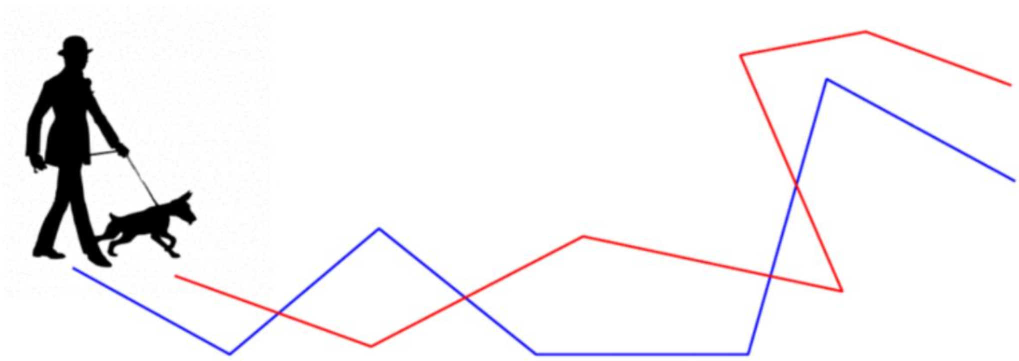
סדנה
לפרויקטים



Fréchet distance

בהנחיית ד"ר שלמה ינץ מאוניברסיטת בר אילן ובשיתוף
ד"ר אלי פקר ראש מחקר אינטל

עמית בן דוד | קים אחרפי | איתי שמעוני

תעודות זהות

326825247 | 207909979 | 305144685

תוכן עניינים

הקדמה על מטרת הפרויקט

בתחילת הסמסטר קיבלנו מאגר מידע ומאמרים מד"ר אלי פקר על חישוב מרחק Fréchet וחישוב המסלול המשותף של שתי עקומות ע"י שימוש במרחק Fréchet

לאחר שחקרנו את הנושא ולמדנו איך מחשבים את המרחק עבור שתי עקומות ואיך מחושב המסלול המשותף בין שתי העקומות ומה הוא מייצג,

קיבלנו מד"ר אלי פקר את המשימה הבאה: **לחשב את המרחק על שלושה עקומות או יותר**

משימה זו הינה חדשנית בעוד שרוב המחקרים עד היום עסקו במרחק בין שתי עקומות בלבד.

תחילה רצינו לחשב את המרחק והמסלול לשתי עקומות על מנת לחקור את הנושא לעומק, כדי לחשב בהמשך את המרחק עבור מספר עקומות כרצוננו.

כפי שניתן יהיה לראות בהמשך, בהסתמך על מאמרים שקראנו מימשנו אלגוריתם למציאת המרחק עבור שתי עקומות על מנת להכליל אותו בהמשך למספר רב יותר של עקומות.

במקביל, קראנו ולמדנו מתוך המאמרים איך לחשב את המסלול המשותף בין שתי העקומות.

בלומר המסלול המשותף הוא המסלול המתאר את הדרך המשותפת שעושים האיש והכלב יחד בהנתן אורך רצועה המתאים להם.

במהלך המחקר נחשפנו לפלטפורמה Fréchet view אשר מחשבת את המסלול ומראה את התוצאות,

ניתן להחשף לקוד המקור של פלטפורמה זו ולהריץ עליה דוגמאות באופן חופשי.

חשוב לשים לב כי חישוב מסלול משותף עבור שלושה עקומות יתן תוצאה מימד 3.

ובאופן כללי חישוב מסלול עבור n עקומות יתן תוצאה ממימד n , לאחר מחקר רב לא מצאנו כי יש שימוש טכנולוגי למסלולים כאלה, המסלול נועד לבטא את הדמיון והשוני בין מסלולי העקומות, עליו נרחיב בפרק על חישוב המסלול המשותף

רקע כללי ושימושים טכנולוגיים

רקע כללי:

גיאומטריה חישובית ממלאת תפקיד חשוב במספר רחב של תחומי מחקר שונים ומגוונים. תוצאות שונות ואלגוריתמים יעילים הכרחיים לצורך חישובים כמו "מיקום נקודה חמה של WI-FI", "ראיית מחשב", "ניתוב מפות", "עיבוד דיבור וכתב יד" ומספר רב של יישומים נוספים.

בעיות רבות נפתרות על ידי התאמה ומציאת מסלול העובר דרך נקודות. הבעיות עוסקות ברוב המקרים או שניתן להפוך אותן לכאלו – בעקומות פוליגונליות שבהן נתמקד בפרוייקט זה. בעיה טבעית שעולה כאשר רוצים להשוות צורות היא בהינתן שתי עקומות: "כמה הן דומות אחת לשניה".

על מנת לענות על השאלה הזאת, נשאל קודם כל "מה הוא המרחק בין 2 העקומות". את המרחק ניתן למדוד באמצעות סוגי מרחק שונים – תלוי באופן השימוש.

דוגמאות למרחקים:

המרחק שגוף מסוים עשה - אורכו של מסלול ספציפי העובר בין שתי נקודות, כמו המרחק שאדם הלך תוך כדי ניווט במבוך.

מרחק "אוקלידי" - אורך המסלול הקצר ביותר האפשרי במרחב, בין שתי נקודות, שניתן היה למדוד ללא "הפרעות".

מרחק גאודזי - אורך המסלול הקצר ביותר בין שתי נקודות תוך שהוא נשאר על פני שטח כלשהו, כמו מרחק המעגל הגדול לאורך עקומת כדור הארץ.

אורכו של מסלול ספציפי החוזר לנקודת ההתחלה, למשל: כדור שנזרק ישר מעלה, או כדור הארץ כשהוא משלים סיבוב אחד.

"מרחק מעגלי" - הוא המרחק שעבר על ידי גלגל, שיכול להיות שימושי בעת תכנון כלי רכב או הילוכים מכניים.

מרחקי מנהטן / גיאומטרית נהגי המוניות -כינוי למרחב מטרי, שבו מודדים את המרחקים על-פי אילוצי הנסיעה של נהג מונית, במקום ב"מעוף הציפור".

מרחק צ'בישב - מרחק השחמט: הוא שווה למספר הצעדים המינימלי שהמלך עושה בלוח שחמט כדי להגיע למשבצת כלשהי (המלך יכול לנוע אופקית, אנכית ובאלכסון). למשל המרחק בין ו5 ל ו1 או ב1 הוא: 4.

מרחקי האסודרוף – המרחק המרבי של קבוצה מנקודה הקרובה ביותר בקבוצה אחרת.

בין כל סוגי המרחקים המצוינים לעיל, אף אחד מהם לא לוקח בחשבון את האוריינטציה של העקומות. למשל, מרחק האוסדורף לוקח בחשבון את קבוצת הנקודות בשתי העקומות אך אינו מתייחס כלל לכיוון העקומה.

בשימושים שונים, כיוון העקומה חשוב, למשל עקומות שנקלטות על ידי דיגיטייזר כמו כתב יד, כאשר הפרמטריזציה של העקומה נתונה כבר על ידי מכשיר הקלט.

כדי להשיב לבעיה זאת, נציג כאן מרחק בשם "**מרחק פרשה**" שפותח על ידי המתמטיקאי מוריס רנה פרשה ועל מרחק זה מבוסס הפרויקט שלנו.

בצורה אינטואיטיבית נוכל לחשוב על מרחק פרשה בצורה הבאה:

נדמיין שאנו מוציאים את הכלב שלנו לטיול, בנינו לבין הכלב מחברת רצועה, נביט במסלול ההליכה שלנו בשתי עקומות שונות, כאשר על אחת מהם הולך כלב ועל השניה הולך אדם בקצב שונה ובניהם הרצועה. מרחק פרשה מוגדר להיות **מרחק הרצועה המינימלי הנדרש לנו**.

מרחק פרשה לוקח בחשבון את הכיוון של שתי העקומות מכיוון שזוגות הנקודות שמרחקן תורם למרחק פרשה מתקדמות ברציפות לאורך העקומה שלהן.

בשנת 1995 המציאו Alt ו Godue אלגוריתם שנועד למצוא פתרון לשאלה זו בזמן פולינומיאלי. בפרויקט שלנו נתבסס על אלגוריתם זה עבור מרחקי פרשה בין 2 עקומות ונפתח את הרעיון למרחקי פרשה עבור n עקומות.

שימושים טכנולוגיים

בין השימושים הטכנולוגיים במרחקי פרשה ניתן למנות: מורפינג (אפקט של שינוי צורה), זיהוי כתב יד, חיזוי מבנה חלבון, עיבוד מקבילי, הזרמת מידע ושיפור איכות תמונה ואיכות אודיו.

זיהוי כתב יד: חישוב תכונות כתב יד מבוסס מרחקי פרשה" מתחיל בחלוקה של דגימת תווים לאזורים מלבניים שונים. ואז מחושבים ערכי המרחקים מכל אזור לכל אזור אחר. בטכניקת חילוץ תכונות מבוססות מרחק גם דגימת תווים מחולקת למספר מקטעים ומרחקים נמדדים מקטע מסוים לכל הקטעים האחרים.

