

# Fréchet distance

בהנחיית ד"ר שלמה ינץ מאוניברסיטת בר אילן ובשותוף

ד"ר אליא פקר ראש מחלקת אינטלק

עמיית בן דוד | קים אחראפי | איתן שמעוני

תעודות זהות

326825247 | 207909979 | 305144685

## תוכן עניינים

2 .....	הקדמה על מטרת הפרויקט .....
3 .....	רקע כללי ו שימושים טכנולוגיים .....
3 .....	• רקע כללי: .....
5 .....	• שימושים טכנולוגיים .....
6 .....	רקע מתמטי .....
7 .....	אלגוריתם לחישוב מרחק <b>frechet</b> .....
7 .....	• חישוב המרחק בין שתי עקומות: .....
13 .....	• חישוב המרחק עבור יותר מ 2 עקומות .....
17 .....	• חישוב המרחק עבור $n$ עקומות .....
22 .....	חישוב המסלול המשותף של העקומות .....
24 .....	מסקנות .....
25 .....	ביבליוגרפיה .....
26 .....	נספחים .....

## הקדמה על מטרת הפרויקט

בתחילת הסמסטר קיבלנו מאגר מידע ומאמרם 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 מחשב 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 החוקרים עד היום עוסקו במרחק בין שתי עקומות בלבד.

תחילה רצינו לחשב את המרחק והמסלול לשתי עקומות על מנת לחזור את הנושא לעומק, כדי לחשוב בהמשך את המרחק עבור מספר עקומות ברצוננו.

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

במקביל, קראנו ולמדנו מתרן המאמרים איך לחשוב את המסלול המשותף בין שתי העקומות.

כלומר המסלול המשותף הוא המסלול המתראור את הדרכ המשותפת שעושים האיש והכלב יחד בהנטן אורך רצועה המתאים להם.

במהלך המחקר נחשפנו לפלטפורמה view Fréchet 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 ניתן תוצאה מימד 3.

ובאופן כללי חישוב מסלול עבור Ch עקומות ניתן תוצאה ממימד Ch, 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 ו שימושים טכנולוגיים

### רקע כללי:

גיאומטריה חיישובית מלאת תפקיד חשוב במספר רחב של תחומי מחקר שונים ומגוונים. תוצאות שונות ואלגוריתמים עילים הכרחיים לצורך חישובים כמו "מייקום נקודת חמה של WF-WI", "ראיית מחשב", " ניתוב מפות", "עיבוד דיבור וכותב יד" ומספר רב של שימושים נוספים.

בעיות רבות נפתרות על ידי התאמת ומציאת מסלול העובר דרך נקודות. הבעיה עוסקת ברוב המקרים או שונית להפוך אותן לכאלו – בעקומות פוליגונליות שבן נתמך בפרויקט זה. בעיה טבעית שעולה כאשר רוצים להשווות צורות הוא בהונטה שתי עקומות: "כמה הן דומות אחת לשניה".

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

### דוגמאות למרחקים:

**המרחק שగוף מסוים עשה** - אורך של מסלול ספציפי העובר בין שתי נקודות, כמו המרחק שאדם הלך תוך כדי ניוטם מבון.

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

**מרחק גאודי** - אורך המסלול הקצר ביותר בין שתי נקודות תוך שהוא נשאר על פני שטח כלשהו, כמו מרחק המעלג הגדול לאורך עקמת כדור הארץ.

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

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

**מרחקי מנהטן / גיאומטרית נהגי המוניות** - בניית מרחב מטרי, שבו מודדים את המרחקים על-פי אילוצי הנסעה של נהג מונית, במקום ב"מעוף הציפור".

**מרחק צ'יביש – מרחק השחמט**: הוא שווה למספר הצעדים המינימלי שהמלך עושה בלוח שחמט כדי להגיע לשבצת כלשהו (המלך יכול לנوع אופקיות, אנכיות ובאלכסון). למשל המרחק בין 5 ל 1 או בין 4.

**מרחק האסודרפ** – המרחק המרבי של קבוצה מנוקדה 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, בין העקומה חשוב, למשל עקומות שנקלטו על ידי דיגיטלי במו כתוב יד, כאשר הפרמטריזציה של העקומה נתונה כבר על ידי מבשיר הקלט.

