**עבור בעיית CVRP**, השלב הראשון מתבצע ע"י השימוש באלגוריתם K-means המותאם לבעיה שלנו, כאשר k הוא מספר הרכבים בבעיה. הרכבים באלגוריתם שלנו מחולקים למספר של קבוצות שהוא עד k, כאשר הערים בקבוצה אחת נכללים באותו מסלול\רכב. 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 בהתחלה רק את העיר. ההחלטה על פתיחת מסלול חדש לעיר מסוימת מתקבל אם המרחק של העיר מהמחסן קטן מכל מרחק ממרכז של אחת הקבוצות (המסלולים) 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. 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.

מנגד, **עבור בעיית פונקציית ה- Ackley**, האתחול הוא אקראי מכיוון שהוא הביא לתוצאות הכי טובות. דרך אחרת שנבחנה היא חלוקת המרחב לתתי מרחב כמספר הפרטים\הפתרונות ובחירת פרט מכל תת-מרחב, מתוך הנחה שזה יביא לגיוון, אולם התוצאות היו פחות טובות בהשוואה לשיטה שאומצה. השלב השני שואף לשפר את הפתרונות ולשם כך נעשה שימוש בחיפוש לוקאלי ע"י יצירת 1000 עותקים של הוקטור ולהוסיף לכל אחד רעש גאוסיאני לכל ממד בווקטור 1000 פעמים, שהטוב מביניהם מחליף את הוקטור הנוכחי.

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:

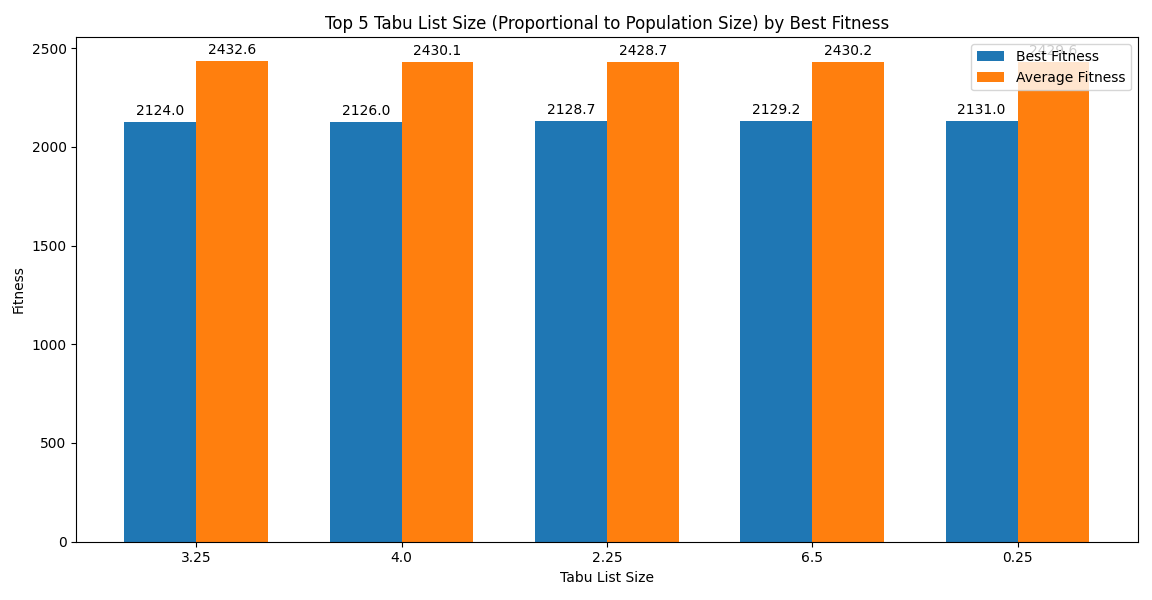
בעיית ה- CVRP משתמשת בשיטות הבאות למציאת שכן:

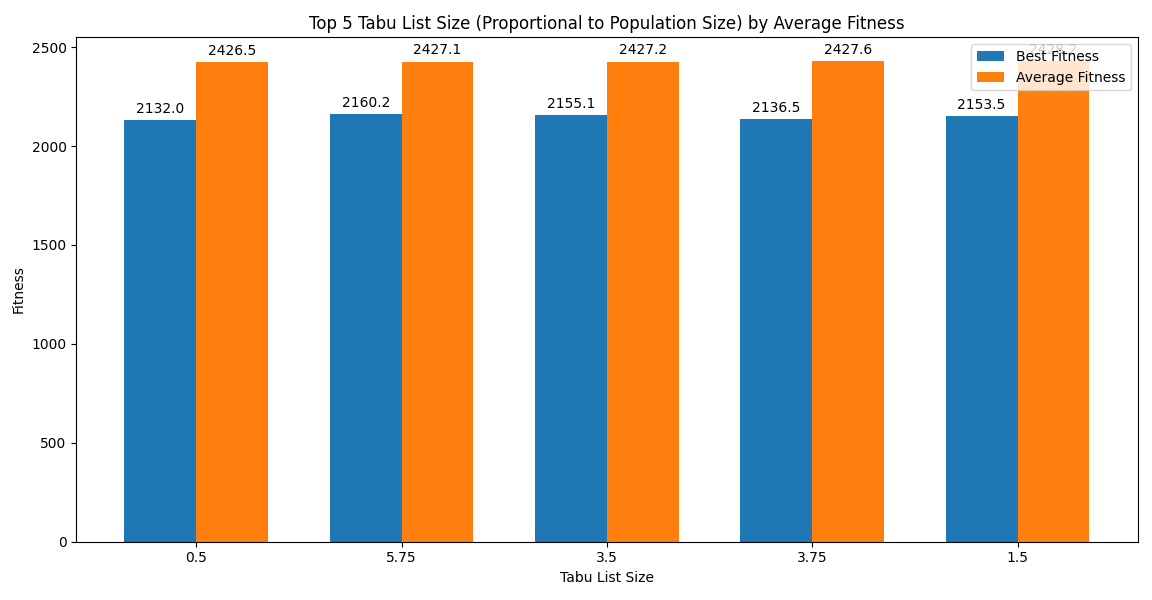
* 2-opt: בוחרת קטע אקראי מתוך מסלול אקראי והופכת אותו.
* relocate: בוחרת עיר באופן אקראי, מוציאה אותה מהמסלול שלה ומכניסה אותה למסלול אקראי אחר. אם המסלול החדש עדיין עומד במגבלת התכולה של הרכב היא מחזירה את השכן.
* reposition: בוחרת עיר ומשנה את המיקום שלה באופן אקראי בתוך המסלול אליו היא שייכת.
* swap: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

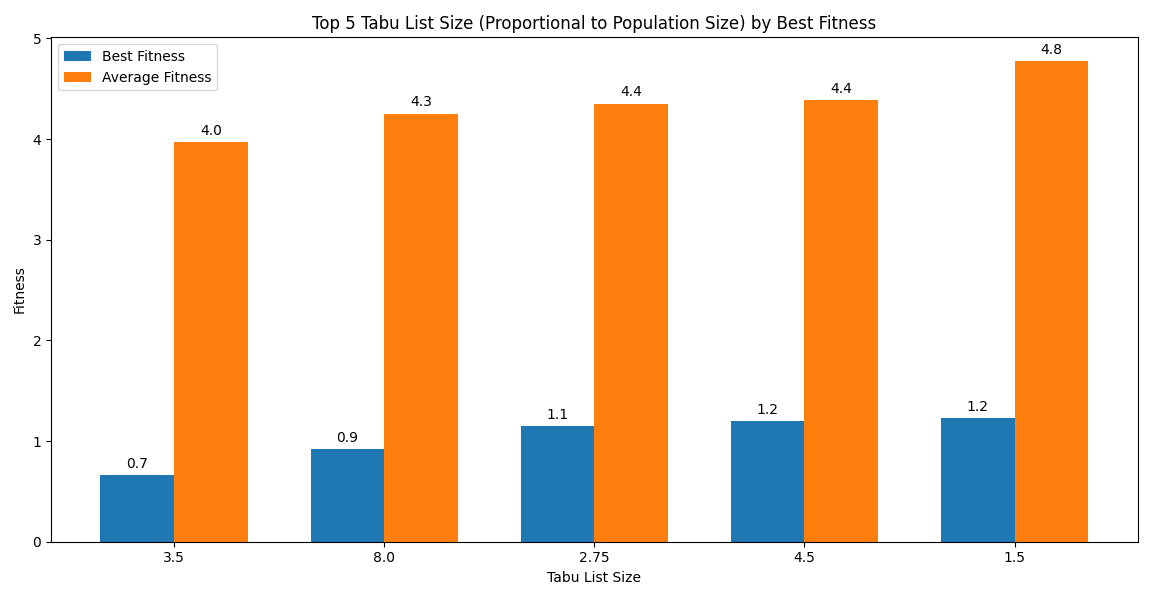
עבור בעיית Ackley, השימוש היה בשיטות הבאות:

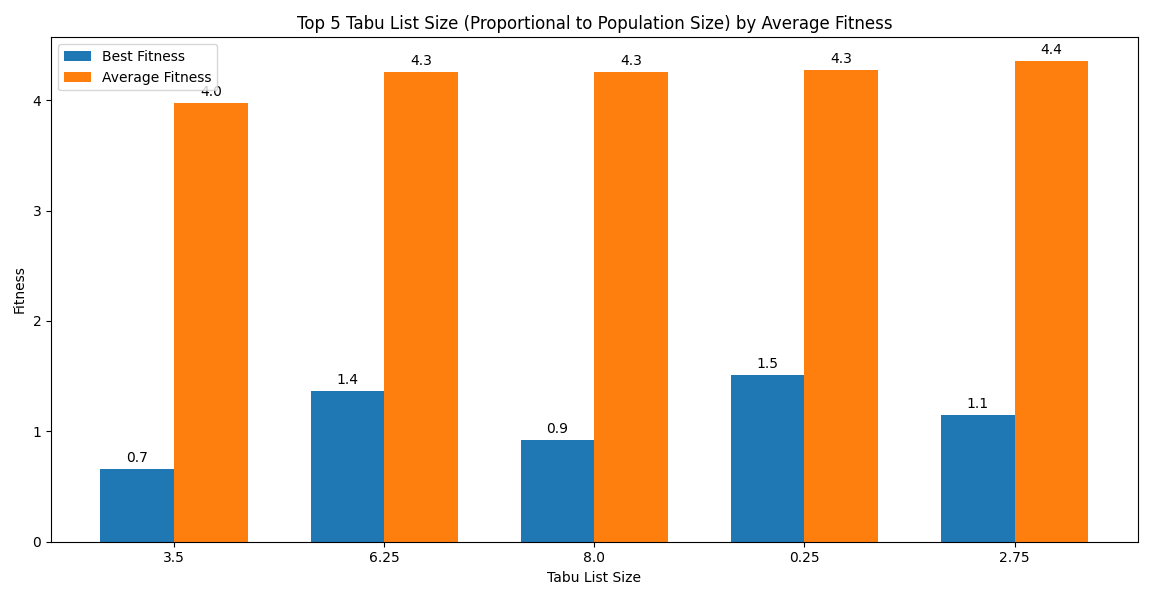
* shift one: הוספת רעש (ערך שבין -0.1 ל- 0.1) לאחד מהממדים הנבחר באופן אקראי.
* shift all: הוספת רעש (וקטור שהערכים שלו בין -0.05 ל- 0.05) לכל אחד מהממדים.
* set random: שינוי הערך של אחד הממדים בערך רנדומלי.

במקרה שהמיטה היוריסטיקה היא Tabu Search, האלגוריתם מאתחל טבלת טאבו בגודל פי 3.5 מגודל האוכלוסייה, ומתחזק אותה בהתאם. הוא מחלץ לכל פרט את השכנים שלו באמצעות כל אחת מהשיטות ובוחר מתוכם את הכי טוב בתנאי שהוא לא נמצא בתוך הטבלה. 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 ההיסטוגרמות הראשונות) היו מאד קרובות וההבדלים לא היו משמעותיים. מנגד, כן נצפו הבדלים במקרה של בעיית Ackley (שתי ההיסטוגרמות האחרונות). הערך נבחר להיות 3.5\* גודל האוכלוסייה נבחר בשל היותו מופיע ב- 3 מ- 4 ההיסטוגרמות, ובמיוחד שהוא נתן התוצאות הכי טובות במקרה של בעיית Ackley:

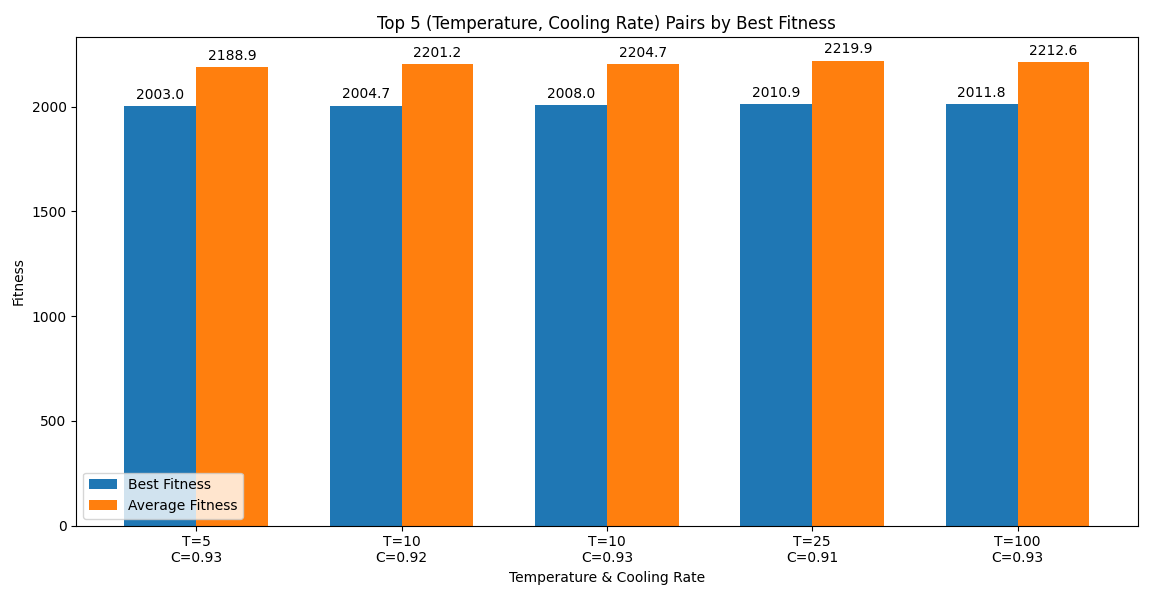


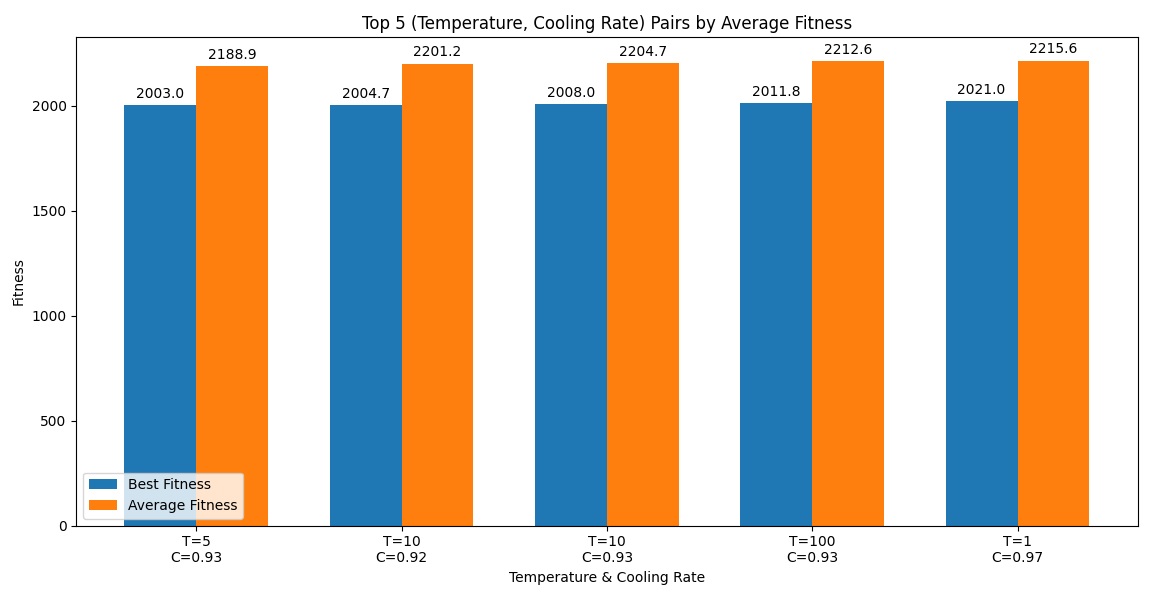


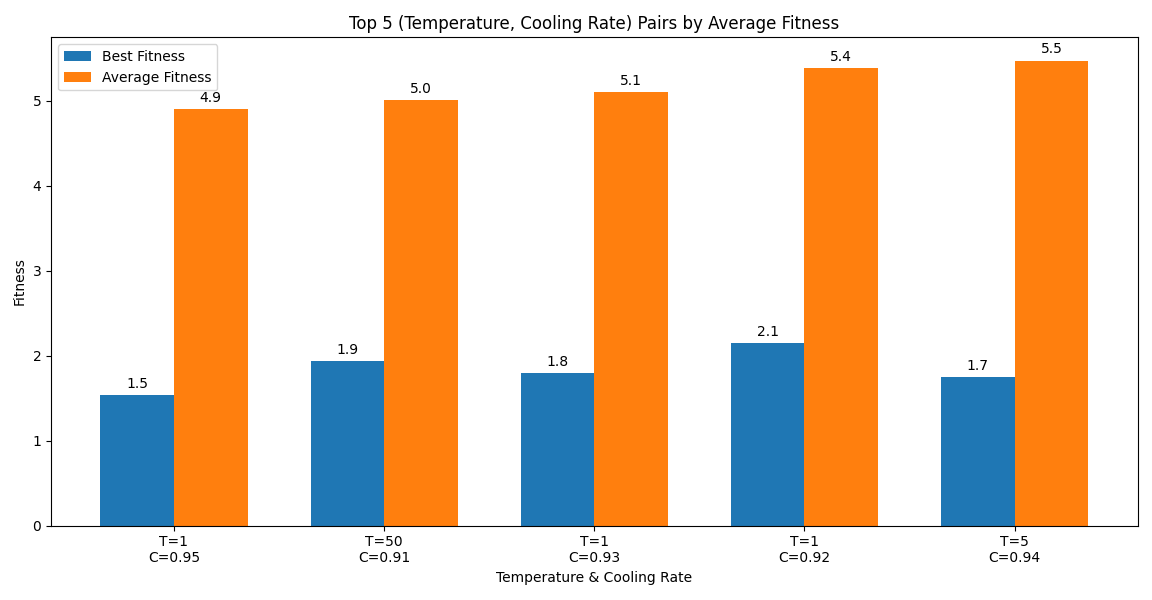


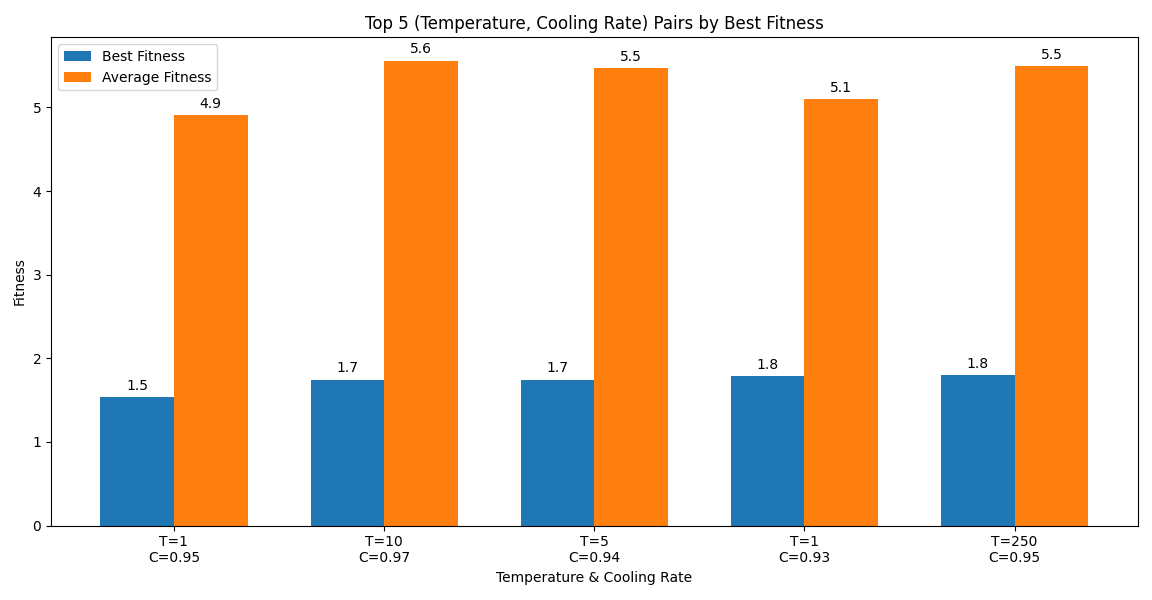


ה- Simulated Annealing במקום זאת בוחרת שיטת שכנות באופן אקראי ומחלצת את השכן על פיה, אם הוא יותר טוב מהפרט הנוכחי היא מאמצת אותו במקומו אחרת, היא מאמצת אותו בהסתברות שהשיטה של Simulated Annealing קובעת. הערכים של ה- Temperature ההתחלתי וה- Cooling rate נבחרו להיות שונים בהתאם לבעיה על סמך התוצאות המצורפות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









היוריסטיקת Discrete Ant Colony Optimization פעולת באופן שונה. כאשר היא מאתחלת את ה- Pheromone להיות 1.0 בין כל שתי ערים 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. הפרמטרים השונים בהם ההיורסטיקה משתמשת (Evaporation rate, Q, α, β) היו בהתאם להשוואה שנעשתה ותוצאותיה מצורפות להלן:

לצרף את התוצאות

סעיף 3

כ

1 K-means iterations: Average value: 2075.03, Average time: 4.32