עיבוד מקבילי :עיבוד בו זמנית של של בעיה מסוימת ע"י מספר מעבדים או מספר ליבות, כך המטלה מחולקת בין מספר המעבדים כדי להגיע לתוצאה מהירה יותר . בעזרת מרחקי Fréchet ניתן לחשב את הזמן המינימלי לחלוקת המטלה בין המעבדים השונים .כך למשל אם נרצה לקבוע את הזמן המינימלי לביצוע משימה מסוימת נוכל לדעת האם ניתן לבצע את המשימה לפי כמות המעבדים והתנהגותם.

הזרמת מידע: מכונה גם "סטרימינג" טכנולוגיית אינטרנט להעברת מדיה באופן רצוף ומתמשך ע"י ספק תוכן .שידור מידע ב"סטרימינג" מאפשר האזנה או צפיה בקובץ טרם התקבל כל המידע המרכיב אותו, המדיה מוזרמת בחלקים קטנים .טכנולוגיה זו מביאה לחסכון בכמות התעבורה שעוברת ברשת אל מול איכות הוידיאו המוצג ומאפשרת שידורים חיים באינטרנט או צפייה מקוונת .בעזרת מרחקי frechet ניתן לחשב מסלול טוב לשידור כל פריים – .שיפור איכות תמונה ואיכות אודיו :לאחרונה הציעו שני חוקרים בגוגל מדדים חדשים לאודיו ווידיאו הבנויים על עקרונות מרחקי Fréchet

חיזוי מבנה חלבון: חיזוי מבנה החלבון הוא הסקת המבנה התלת מימדי של חלבון מרצף חומצות האמינו שלו - כלומר חיזוי קיפולו והמבנה המשני והשלישי שלו מהמבנה העיקרי שלו. בעזרת מרחקי frechet ניתן לחשב את המסלולים הטובים ביותר עבור מבנה החלבון ובכך לדעת את מבנו.

רקע מתמטי

עקומה מוגדרת להיות פונקציה (מפה) רציפה $f:[a,b]\rightarrow V$, כך ש - $a,b\in R$

ו$a\leq b$ ו *(V,d)* הינו מרחב מטרי.

מרחק פרשה

נתונות שתי עקומות $f:[a,b]\rightarrow V$ ו- $g:[a',b']\rightarrow V$ מרחק הפרשה שלהן מוגדר להיות

$$\delta_F\ f(f,g)=\inf_{\alpha,\beta\ t\epsilon[0,1]}\max d(f(\alpha(t),g(\beta(t)))$$

כאשר α (ובהתאמה גםβ) הינה פונקציה שרירותית, עולה ורציפה מ$[0,1]$ אל $[a,b]$ ו– $[a',b']$ בהתאמה.

לחישוב מרחקי פרשה בין עקומות שרירותיות, נשתמש בעקומות פוליגונליות.

עקומה פוליגונלית הינה עקומה $P[0,n]\rightarrow V$ כאשר n הינו שלם חיובי כך ש

לכל $i\in\{0,1,\dots,n-1\}$ הצמצום של P על הקטע [i,i+1]

הינו **העתקה אפינית כך ש**

$$P(i+\lambda)=(1-\lambda)P(i)+\lambda P(i+1).$$

כאשר נייצג את העקומות הפוליגונליות B,A ע"י Q,P : אוסף נקודות על העקומות המייצגות אותן בהתאמה [V1,..,Vn],[U1,..,Un]

תרשים "השטח הפנוי" (free space era)

כלי חשוב למרחקי פרשה הינו תרשים השטח הפנוי (שהוצגה עלי ידי אלט וגודאו)

כאשר דיאגרמת השטח הפנוי בין שתי עקומות עבור מרחק נתון **ε** הינו

$$D_\varepsilon(A,B):=\{\,(\alpha,\beta)\in[0,1]^2\mid d(A(\alpha),B(\beta))\leq\varepsilon\,\}$$

כאשר דיאגרמה זו היא תחום דו מימדי המורכב מכל הנקודות בשתי העקומות אשר המרחק בניהם הוא לכל היותר ε

אלגוריתם לחישוב מרחק frechet:

חישוב המרחק בין שתי עקומות:

עבור A,B שתי עקומות פוליגונליות, כלומר מורכבות מקטעים ישרים, נרצה לחשב את מרחק Fréchet ביניהן.

חישוב המרחק עבור שתי העקומות לא דורש מבני נתונים מסובך ומתבצע ע"י תכנות דינאמי.

שיטת התכנון הדינאמי לבניית אלגוריתמים משמשת לפתרון בעיות שאינן ניתנות לפתרון יעיל בשיטת הפרד ומשול נאיבית.

כלומר, לא ניתן לפתור בעיה זו ע"י חלוקתה לתת-בעיות הנפתרות בתורן על ידי חלוקתן לתת-בעיות קטנות אף יותר, עד שמתקבלות בעיות קטנות מספיק שיהיה ניתן לפתור אותן באופן ישיר.

האלטרנטיבה שמציעה שיטת התכנון הדינמי היא פתרון של כל תת-הבעיות בזו אחר זו, כאשר פתרון כל אחת מתת-הבעיות מאוחסן לשימוש עתידי.

נזכר כי לפי ההגדרה:

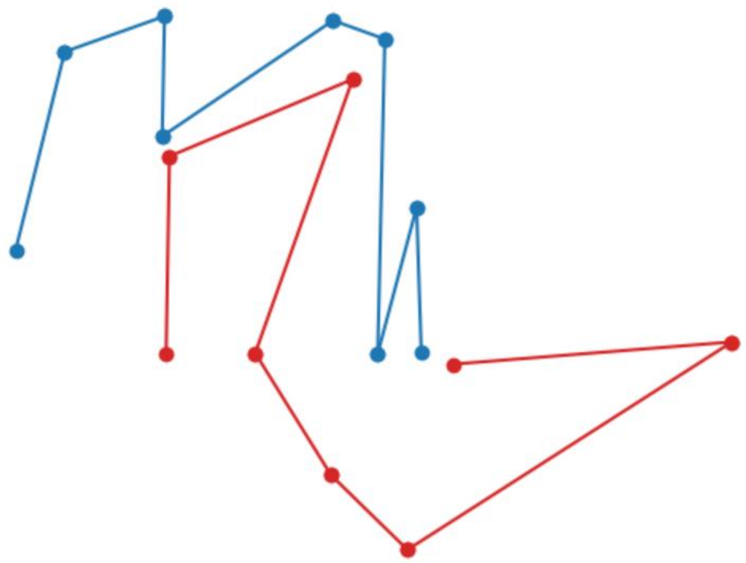
עבור עקומות A,B עם פרמטריזציה β,α מרחק Fréchet מוגדר להיות:

$$F(A,B) = \inf_{\alpha,\beta} \max_{t\in[0,1]} \left\{ d\Big(A(\alpha(t)),\ B(\beta(t)) \Big) \right\}$$

את העקומה הפוליגונלית A נייצג ע"י P : אוסף נקודות על העקומות המייצגות אותן U1,..,Un

את העקומה הפוליגונלית B נייצג ע"י Q : אוסף נקודות על העקומות המייצגות אותן V1,..,Vn

באופן הבא:



את העקומות בציור נייצג ע"י הנקודות המסומנות, כלומר:

P =
[[12.8289,228.248],[24.0543,253.104],[46.8761,257.56],[46.3385,242.508],[85.4033,257.023],[97.2302,254.693],[95.6174,215.27],[104.502,233.594],[105.571,215.419]]
Q =
[[47.2345,215.27],[48,240],[89.8827,249.659],[67.6065,215.331],[84.9682,200.029],[102.413,190.692],[176.614,216.736],[112.902,213.953]]

נחזיק מטריצה דו מימדית עבור המרחקים בין אוסף הנקודות P לבין אוסף הנקודות Q.

נאתחל את המטריצה לערך 1- ונשלח אותה לפונקציה רקוסיבית שתעצור רק לאחר מילוי המטריצה.

בכל שלב באלגוריתם נצטרך לבדוק האם אורך הרצועה הקטן ביותר עד כה גדול או קטן מהמרחק האוקלידי הנוכחי, אם הוא קטן יותר נאלץ להגדיל את אורך הרצועה למרחק הנוכחי.

אם הוא גדול יותר אז הוא אכן אורך רצועה טוב גם עבור המרחק הנוכחי.