כדי להסביר לבעה זאת, נציג כאן מרחק בשם "**מרחק פרשה**" שפותח על ידי המתמטיקאי מורייס רנה פרשה ועל מרחק זה מבוסס הפרויקט שלנו.

בצורה אינטואיטיבית נוכל לחשב על מרחק פרשה בצורה הבאה:

נדמיין שאנו מוצאים את הכלב שלנו לטoil, בינו לבין הכלב לחברת רצואה, נביט במסלול ההליכה שלנו בשתי עקומות שונות, כאשר על אחת מהם הולך כלב ועל השניה הולך אדם בקצב שונה ובניהם הרצואה. מרחק פרשה מוגדר להיות **מרחק הרצואה המינימלי הנדרש לנו**.

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

בשנת 1995 המציאו Alt ו Godue אלגוריתם שנועד למצוא פתרון לשאלה זו בזמן פולינומיAli. בפרויקט שלנו נתבפס על אלגוריתם זה עבור מרחקי פרשה בין 2 עקומות ונפתח את הרעיון למרחקי פרשה עבור ח' 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" מתחילה בחלוקת של דגימת תווים לאזורים מלכינים שונים. ואז מחושבים ערכי המרחקים מכל אזור לבן אזור אחר. בטכניקת חילוץ תכונות מבוססות מרחק גם דגימת תווים מחולקת במספר מקטעים ומרחקים נבדדים מקטע מסוים לכל הקטעים האחרים.

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

**הזרמת מידע:** מכונה גם "סטרימינג" טכנולוגית אינטראקט להעברת מידע באופן רצוף ומתרשף ע"י ספק תוכן. שידור מידע ב"סטרימינג" מאפשר האזנה או צפייה בקובץ טרם התקבל כל המידע המרכיב אותו, המדיה מוזכרת בחלקים קטנים. טכנולוגיה זו מביאה לחסוך בכמות התעבורה שעוברת בראשת אל מול איכות היידיאו המוצג ומאפשרת שידורים חיים באינטרנט או צפייה מקוונת. בעדרת מרחקי frechet ניתן לחשב מסלול טוב לשידור כל פריטים – שיפור איכות תמונה ואיכות אודיו: לאחרונה הציעו שני חוקרים בגוגל מัดדים חדשים לאודיו ווידאו הבנויים על עקרונות מרחקי Fréchet

**חיזוי מבנה חלבון:** חיזוי מבנה החלבן הוא הסקת המבנה התלת מימדי של חלבון מרצף חומצות האmino שלו - כלומר חיזוי קוילו והמבנה המשני והשלישי שלו מהמבנה העיקרי שלו. בעדרת מרחקי frechet ניתן לחשב את המסלולים הטובים ביותר עבור מבנה החלבן ובכך לדעת את מבנו.

## רקע מתמטי

עקומה מוגדרת להיות פונקציה (מפה) רציפה  $V \rightarrow R$ , במשמעות  $f: [a, b] \rightarrow V$ .

$a \leq b$  ו  $(V, d)$  הינו מרחב מטרי.

מרחב פרשה

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

$$\delta_f(f, g) = \inf_{\alpha, \beta} \max_{t \in [0, 1]} d(f(\alpha(t)), g(\beta(t)))$$

באשר  $\alpha$  (ובהתאם גם  $\beta$ ) הינה פונקציה שרירותית, עולה ורציפה מ  $[0, 1]$  אל  $[a, b]$  ו  $[a', b']$  בהתאם.

לчисוב מרחקי פרשה בין עקומות שרירותיות, משתמש בעקומות פוליגונליות.

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

לכל  $i = 1, \dots, n$  ה策ומ של ק על הקטע  $[i+1, i]$

הינו העתקה אפינית כך ש

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

באשר ניצג את העקומות הפוליגונליות  $A, B, C, Q$  : אוסף נקודות על העקומות המיצגות אותן בהתאם  $[U_1, \dots, U_n], [V_1, \dots, V_n]$

תרשים "השטח הפנוי" (free space era)

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

באשר דיאגרמת השטח הפנוי בין שתי עקומות עברו מרחק נתון  $\epsilon$  הינו

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

באשר דיאגרמה זו היא תחום דו מימדי המורכב מכל הנקודות בשתי העקומות אשר המרחק ביניהם הוא לכל היותר  $\epsilon$

## אלגוריתם לחישוב מרחק :frechet

### חישוב המרחק בין שתי עקומות:

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

חישוב המרחק עבור שתי העקומות לא דורש מבני נתונים מסובך ומתבצע ע"י תכונות דינמי.