2 K-means iterations: Average value: 2053.76, Average time: 5.28

3 K-means iterations: Average value: 2069.99, Average time: 3.49

4 K-means iterations: Average value: 2036.90, Average time: 3.33

5 K-means iterations: Average value: 2070.71, Average time: 4.67

6 K-means iterations: Average value: 2078.71, Average time: 3.23

7 K-means iterations: Average value: 2080.62, Average time: 5.92

8 K-means iterations: Average value: 2068.10, Average time: 3.48

9 K-means iterations: Average value: 2054.55, Average time: 3.12

10 K-means iterations: Average value: 2079.68, Average time: 5.61

11 K-means iterations: Average value: 2058.05, Average time: 3.20

12 K-means iterations: Average value: 2079.82, Average time: 4.39

13 K-means iterations: Average value: 2084.95, Average time: 3.06

14 K-means iterations: Average value: 2077.96, Average time: 3.09

15 K-means iterations: Average value: 2074.67, Average time: 3.57

16 K-means iterations: Average value: 2061.61, Average time: 3.65

17 K-means iterations: Average value: 2068.01, Average time: 3.99

18 K-means iterations: Average value: 2077.56, Average time: 4.67

19 K-means iterations: Average value: 2053.23, Average time: 3.61

20 K-means iterations: Average value: 2082.52, Average time: 5.29

21 K-means iterations: Average value: 2055.14, Average time: 4.90

22 K-means iterations: Average value: 2064.24, Average time: 4.20

23 K-means iterations: Average value: 2055.38, Average time: 3.45

24 K-means iterations: Average value: 2071.73, Average time: 4.34

25 K-means iterations: Average value: 2056.06, Average time: 3.89

26 K-means iterations: Average value: 2071.46, Average time: 3.17

27 K-means iterations: Average value: 2068.97, Average time: 3.13

28 K-means iterations: Average value: 2077.48, Average time: 3.08

29 K-means iterations: Average value: 2060.41, Average time: 4.31

30 K-means iterations: Average value: 2062.23, Average time: 4.52