את האורך הנבחר נשמור במיקום הנוכחי במטריצה, כך באופן רקורסיבי נקבל במיקום האחרון במטריצה את אורך הרצועה המינימלי (אשר השווה באופן רקורסיבי לרצועות המינימליות באיטרציות הקודמות ברקורסיה)

❖ האלגוריתם נלקח מתוך מאמר המצורף בנספחים

קוד בפייתון: מתוך frechetcalcu.py

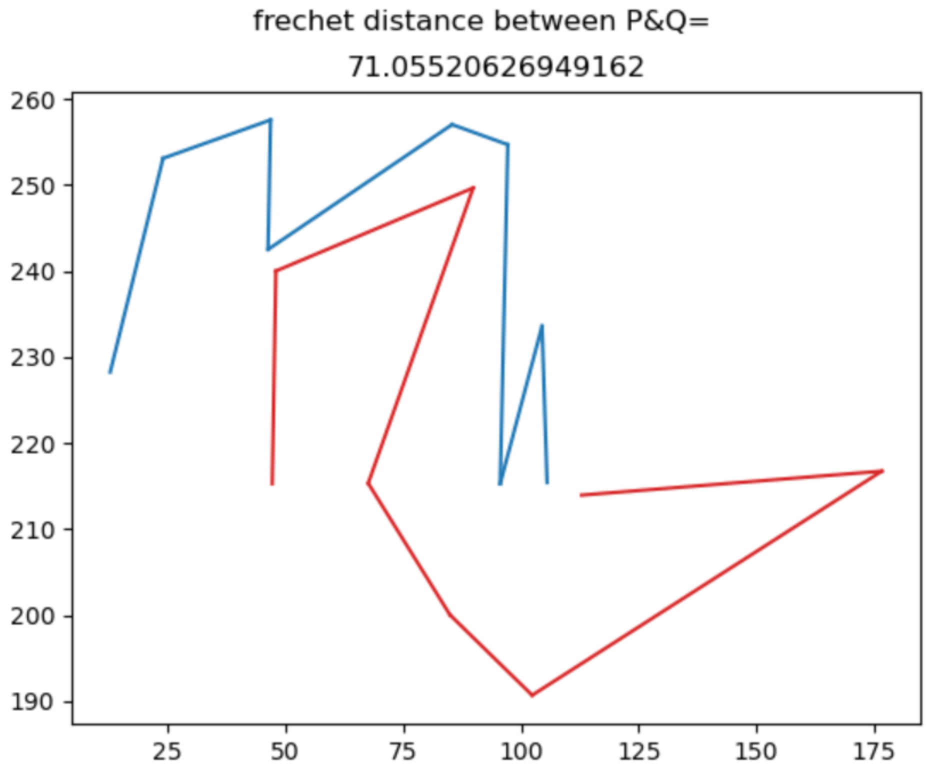
```
import math
import numpy as np
import itertools

def euc_dist(pt1,pt2):
    #  return dist of the 2 point sended
    return math.sqrt((pt2[0]-pt1[0])*(pt2[0]-pt1[0])+(pt2[1]-
pt1[1])*(pt2[1]-pt1[1]))


def _c(ca,i,j,P,Q):
    # i for P j for Q
    if ca[i,j] > -1:
        return ca[i, j],ca
    elif i == 0 and j == 0:
        ca[i,j] = euc_dist(P[0],Q[0])  # calculate the dist between
the last(first) two point of P,Q
    elif i > 0 and j == 0:
        #  first col
        ca[i,j] = max(_c(ca,i-1,0,P,Q)[0],euc_dist(P[i],Q[0]))#take
max between:
        # 1.recalculate _c at index i-1,0 (last distance calcu) -
next untreated point
        # 2.calculate the dist between the one point of Q and next
untreated point of P
    elif i == 0 and j > 0:

        #  first row
        ca[i,j] = max(_c(ca,0,j-1,P,Q)[0],euc_dist(P[0],Q[j]))
    elif i > 0 and j > 0:
        # more then one col and row are filled
        # check neighbors best way, and take the max between the
neighbors best way and the real distance
        # explanation why take the max: (same explanation for all
conditions use max)
        # if distance is smaller then neighbors take the min
neighbors distance - so the leash wont tear for neighbors
        # else if distance is bigger then neighbors take the real
distance - so the leash wont tear for current points
        # to make shore that the path is feasible
        ca[i,j]  = max(min(_c(ca,i-1,j,P,Q)[0],_c(ca,i-1,j-
1,P,Q)[0],_c(ca,i,j-1,P,Q)[0]),euc_dist(P[i],Q[j]))
    else:
        ca[i,j] = float("inf")
    return ca[i,j],ca


def frechetDist(P,Q):
    ca = np.ones((len(P),len(Q)))
    # a matrix of one size: rows- rows of P, col- rows of Q
    ca = np.multiply(ca,-1)
    # same matrix all values double minus one
    return _c(ca,len(P)-1,len(Q)-1,P,Q)
    # send ca and index of the last places at ca with P,Q
```

דוגמא לשימוש בקוד: מתוך inputfinction.py

```
import matplotlib.pyplot as plt
import numpy as np
import sympy as sym
import frechetcalcu

# this class is the main class
# use for input P,Q by user
# use frechetcalcu to calculate the distance

# function draw
# can use for all P matrix
# do not forget to use plt.show() after using this function!
def functionDraw(P,color): # P matrix size 2 x n #color = tab:green
    n = len(P)
    for i in range(1,n):
        plt.plot([P[i-1][0],P[i][0]],[P[i-1][1],P[i][1]],c=color)



if __name__ == '__main__':
# example 1:
    P =
[[12.8289,228.248],[24.0543,253.104],[46.8761,257.56],[46.3385,242.50
8],[85.4033,257.023],[97.2302,254.693],[95.6174,215.27],[104.502,233.
594],[105.571,215.419]]
    Q =
[[47.2345,215.27],[48,240],[89.8827,249.659],[67.6065,215.331],[84.96
82,200.029],[102.413,190.692],[176.614,216.736],[112.902,213.953]]
# example 1:
    #P = [[0,0],[1,1],[2,2],[3,3]]
    #Q = [[0,0.25],[1,1.25],[2,2.25],[3,3.25]]

    functionDraw(P,'tab:blue')
    functionDraw(Q,'tab:red')
    # frechet diatance for 2 curves
    frec_P_Q = frechetcalcu.frechetDist(P,Q)  # frec[0] is the leash
length frec[1] is the distance matrix
    plt.title(frec_P_Q[0])
    plt.suptitle('frechet distance between P&Q=')

    plt.show()
```

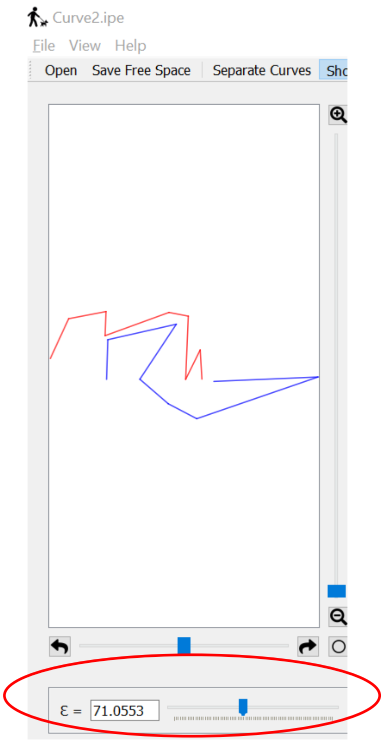
תוצאות:

דוגמא 1:

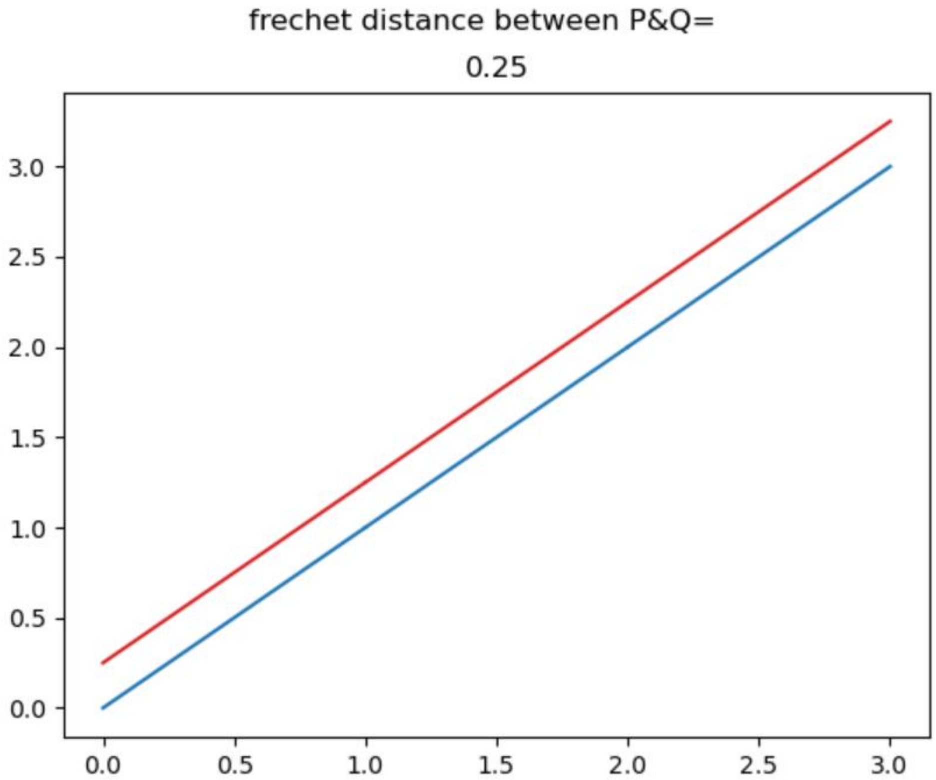


את העקומות בדוגמא זו לקחנו מתוך חישוב במחקר "Fréchet view", ובדקנו את התוצאה שלנו מול התוצאה שלהם.