שיטת התכנון הדינמי לבניית אלגוריתמים משמשת לפתרון בעיות שאין ניתנות לפתרון עיל בשיטת הפרד ומשל נאיבית.

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

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

זכור כי לפי ההגדרה:

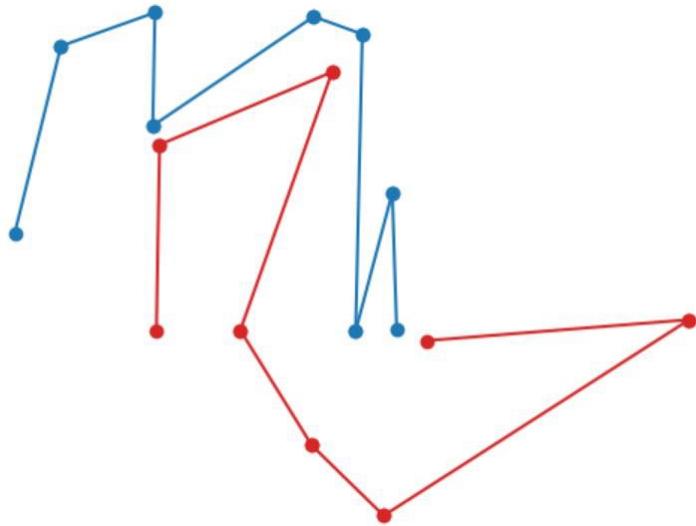
עבור עקומות  $B, A$  עם פרמטריזציה  $\alpha, \beta$  מרחק Fréchet מוגדר להיות:

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

את העקומה הפוליגונלית  $A$  נציג ע"י  $P$  : אוסף נקודות על העקומות המייצגות אותן  $U_1, \dots, U_n$

את העקומה הפוליגונלית  $B$  נציג ע"י  $Q$  : אוסף נקודות על העקומות המייצגות אותן  $V_1, \dots, V_n$

באופן הבא:



את העקומות בציור ניצג ע"י הנקודות המסומנות, ככלمر:

$P =$

$[[12.8289,228.248],[24.0543,253.104],[46.8761,257.56],[46.3385,242.508],[85.4033,257.023],[97.2302,254.693],[95.6174,215.27],[104.502,233.594],[105.571,215.419]]$

$Q =$

$[[47.2345,215.27],[48,240],[89.8827,249.659],[67.6065,215.331],[84.9682,200.029],[102.413,190.692],[176.614,216.736],[112.902,213.953]]$

נחזיק מטריצה זו מיידית עבור המרחקים בין אוסף הנקודות  $P$  לבין אוסף הנקודות  $Q$ .

נאותחל את המטריצה לערך -1 ונשלח אותה לפונקציה רקורסיבית 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 נשמר 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 בסוף חיבור

## 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 `inputfunction.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)

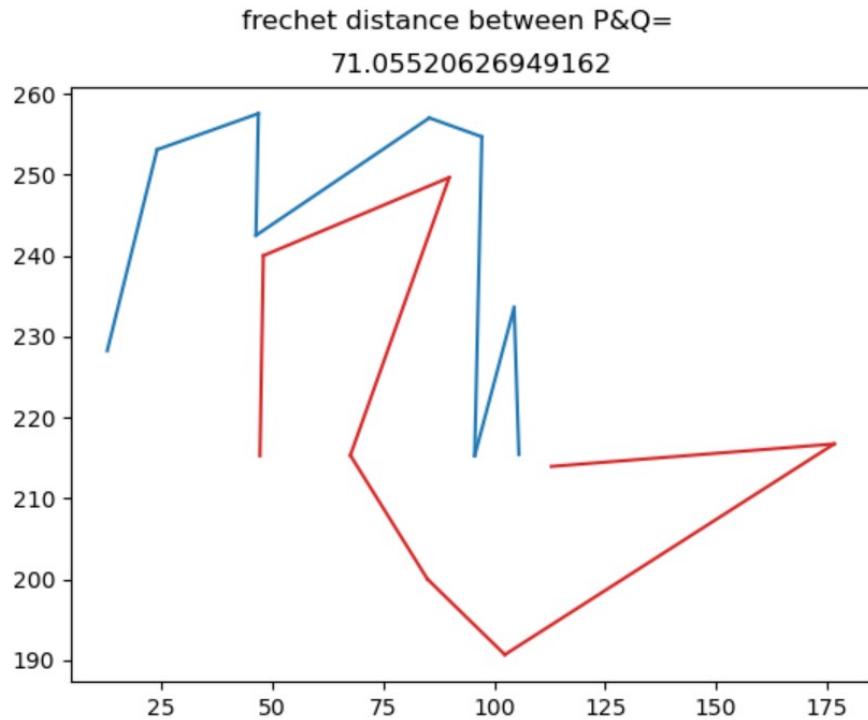
if __name__ == '__main__':
    # example 1:
    P =
    [[12.8289,228.248],[24.0543,253.104],[46.8761,257.56],[46.3385,242.50
    8],[85.4033,257.023],[97.2302,254.693],[95.6174,215.27],[104.502,233.
    594],[105.571,215.419]]
    Q =
    [[47.2345,215.27],[48,240],[89.8827,249.659],[67.6065,215.331],[84.96
    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
    freq_P_Q = frechetcalcu.frechetDist(P,Q) # freq[0] is the leash
length freq[1] is the distance matrix
    plt.title(freq_P_Q[0])
    plt.suptitle('frechet distance between P&Q=')