אכן קיבלנו את אותה התוצאה.מתוך frechet view בהפעלת approximate על אותן עקומות פוליגונליות:



דוגמא 2:



קל לראות שזה אכן המרחק שמתקבל

P = [[0,0],[1,1],[2,2],[3,3]]
Q = [[0,0.25],[1,1.25],[2,2.25],[3,3.25]]

טבלת המרחקים(האוקלידים) ביניהם

0.25	1.6007810593582121	3.010398644698074	4.422951503238533
1.25	0.25	1.6007810593582121	3.010398644698074
2.6575364531836625	1.25	0.25	1.6007810593582121
4.0697051490249265	2.6575364531836625	1.25	0.25

כל עמודה או שורה מכילה את הערך 0.25 לכן מקבלים שזה אורך הרצועה.

למה:

עבור P,Q עקומות פוליגונליות חישוב המרחק ניתן לחישוב ב o(pq) כאשר p הוא מספר הנקודות בפרמטריזציה של P וq הוא מספר הנקודות בפרמטריזציה של Q

- ❖ הוכחת הלמה נמצאת בנספחים

חישוב המרחק עבור יותר מ2 עקומות

לאחר המחקר שביצענו בשלב הראשון של הפרויקט, המשימה שקיבלנו מהמנחה היא למצוא את המרחק עבור שלושה עקומות או יותר.

תחילה על מנת למצוא את החוקיות כדי לבנות מודל המותאם לכל כמות של עקומות מכל סוג בנינו מודל עבור 3 עקומות.

הרחבנו את האלגוריתם עבור שתי עקומות פוליגונליות והוספנו לכל חישוב את האופציות עבור המרחק משתי העקומות גם לעקומה השלישית.

קוד בפייתון: מתוך frechetcalcu.py

```
# calculate frechet multiple distance
# while P is the man and Q,R dogs
# temporary for 3 curve
def _c_multi(ca,i,j,k,P,Q,R):
    # i for P j for Q
    if (int(ca[i,j,k]) > -1):
        return ca[i, j, k],ca
    elif i == 0 and j == 0 and k == 0:
        ca[i,j,k] =
max(euc_dist(P[0],Q[0]),euc_dist(P[0],R[0]),euc_dist(Q[0],R[0])) # calculate
the dist between the last(first) two point of P,Q
    elif i > 0 and j == 0 and k == 0:
        # first col
        ca[i,j,k] = max(_c_multi(ca,i-
1,0,0,P,Q,R)[0],euc_dist(P[i],Q[0]),euc_dist(P[i],R[0]),euc_dist(Q[0],R[0]))
    elif i == 0 and j > 0 and k == 0:
        ca[i,j,k] = max(_c_multi(ca,0,j-
1,0,P,Q,R)[0],euc_dist(P[0],Q[j]),euc_dist(P[0],R[0]),euc_dist(Q[j],R[0]))
    elif i == 0 and j == 0 and k > 0:
        ca[i,j,k] = max(_c_multi(ca,0,0,k-
1,P,Q,R)[0],euc_dist(P[0],Q[0]),euc_dist(P[0],R[k]),euc_dist(Q[0],R[k]))
    elif i > 0 and j > 0 and k == 0:
        ca[i,j,k] = max(min(_c_multi(ca,i-1,j,0,P,Q,R)[0],_c_multi(ca,i-1,j-
1,0,P,Q,R)[0],_c_multi(ca,i,j-
1,0,P,Q,R)[0]),euc_dist(P[i],R[0]),euc_dist(Q[j],R[0]),euc_dist(P[i],Q[j]))
    elif i > 0 and j == 0 and k > 0:
        ca[i,j,k] = max(min(_c_multi(ca,i-1,0,k,P,Q,R)[0],_c_multi(ca,i-
1,0,k-1,P,Q,R)[0],_c_multi(ca,i,0,k-
1,P,Q,R)[0]),euc_dist(P[i],R[k]),euc_dist(Q[0],R[k]),euc_dist(P[i],Q[0]))
    elif i == 0 and j > 0 and k > 0:
        ca[i,j,k] = max(min(_c_multi(ca,0,j-1,k,P,Q,R)[0],_c_multi(ca,0,j-
1,k-1,P,Q,R)[0],_c_multi(ca,0,j,k-
1,P,Q,R)[0]),euc_dist(P[0],R[k]),euc_dist(Q[j],R[k]),euc_dist(P[0],Q[j]))
    elif i > 0 and j > 0 and k > 0:
        ca[i,j,k] = max(min(_c_multi(ca,i-1,j-1,k,P,Q,R)[0],_c_multi(ca,i,j-
1,k,P,Q,R)[0],_c_multi(ca,i-1,j,k,P,Q,R)[0],_c_multi(ca,i-1,j-1,k-
1,P,Q,R)[0],_c_multi(ca,i-1,j,k-1,P,Q,R)[0],_c_multi(ca,i,j-1,k-
1,P,Q,R)[0],_c_multi(ca,i-
1,j,k,P,Q,R)[0]),euc_dist(P[i],R[k]),euc_dist(Q[j],R[k]),euc_dist(P[i],Q[j])
)
    else:
```

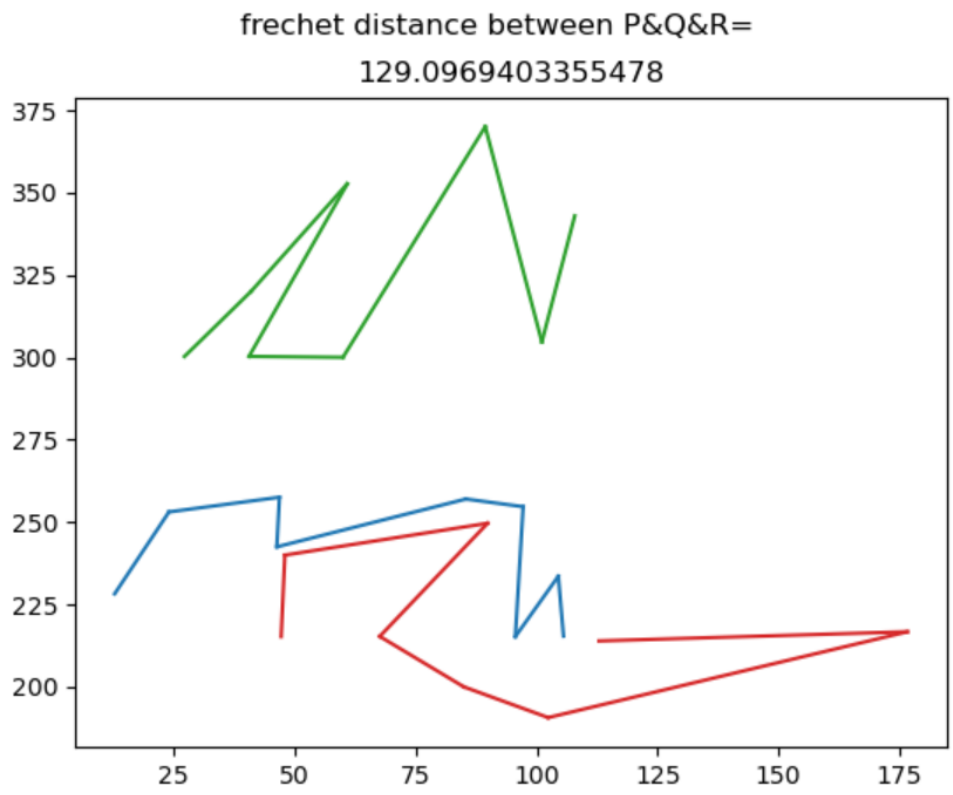
```
        ca[i,j,k] = float("inf")
    return ca[i,j,k],ca

def multiple_frechetDist(P,Q,R):
    ca = np.ones((len(P), len(Q),len(R)))
    # a matrix of one size: rows- rows of P, col- rows of Q
    ca = np.multiply(ca, -1)
    # same matrix all values double minus one
    return _c_multi(ca, len(P) - 1, len(Q) - 1, len(R) - 1,P,Q,R)
    # send ca and index of the last places at ca with P,Q
```