    plt.show()
```

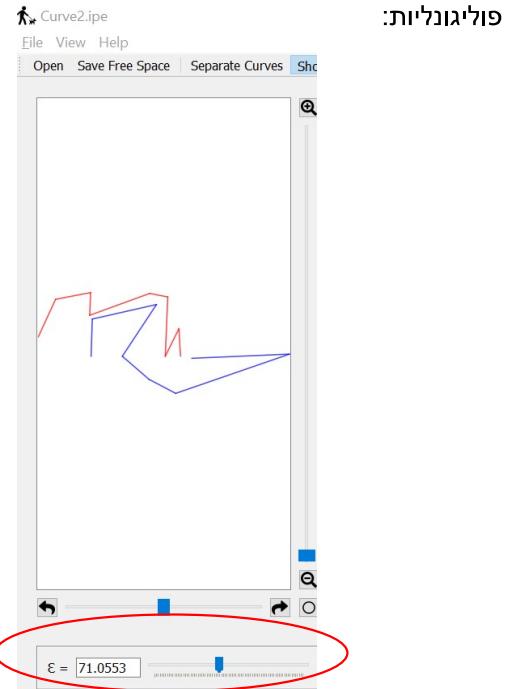
## פתרונות:

### דוגמה 1

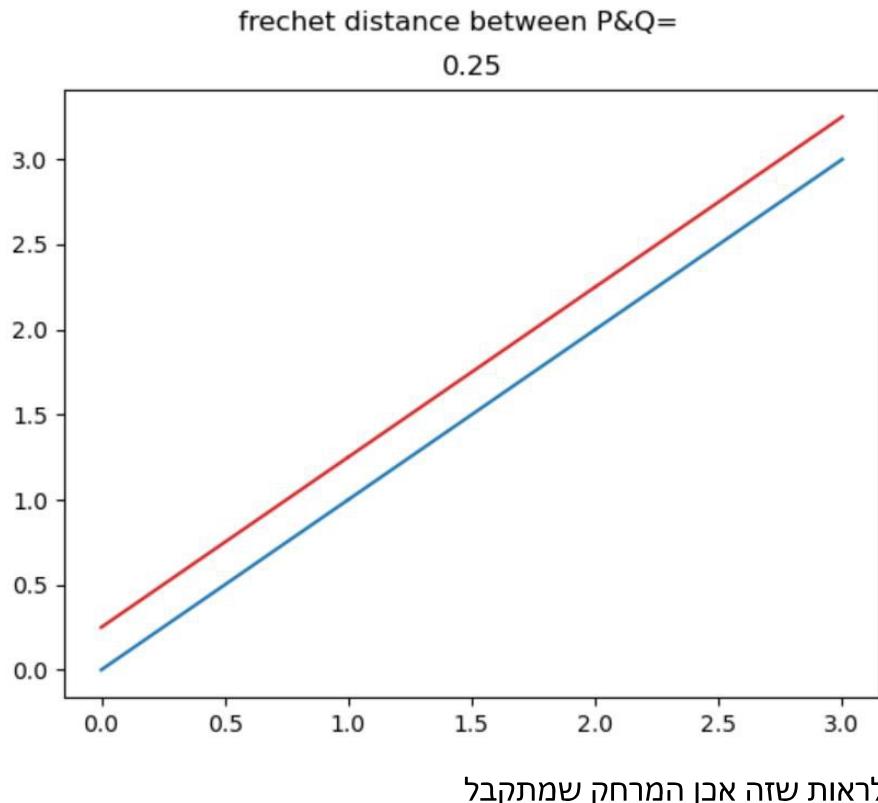


את העקומות בדוגמה זו ללחנו מתוך חישוב במחקר "Fréchet view", ובדקנו את התוצאה שלנו מול התוצאה שלהם.

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



## דוגמה 2:



$$P = [[0,0],[1,1],[2,2],[3,3]]$$

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

טבלת המרחקים(האוקלידיים) ביןיהם

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 שכן מקבלים שזה אורך הרצעה.

**למה:**

עבור  $Q, P$  עקומות פוליגונליות חישוב המרחק ניתן לחישוב ב( $\text{פkm}$ )ו כאשר ק הוא מספר הנקודות בפרמטריזציה של  $Q$  ו  $Q$  הוא מספר הנקודות בפרמטריזציה של  $P$

❖ הוכחת הלמה נמצאת בסוף ח

## 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.

תחליה על מנת למצוא את החוקיות כדי לבנות מודל המותאם לכל במות של עקומות מכל סוג בניינו מודל עבור 3 עקומות.

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

### קוד בפייטון: 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-
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]),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 `ym`.

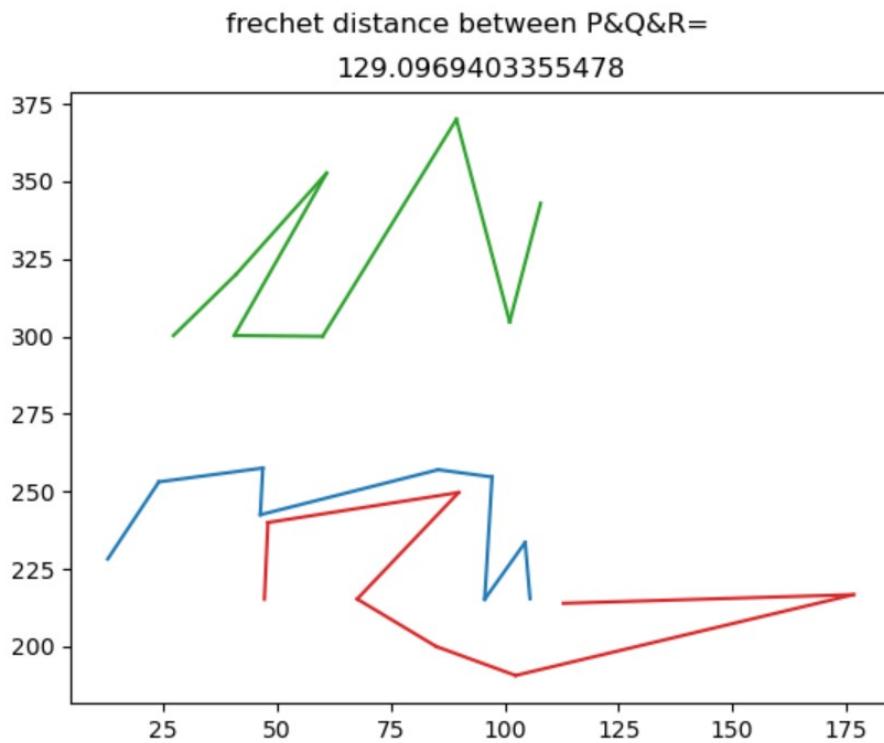
```