שימוש באלגוריתם על שלושה עקומות פוליגונליות: מתוך inputfinction.py

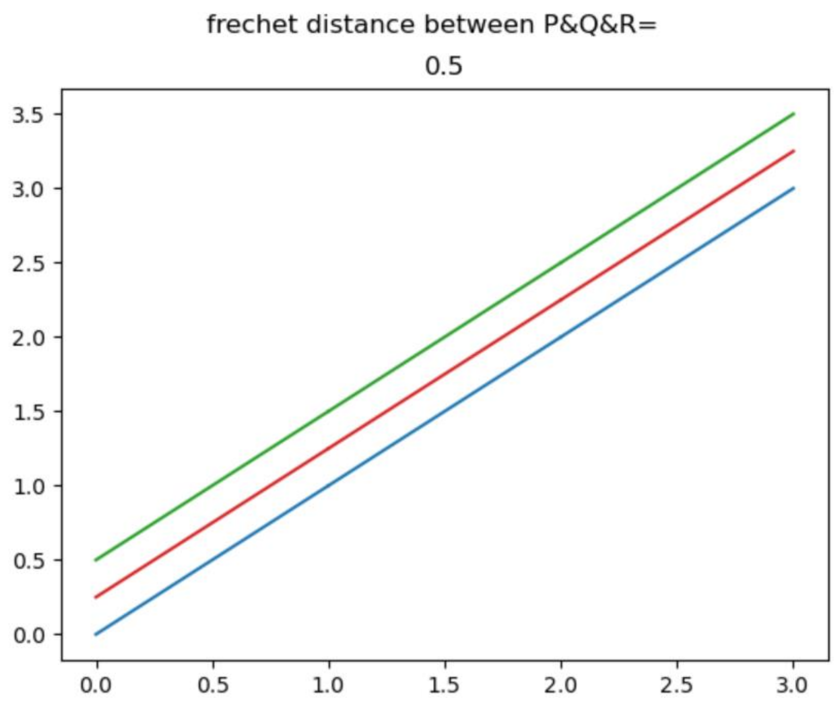
```
P =
[[12.8289,228.248],[24.0543,253.104],[46.8761,257.56],[46.3385,242.508],[85.4033,257.0
23],[97.2302,254.693],[95.6174,215.27],[104.502,233.594],[105.571,215.419]]
Q =
[[47.2345,215.27],[48,240],[89.8827,249.659],[67.6065,215.331],[84.9682,200.029],[102.
413,190.692],[176.614,216.736],[112.902,213.953]]
R =
[[27.25,300.27],[41,320],[60.8827,352.65],[40.6,300.3],[60,300.02],[89.4,370],[101.1,3
04.7],[107.9,342.953]]
#P = [[0,0],[1,1],[2,2],[3,3]]
#Q = [[0,0.25],[1,1.25],[2,2.25],[3,3.25]]
#R = [[0,0.5],[1,1.5],[2,2.5],[3,3.5]]
#example 1/2:
functionDraw(P,'tab:blue')
functionDraw(Q,'tab:red')
functionDraw(R,'tab:green')
# frechet diatance for 3 curves
plt.title(frechetcalcu.multiple_frechetDist_new([P,Q,R])[0])
plt.suptitle('frechet distance between P&Q&R=')
print(frechetcalcu.multiple_frechetDist_new([P,Q,R])[0])
print(frechetcalcu.multiple_frechetDist(P,Q,R)[0])
plt.figure()
#example 3:
functionDraw(P,'tab:blue')
functionDraw(Q,'tab:red')
functionDraw(R,'tab:green')
functionDraw(T,'tab:brown')
# frechet diatance for 4 curves
plt.title(frechetcalcu.multiple_frechetDist_new([P,Q,R,T])[0])
plt.suptitle('frechet distance between P&Q&R&T=')
plt.show()
```

תוצאות:

דוגמא 1:



דוגמא 2:



מהוכחה שנמצאת בבליוגרפיה נובע שעבור דוגמא 2 מרחק Fréchet בין שלושת העקומות שווה לסכום מרחקי הפרשה של העקומה P מהעקומות R וQ ואכן המרחק שמתקבל שווה לסכום המרחקים.

המרחק שהתקבל מהווה רצועה מספקת עבור כל אחד מהמרחקים בין כל שתי עקומות ובנוסף מהווה את מרחק הfrechet עבור שלושת העקומות יחד.

ההוכחה הזאת לא נכונה עבור דוגמא 1(לא מקיימת את ההנחות)

עם זאת, עדיין ניתן לראות שזה המרחק הטוב ביותר עבור שלושת העקומות,

כי המרחק בין P לQ הוא 71.055 המרחק בין Q לR הוא 129.09 והמרחק בין P לR הוא 127.55

בנוסף ניתן לראות כי 129.09 < 71.055+127.55 כלומר סכום המרחקים לא מהווה את הרצועה המינימלית שאפשר לקחת עבור שלושת העקומות.

סה"כ המרחק המינימלי המתאים לשלושת העקומות שווה למרחק הפרשה בין Q לR על כן התוצאה שקיבלנו עבור שלושתם הינה 129.09

חישוב המרחק עבור n עקומות

על מנת להכליל את האלגוריתם ל n עקומות יש למצוא את אוסף כל הקומבינציות של נקודות מכל אחת מהעקומות.

המודל שלנו יכיל סט של כמות הנקודות בכל עקומה, וסט של עקומות.

כלומר עלינו למצוא את אוסף כל הקומבינציות של אינדקסים כאשר כל קומבינציה מכילה כמות אינדקסים ככמות העקומות הקיימות.

בעזרת קומבינציות אלה נוכל להכליל את האולגוריתם.

למציאת הקומבינציות השתמשנו בספרייה של פייתון itertools בפונקציית product,

פונקציה זו מחזירה את המוצר הקרטזיאני של שתי קבוצות.

במונחים של מתמטיקה מוצר קרטזיאני של שתי קבוצות, מוגדר כסט של כל הזוגות המסודרים (a, b) שבהם a שייך ל- A ו- b שייך ל B. למשל:

$Input:$

$arr1 = [1,2,3]$
$arr2 = [5,6,7]$
$Output:$

$$[(1,5),(1,6),(1,7),(2,5),(2,6),(2,7),(3,5),(3,6),(3,7)]$$

עבור האינדקסים i,j נרצה לחשב את הקומבינציות (i-1,j-1),(I,j-1),(i-1,j),(I,j)

לכן נשתמש בפונקנציה עבור המערכים arr1=[j,j-1], arr2=[i,i-1] ונקבל את הקומבינציות הנ"ל.

על מנת לקבל את הקומבינציות הרצויות גם עבור מערך אינדקסים, נשתמש בפונקצייה product עבור כל זוג אינדקסים, על כל אוסף קומבינציות של האינדקסים נפעיל שוב את הפונקציה לקבלת כל האופציות האפשרויות (בהתחשבות במצב של כמות אינדקסים אי זוגית)

מתוך הקומבינציות שמצאנו לקחנו רק את הקומבינציות המכילות אינדקסים חוקיים בלבד (חיוביים).

על כל קומבינציה חוקית הפעלנו את הרקורסיה, שמרנו את תוצאת הרקורסיה המינימלית, כאשר בפונקציה עצמה לקחנו את המקסימום בין - הרקורסיה המינימלית שקיבלנו עד כה, לבין המרחק האוקלידי הנוכחי בין הנקודות.

כך התוצאה שנקבל תהווה את אורך הרצועה המינימלי המתאים לכל העקומות, לפי ההגדרה.

כמובן שיתכן שיהיו עקומות מתוך האוסף , שביניהם מרחק Fréchet קטן יותר ממרחק ה frechet המשותף, אך מרחק זה יהיה המרחק הקטן ביותר המתאים לכל המרחקים בין כל עקומה ועקומה.

בכך כאשר אדם יצא לטיול עם n-1 כלבים הוא יוכל לדעת מראש את אורך הרצועה "שתספיק לו" על מנת ללכת במסלול משותף עם כל הכלבים

בנוסף רצועה זו תתאים למסלול בין האדם לבין כל כלב וגם בין כל כלב לכלב (גם אם אינה מינימלית)

קוד בפייתון: מתוך frechetcalcu.py

```
def euc_dist(pt1,pt2):
    #  return dist of the 2 point sended
    return math.sqrt((pt2[0]-pt1[0])*(pt2[0]-pt1[0])+(pt2[1]-
pt1[1])*(pt2[1]-pt1[1]))


def max_euc_dist(curves,index):
    max_arr = []
    for i in range(len(curves)):
        if i == len(curves)-1:

max_arr.append(euc_dist(curves[i][index[i]],curves[0][index[0]]))
        else:

max_arr.append(euc_dist(curves[i][index[i]],curves[i+1][index[i+1]]))
    return max(max_arr)


def combi(index):
    combindex = []
    finalcombi = []
    all_combi = []
    for i in np.arange(0, len(index), 2):
        if (len(index)) % 2 == 0 or i != (len(index) - 1):
            temp_combi = []
            for j in itertools.product([index[i], index[i] - 1],
[index[i + 1], index[i + 1] - 1]):
                temp_combi.append(j)
            combindex.append(temp_combi)
    for i in itertools.product(*combindex):
        all_combi.append(i)
    for arr_tuple in all_combi:
        arr_combi = []
        for t in arr_tuple:
            arr_combi.append(t[0])
            arr_combi.append(t[1])
        finalcombi.append(arr_combi)
    if (len(index)) % 2 != 0:
        finalcombi_temp = []
        for k in finalcombi:
            temp1,temp2 = k.copy(),k.copy()
            temp1.append(index[int(len(index)) - 1])
            temp2.append(index[int(len(index)) - 1] - 1)
            finalcombi_temp.append(temp1)
            finalcombi_temp.append(temp2)
        finalcombi = finalcombi_temp.copy()
    if index in finalcombi:
        finalcombi.remove(index)
    validcombi = []
    for i in finalcombi:
        if -1 not in i:
            validcombi.append(i)
    return validcombi


#for multi curves
def _c_multi_new(ca,index,curves):
    # i for P j for Q
    if int(ca[tuple(index)]) > -1:
```

```
        return ca[tuple(index)],ca
    elif np.count_nonzero(index) == 0:
        ca[tuple(index)] = max_euc_dist(curves,index) # calculate the
dist between the last(first) two point of P,Q
    elif combi(index):
        temp_min = []
        for current_indexes in combi(index):

temp_min.append(_c_multi_new(ca,current_indexes,curves)[0])
        rec_min = min(temp_min)
        ca[tuple(index)] = max(rec_min,max_euc_dist(curves,index))
    else:
        ca[tuple(index)] = float("inf")
    return ca[tuple(index)],ca


def multiple_frechetDist_new(curves):
    length = []
    index = []
    for i in curves:
        length.append(len(i))
        index.append(len(i)-1)
    ca = np.ones(length)
    # a matrix of one size: rows- rows of P, col- rows of Q
    ca = np.multiply(ca, -1)
    # same matrix all values double minus one
    return _c_multi_new(ca,index,curves)
    # send ca and index of the last places at ca with P,Q
```

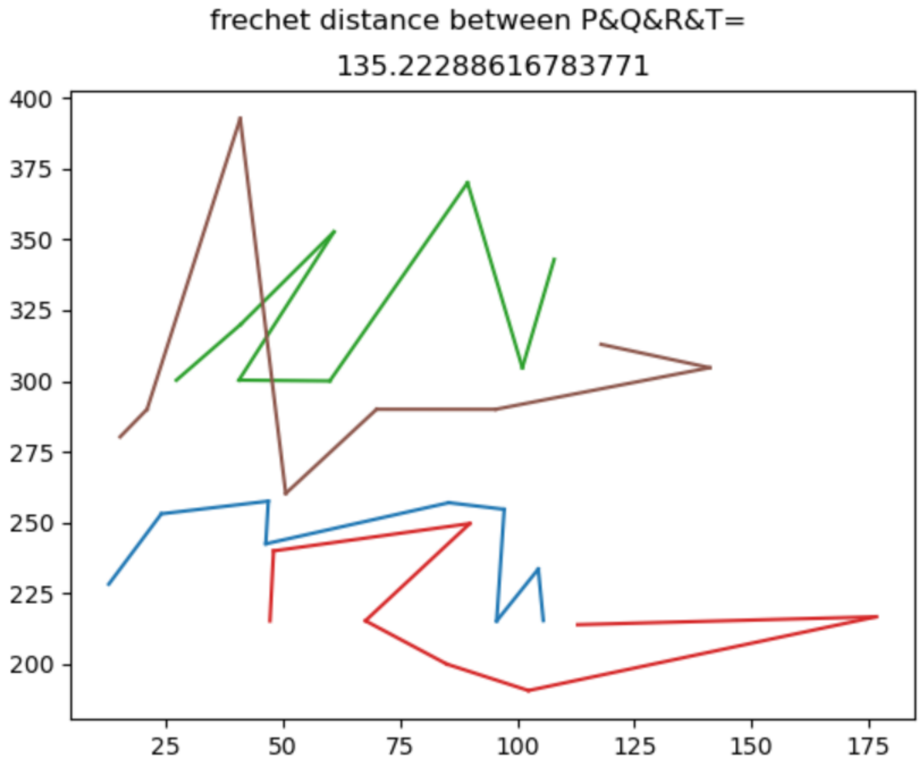
דוגמא לשימוש בקוד: מתוך inputfinction.py

```
#P =
[[12.8289,228.248],[24.0543,253.104],[46.8761,257.56],[46.3385,242.508],[85.4033,257.0
23],[97.2302,254.693],[95.6174,215.27],[104.502,233.594],[105.571,215.419]]
#Q =
[[47.2345,215.27],[48,240],[89.8827,249.659],[67.6065,215.331],[84.9682,200.029],[102.
413,190.692],[176.614,216.736],[112.902,213.953]]
#R =
[[27.25,300.27],[41,320],[60.8827,352.65],[40.6,300.3],[60,300.02],[89.4,370],[101.1,3
04.7],[107.9,342.953]]
#T =
[[15.25,280.27],[21,290],[40.8827,392.65],[50.6,260.3],[70,290.02],[95.4,290],[141.1,3
04.7],[117.9,312.953]]

P = [[0,0],[1,1],[2,2],[3,3]]
Q = [[0,0.25],[1,1.25],[2,2.25],[3,3.25]]
R = [[0,0.5],[1,1.5],[2,2.5],[3,3.5]]

functionDraw(P,'tab:blue')
functionDraw(Q,'tab:red')
# frechet diatance for 2 curves
frec_P_Q = frechetcalcu.multiple_frechetDist_new([P,Q])  # frec[0] is the leash lenght
frec[1] is the distance matrix
plt.title(frec_P_Q[0])
plt.suptitle('frechet distance between P&Q=')
plt.figure()
functionDraw(P,'tab:blue')
functionDraw(Q,'tab:red')
functionDraw(R,'tab:green')
# frechet diatance for 3 curves
plt.title(frechetcalcu.multiple_frechetDist_new([P,Q,R])[0])
plt.suptitle('frechet distance between P&Q&R=')
plt.show()
```

תוצאות

עבור דוגמאות 1-2 קיבלנו את אותן התוצאות בידיוק

דוגמא עבור 4 עקומות: (P,Q,R,T)



חישוב המסלול המשותף של העקומות

במהלך המחקר השתמשנו בבלי שנקרא Fréchet View. השתמשנו בבלי כדי להבין טוב יותר את מרחק Fréchet בכללותו ולייצוגים גרפיים יותר של מרחקי העקומות.

אנו יכולים לקבל "מסלול אפשרי", מסלול המתאר את הדרך המשותפת שלהם ומייצג את הדמיון בין העקומות, לאחר שיצרנו תצוגת Fréchet לעקומות שלנו.

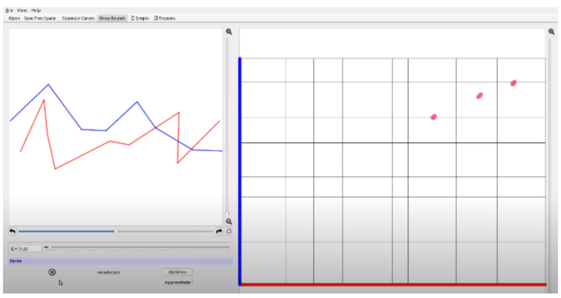
לקבלת תצוגת Fréchet View מתחילים בלקיחת שתי עקומות P ו- Q ופירוקן למרחקיהן בין כל שתי נקודות בכל עקומה:

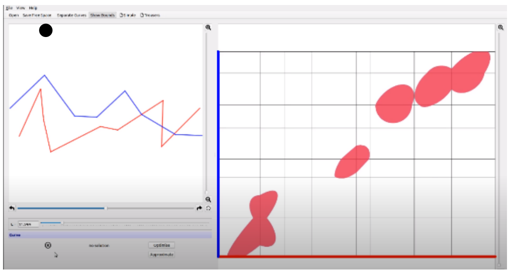
מרחקים אלו יהיו ציר ה- X וציר ה- Y (שנבחר באופן שרירותי) עבור גרף חדש, אפשר לדמיין שלקיחת פונקציה מחלקת אותה לקו ישר ומשתמשים בו כאחד הצירים בגרף החדש.

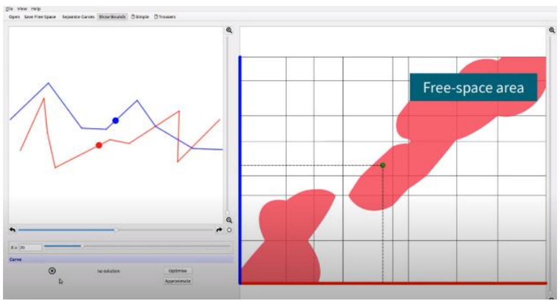
בשלב הבא נעסוק במרחק Fréchet בין העקומות, החל מ- 0 כמרחק הרצועה עד מרחק Fréchet. הגדלנו את מרחק Fréchet באופן הדרגתי בין שתי העקומות וזה גרם לתחום המכיל את המרחק המתאם שלהן (בגרף החדש, או בערך ה-x בפונקציה הישנה) לגדול בגרף החדש שלנו. לדוגמה, אם נקרא למרחק "D" אנו רואים בגרף הראשון, כאשר D קטן מאוד המקומות שבהם הגרף מתחיל להתמלא מתחילים בנקודות בהן הגרפים נחתכים. בגרפים הבאים אנו רואים שהגרף מתחיל להתמלא יותר, אנו מגדילים את D ומוצאים מקומות נוספים שאפשר להגיע אליהם עם ה- D לאחר ההגדלה.