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 diantance 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 diantance 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()

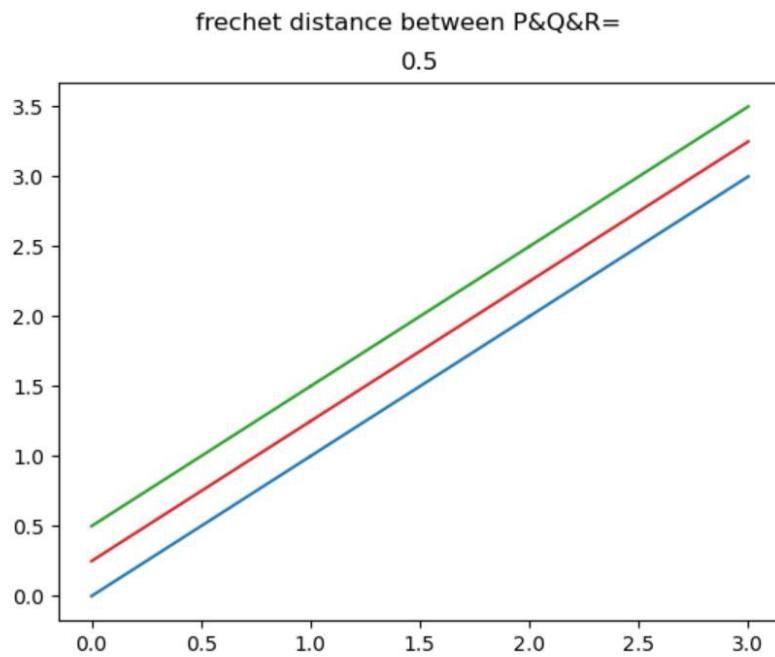
```

## פתרונות:

דוגמא 1



דוגמא 2



הוכחה שנמצאת בבליאוגרפיה נבע שבעור דוגמא 2 מרחק Fréchet בין שלושת העקומות שווה לסכום מרחקי הפרשה של העקומה  $P$  מהעקומות  $R$  ו- $Q$  וכן המרחק שמתקיים שווה לסכום המרחקים.

המוכיח שהתקבל מהו רצואה מספקת עבור כל אחד מהמרחקים בין כל שתי עקומות ובנוסף מהו את מרחק frechet עבור שלושת העקומות יחד.

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

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

כי המרחק בין  $P$  ל- $Q$  הוא 71.055 המרחק בין  $Q$  ל- $R$  הוא 129.09 והмарחק בין  $P$  ל- $R$  הוא 127.55

בנוסף ניתן לראות כי  $129.09 > 71.055 + 127.55$  כלומר סכום המרחקים לא מהו רצואה המינימלית שאפשר לקחת עבור שלושת העקומות.

סה"כ המרחק המינימלי המתאים לשילושת העקומות שווה למרחק הפרשה בין  $Q$  ל- $R$  על כן התוצאה שקיבלנו עבור שלושתם הינה 129.09

## חישוב המרחק עבור ח' עקומות

על מנת להכפיל את האלגוריתם לח' עקומות יש למצאו את אוסף כל הקומבינציות של נקודות מכל אחת מהעקומות.

המודל שלנו יוכל סט של במות הנקודות בכל עקומה, וסת של עקומות.

כלומר علينا למצאו את אוסף כל הקומבינציות של אינדקסים כאשר כל קומבינציה מכילה במות אינדקסים בכמות העקומות הקיימות.

בעזרת קומבינציות אלה נוכל להכפיל את האלגוריתם.

למציאת הקומבינציות השתמשנו בספריה של פיתון `itertools` בפונקציית `product`, פונקציה זו מוחזירה את המוצר הקרטזי של שתי קבוצות.

במונחים של מתמטיקה מוצר קרטזי של שתי קבוצות, מוגדר בסט של כל הזוגות המסדרדים (b, a) 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 j, נרצה לחשב את הקומבינציות  $(1-j, i)$ ,  $(j-i)$ ,  $(i-1, j-1)$ .

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

על מנת לקבל את הקומבינציות הרצויות גם עבור מערך אינדקסים, נשתמש בפונקציית `product` עבור כל זוג אינדקסים, על כל אוסף קומבינציות של האינדקסים נפעיל שוב את הפונקציה לקבלת כל האופציות האפשריות (בהתחשבות במצב של במות 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 המקסימים בין - הרקורסיה המינימלית שקיבלו עד כה, לבין המרחק האוקלידי הנוכחי בין הנקודות.

בר התוצאה שנתקבל תהווה את אורך הרצועה המינימלי המתאים לכל העקומות, לפי ההגדרה.

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

בכך כאשר אדם יצא לטויל עם 1-א לבבים הוא יוכל לדעת מראש את אורך רצועה "שתספיק לו" על מנת ללכנת במסלול משותף עם כל הלבבים  
בנוסף רצועה זו תתאים למסלול בין האדם לבין כל לב ובין כל לב ללכבל (גם אם אינה מינימלית)

## קוד בפייתון: מתרן 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 `mf.function`.