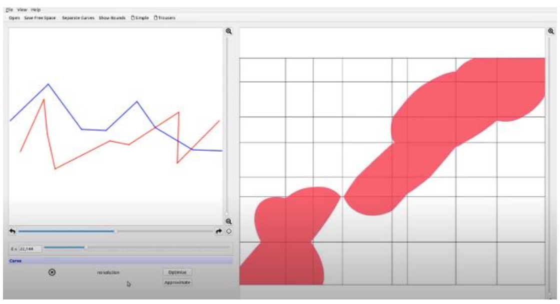
לסיכום, כל האזורים ש- D יכול להגיע אליהם או שכבר הצליחו להגיע "מתרחבים" בתרשים וממשיכים "להתרחב".

כל מסלול אשר כולו נמצא בתוך התחום D יחשב "מסלול אפשרי"









התהליך מסתיים בדרך כלל כאשר הקבוצות של אזורי ה- Fréchet View האדומים הפכו לקבוצה אחת עם נקודת ההתחלה (0,0) ונקודת הסיום (n, m) (הפינה השמאלית התחתונה והפינה הימנית העליונה).

תהליך זה תואר לעיתים קרובות כהוספת "טיפות מים" למקומות בהם המרחק שווה או גדול יותר, ובתורו גורם למים להתפשט סביב הגרף.

כאשר Fréchet View מסתיים, אנו יכולים להתמקד במסלול האפשרי.

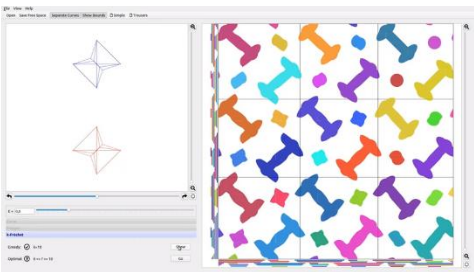
מסלול אפשרי במקרה זה הוא הרגע בו הושלם Fréchet View, (נקודות התחלה וסיום נמצאות באותה קבוצה). הגרף הופך לגרף עם כל הנקודות בהן שתי הפונקציות נמצאות במרחק של d(המרחק המקומי בין העקומות) או פחות, כלומר הגרף הפך לגרף של כל הנקודות בהן "האדם והכלב" במרחק Fréchet או פחות.

כל התחום האדום בתרשים הוא תחום אפשרי למספר רב של מסלולים אפשריים המתנהגים בצורות שונות, למשל:

- נניח שנרצה שהאיש (שהולך עם כלבו) יגיע קודם לסוף, הוא היה לוקח את הנקודות הקרובות ביותר לציר y והוא יגיע קודם לסוף, אם זה אפשרי (אותו דבר הפוך).
- נניח שיש מקום שאתה מעוניין שהאיש ישהה למשל "חנות" בדרך. אתה פשוט מוסיף את מיקום החנות בפונקציה עצמה (P או Q במקרה זה). אז הם היו נשארים שוב על הנקודות הקרובות ביותר לציר y עד שיגיעו ל"חנות" ומשם ממשיכים רק עם הכלב בשביל (הולכים במקביל לציר ה- x). זה יאפשר לאיש לשהות הכי הרבה זמן בחנות כשהוא מוריד את הכלב מדרכו הרצויה (הרצועה תהיה מספיק ארוכה גם בשיטה זו).
- אנו יכולים לבחור נתיב עם המרחק המינימלי המשמש בפונקציות שלנו או המרחק המרבי, זה שימושי יותר לניתוח נתונים ולבדיקות (לוודא שהדברים מנוצלים בצורה אופטימלית)

Fréchet view מעניק לנו ייצוג ויזואלי מעניין למרחקים בין העקומות שלנו, המתאר את הדימיון בין העקומות, אך זה לא מאוד שימושי כשמדובר ב -4 עקומות או יותר.

בנושא המחקר העיקרי שלנו, בגלל מספר הצירים, התוצאה תיהיה צורות במספר רב של מימדים, זה ללא ספק יצר מחזה ויזואלי מקסים עם כמה עקומות שאנו ממליצים לכל מי שיחקור אותו בעתיד, שינסה לראות.



מסקנות

כיום כפי שהזכרנו משתמשים באלגוריתמים לחישוב מרחק Fréchet של עקומות לשימוש בחישוב במקביל, כלומר עיבוד בו זמנית של בעיה מסוימת ע"י מספר מעבדים או מספר ליבות על מנת להגיע לתוצאה מהירה יותר.

דוגמא זו ודוגמאות נוספות שהזכרנו מהוות מוטיבציה למציאת המרחק עבור n עקומות.

הפתרון שהצגנו עבור בעיה זו הינו פתרון למקרה הבדיד, אך פתרון הבעיה עבור המקרה הרציף יהיה שימושי בתחומים נוספים שהזכרנו, למשל עבור הזרמת מידע (streaming).

ראינו כי לפתרון שמצאנו עלות חישוב גבוה.
עלות החישוב הממוצעת של מציאת המרחק עבור n עקומות, כאשר נסמן את גודל העקומה ה m ב nm , העלות עבור m עקומות הינה:

O(n1*…* nm*log(n1*…* nm*)*m)

אלגוריתם החישוב עבור שתי עקומות מתבצע באופן רקורסיבי, לכן עלות החישוב עבור n עקומות, המתבסס עליו, הינה יקרה מכיוון שהחישוב מתרחש באופן רקורסיבי עבור מספר רב של נקודות.

חישוב מרחק Fréchet מתבצע כך שבכל שלב מחשבים את המרחק בין העקומות עד כה וכך בוחרים את מרחק Fréchet המתאים לכל הנקודות עד שלב זה.
לכן כדי לחשב את המרחק יש לפתור את הבעיה על ידי חלוקתה לתת בעיות הנפתרות בתורן על ידי בעיות קטנות יותר.
לא ניתן לחשב את המרחק על ידי חישוב נאיבי בשיטת הפרד ומשול ולכן יש צורך בתכנון דינאמי הדורש רקורסיה.
כתוצאה מכך חישוב המרחק למספר רב של עקומות הוא בעל עלות חישוב יקרה.

עם זאת, החיסכון שיכול להעניק מרחק Fréchet בשימושים רבים, יכול "לפצות" על העלות הגבוה שלו, למשל בחזרה לעיבוד במקביל, ניתן לקצר תהליכים שקורים בכמה מעבדים במקביל כך שישתלם להשתמש במרחקי Fréchet למרות עלות החישוב הגבוה.

מרחקי Fréchet מתארים את הדימיון בין העקומות ועל כן תורמים במקרים בהם יש להשוות בין פעולות שונות שקורות במקביל ומתוארות ע"י עקומות.
עלות החישוב עבור מספר רב של עקומות גבוה ועל כן יש להשתמש במרחקי Fréchet בצורה התואמת את משמעות המרחק.

ביבליוגרפיה

Sariel Har-Peled; Fréchet distance: How to walk your dog.
https://sarielhp.org/book/chapters/frechet.pdf

Peter Schäfer; Fréchet View
https://hrimfaxi.bitbucket.io/fv/

Frechet view-Peter Schafer
https://www.youtube.com/watch?v=DqlyPbAhKk8

Helmut Alt; Computing the Fréchet Distance between Two Polygonal Curves.
https://www.researchgate.net/publication/220669649_Computing_the_Frechet_Distance_between_Two_Polygonal_Curves

Ning Guo ID, Mengyu Ma, Wei Xiong , Luo Chen , Ning Jing ; An Efficient Query Algorithm for Trajectory Similarity Based on Fréchet Distance Threshold.
https://www.mdpi.com/2220-9964/6/11/326

Adrian Dumitrescu, Günter Rote; On the Fréchet distance of a set of curves.
https://www.researchgate.net/publication/220991483_On_the_Frechet_distance_of_a_set_of_curves#:~:text=In%20the%20Fr%C3%A9chet%20distance%20of,the%20other%20curve.%20...

*הוכחת הלמה נמצאת במקור הנ"ל

Ambika Choudhury; DEV Corner: Google Used Fréchet Distance To Assess AI-Generated Audio & Video Quality
https://analyticsindiamag.com/google-used-frechet-distance-to-assess-ai-generated-audio-video-quality/

Thomas Eiter, Heikki Mannila; Computing Discrete Fréchet Distance
http://www.kr.tuwien.ac.at/staff/eiter/et-archive/cdtr9464.pdf

Wikipedia links ;

- https://he.wikipedia.org/wiki/%D7%94%D7%96%D7%A8%D7%9E%D7%AA_%D7%9E%D7%93%D7%99%D7%94
- https://he.wikipedia.org/wiki/%D7%A2%D7%99%D7%91%D7%95%D7%93_%D7%9E%D7%A7%D7%91%D7%99%D7%9C%D7%99

נספחים

האלגוריתמים של Alt & Godau’

Algorithm 1:

for each feasible pair (i,j) **do** compute L_{ij}^F and B_{ij}^F;
for $i := 1$ **to** p **do** determine $B_{i,1}^R$;
for $j := 1$ **to** q **do** determine $L_{1,j}^R$;
for $i := 1$ **to** p **do**
    for $j := 1$ **to** q **do**
        construct $L_{i+1,j}^R$ and $B_{i,j+1}^R$ from L_{ij}^R, B_{ij}^R, $L_{i+1,j}^F$, $B_{i,j+1}^F$;
answer "yes" if $(p,q) \in L_{p+1,q}^R$ "no" otherwise.