```

#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]]

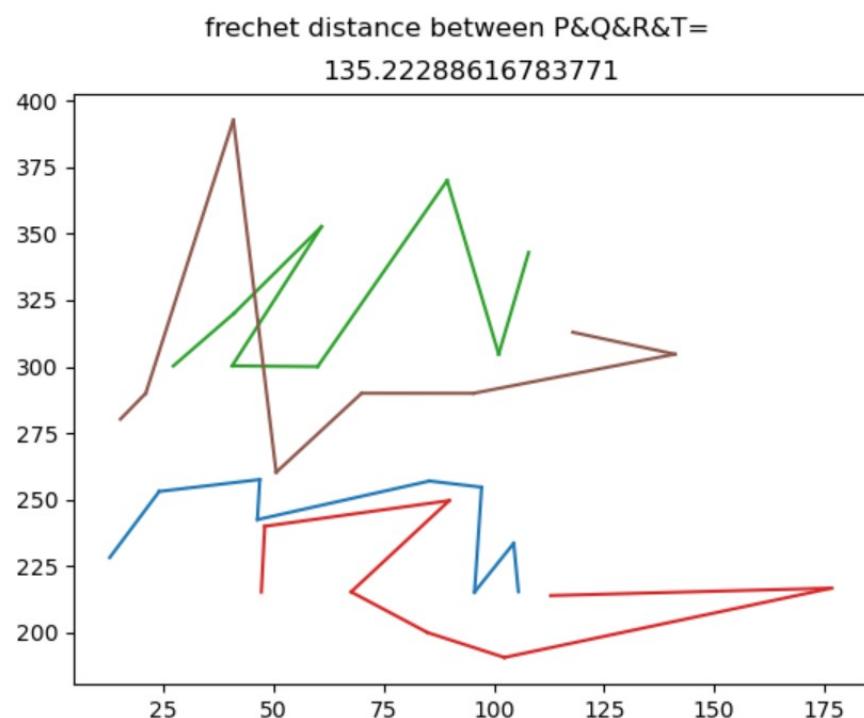
functionDraw(P,'tab:blue')
functionDraw(Q,'tab:red')
# frechet diatance for 2 curves
frec_P_Q = frechetcalcu.multiple_frechetDist_new([P,Q]) # frec[0] is the leash lenght
frec[1] is the distance matrix
plt.title(frec_P_Q[0])
plt.suptitle('frechet distance between P&Q=')
plt.figure()
functionDraw(P,'tab:blue')
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 2-1 קיבלנו את אותן התוצאות בידיו