Algorithm 2:

1. Determine all critical values of ε.
2. Sort them.
3. Do a binary search on the sorted sequence in each search step solving the decision problem, continuing with the half containing smaller critical values if it has a positive answer and with the half containing larger critical values otherwise.

Algorithm 3:

1. Determine all critical values of ε of types a) and b) and apply onto them the technique of Algorithm 2. This gives two values ε_1, ε_2 with $\delta \in [\varepsilon_1, \varepsilon_2]$ and $\varepsilon \notin [\varepsilon_1, \varepsilon_2]$ for any critical value ε of type a) or b) other than ε_1 or ε_2.
2. Let A be the set of endpoints $a_{ij}, b_{ij}, c_{ij}, d_{ij}$ of intervals L_{ij}^F or B_{ij}^F that are nonempty for $\varepsilon \in [\varepsilon_1, \varepsilon_2]$ (A is determined by the critical values ε of Step 1 with $\varepsilon \leqslant \varepsilon_1$). Use Cole's variant of parametric search based on sorting the values in A to find the actual value of δ.

האלגוריתם של עליו התבססנו בחישוב המרחק בין שתי עקומות, של Eiter & Manilla

Function dF(P, Q)**: real;**

input: polygonal curves $P = (u_1, \ldots, u_p)$ and $Q = (v_1, \ldots, v_q)$.
return: $\delta_{dF}(P, Q)$

ca : **array** $[1..p, 1..q]$ **of real;**

function $c(i, j)$**: real;**

begin
 if $ca(i, j) > -1$ **then return** $ca(i, j)$
 elsif $i = 1$ **and** $j = 1$ **then** $ca(i, j) := d(u_1, v_1)$
 elsif $i > 1$ **and** $j = 1$ **then** $ca(i, j) := \max\{\, c(i-1, 1),\ d(u_i, v_1)\,\}$
 elsif $i = 1$ **and** $j > 1$ **then** $ca(i, j) := \max\{\, c(1, j-1),\ d(u_1, v_j)\,\}$
 elsif $i > 1$ **and** $j > 1$ **then** $ca(i, j) :=$
  $\max\{\, \min(c(i-1, j),\ c(i-1, j-1),\ c(i, j-1)),\ d(u_i, v_j)\,\}$
 else $ca(i, j) = \infty$
return $ca(i, j)$;
end; /* function c */

begin
 for $i = 1$ **to** p **do for** $j = 1$ **to** q **do** $ca(i, j) := -1.0$;
 return $c(p, q)$;
end.

המאמר שהתבססנו עליו עבור חישוב מרחק בין שלושה עקומות או יותר באמצעות סכום מרחקי הפרשה בניהם, בנוסף הוכחת הלמה נמצאת גם כן במאמר זה

Theorem 1 *Consider a finite set of $m \geq 3$ curves in arbitrary dimensions. Let $d_{ij} := \delta_F(f_i, f_j)$ and $d_{\mathcal{F}} := \delta_F(\mathcal{F})$. Then*

$$d_{\mathcal{F}} \leq \min_{1 \leq i \leq m} \max_{1 \leq j < k \leq m} (d_{ij} + d_{ik}).$$

This inequality is best possible: already for $m = 3$, and for any choice of numbers d_{12}, d_{13}, d_{23} satisfying the triangle inequality, there are 3 curves for which equality holds.

2 Proof of Theorem 1

The first part is immediate and follows from the fact that the complete graph K_m includes the star $K_{1,m-1}$. Select $i \in [m]$ which minimizes $\max_{1 \leq j < k \leq m} (d_{ij} + d_{ik})$, and let $\alpha_1, \ldots, \alpha_m$ be corresponding parametrization functions so that for any $j \in [m] \setminus \{i\}$, $d_{ij} = \max_{t \in [0,1]} \|f_i(\alpha_i(t)) - f_j(\alpha_j(t))\|$. By the triangle inequality, for any $j, k \in [m]$, $\max_{t \in [0,1]} \|f_j(\alpha_j(t)) - f_k(\alpha_k(t))\| \leq d_{ij} + d_{ik}$. This implies that

$$d_{\mathcal{F}} \leq \min_{1 \leq i \leq m} \max_{1 \leq j < k \leq m} (d_{ij} + d_{ik}). \qquad (2)$$

Intuitively, i is chosen as the "leading" curve, or the "man" curve, while the others are the "dog" curves. Then the distance between any two dogs or between the man and any dog throughout the walk is bounded by the sum of the lengths of the longest two leashes the man holds.

We will now construct an example in which the bound is tight. Even if our bounds hold in any dimensions, the curves in this example can be selected of the simplest form, i.e., polygonal curves on the real line. For illustration however, the different segments are drawn stacked on each other, see Figure 1.

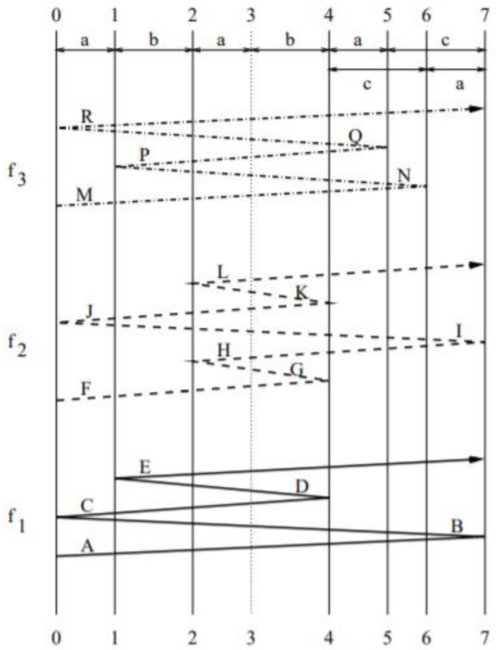


Figure 1: A set of three polygonal curves used in the proof

The following observation is useful in obtaining lower bounds:

Observation 1 *Consider a piece f' of a curve which consists of a horizontal segment of length x, traversed from right to left. Suppose that this piece is matched to a piece of another curve which moves monotonically to the right (or which is just a single point). Then the maximum distance in this joint parametrization is at least $x/2$.*

Lemma 2 *Let $\mathcal{F} = \{f_1, f_2, f_3\}$ be the set of three polygonal curves shown in Figure 1, where $a \leq b \leq c$ and $c \leq a + b$. Then $d_{12} = b$, $d_{13} = a$, $d_{23} = c$ and $d_{\mathcal{F}} = a + b$.*

Proof. Denote the five segments which make up f_1 by A–E. Similarly we denote by F–L and M–R the seven (resp. five) segments of f_2 and f_3.

The idea of the example is as follows: To achieve the minimum distance $d_{23} = c$, the small wiggle FGH on f_2 must be matched to the large zigzag MNP on f_3. On the other hand, the minimum distance $d_{12} = b$ can only be achieved if the small wiggle FGH on f_2 matches the straight movement A on f_1. For achieving the minimum distance $d_{13} = a$, A must be matched to M, however. It follows that not all three pairwise minimum distances can be achieved simultaneously in a joint reparametrization for all three curves.

A graphical representation of the situation is shown in Figure 2 as a three-dimensional box, representing the joint parameter space $(\alpha_1, \alpha_2, \alpha_3)$ of three points moving on the three curves. The three sides of the box are *free-space diagrams* [2] for pairs of curves with a threshold value $\varepsilon = a + b$. White regions (the free space) correspond to pairs of points $f_i(\alpha_i)$ and $f_j(\alpha_j)$ with distance at most ε, and shaded areas are forbidden areas where the distance is too big. A solution with Fréchet distance at most ε is represented by a path from the origin (in the center of the picture) to the opposite corner of the box which is monotone in each direction, and for which the projection to each coordinate hyperplane lies in the free space. The projections of two such paths is shown in the figure. (They have the same projection on the α_1–α_2 plane.) One sees that certain passages are about to become blocked if ε is decreased below $a + b$, for example, point 1 in the α_2–α_3 plane. The segment labeled 2 in the α_2–α_3 plane and in the α_1–α_3 plane will be unpassable because it is no longer monotone when ε is smaller than $a + b$. Some other segments of this kind are shown as dotted lines. It is apparent that for $\varepsilon < a + b$ the only remaining monotone paths in the α_1–α_3 plane go around the obstacles like the path through point 1 shown in these pictures. One can then check that this is inconsistent with a monotone path through space whose projection on the α_1–α_2 plane and on the α_2–α_3 plane lies in the free space.

In the following we give a detailed and elementary proof that does not make recourse to the free-space diagram. We denote different points on the curves by their labels $\in \{0, 1, 2, 3, 4, 5, 6, 7\}$ indexed by the segment to which they