**דוגמא עברו 4 עקומות: (P,Q,R,T)**



## чисוב המסלול המשותף של העקומות

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

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

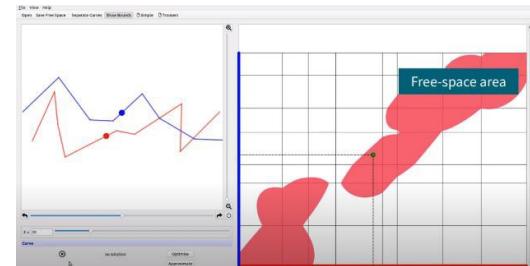
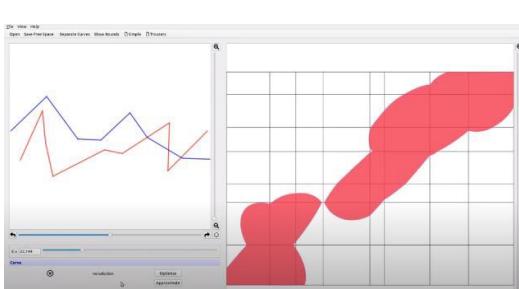
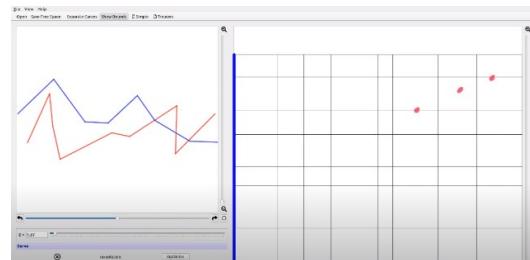
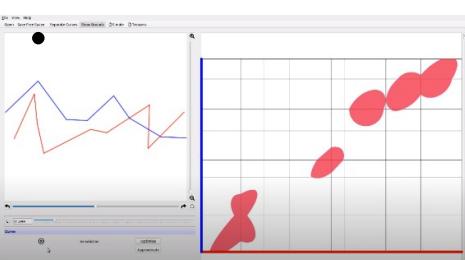
לקבלת תצוגת Fréchet View מתחילה בלקיחת שתי עקומות P - Q ופירוק למרחוקיהן בין כל שתי נקודות בכל עקומה:

מרחוקים אלו יהיה ציר ה- X וציר ה- Y ( שנבחר באופן שרירותי ) עברו גרפ חדש, אפשר לדמיין שלקיחת פונקציה מחלוקת אותה לקו ישר ומשתמשים בו באחד הצירים בגרף החדש.

בשלב הבא נעסק במרחב Fréchet בין העקומות, החל מ- 0 במרחב הרצועה עד מרחק Fréchet. הגדלנו את מרחק Fréchet באופן הדרgtiy בין שתי העקומות וזה גורם לתוך המוביל את המרחק המתאים שלהן (ברף החדש, או בערך ה- A בפונקציה הישנה) לגדול ברף החדש שלנו. לדוגמה, אם נקרא למרחק "D" אנו רואים ברף הראשוני, כאשר D קטן מאוד המקומות שבהם הגרפ מתחילה להתמלא מתחילה בנקודות בהן הגראפים נחתכים. בגרפים הבאים אנו רואים שהגרף מתחילה 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 הצלחו להגיע "מתרחבים" בתרשים וממשיכים "להתרחב".

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



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

טהיליך זה תואר לעיטים קרובות בהוספת "טיפות מים" למקומות בהם המרחק שווה או גדול יותר, ובתווך גורם למים להתרפש סביב הגרפ.

באשר Fréchet View מסתיים, אנו יכולים להתמקד במסלול האפשרי.

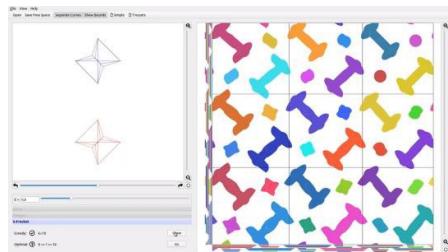
מסלול אפשרי במקרה זה הוא הרגע בו הושלם View Fréchet, (נקודות ההתחלת וסיום נמצאות באותו מקום). הגרפ הופך לגרפ עם כל הנקודות בהן שתי הפונקציות נמצאות במרחב של  $P$  (המרחב המקומי בין העקומות) או פחות, כלומר הגרפ הפך לגרפ של כל הנקודות בהן "האדם והכלב" במרחב Fréchet או פחות.

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

- נניח שנרצה שהאיש (שהולך עם כלבו) יגיע קודם לסוף, והוא היה לו Kohr את הנקודות הקרובות ביותר לציר  $U$  והוא יגיע קודם לסוף, אם זה אפשרי (אותו דבר ההפוך).
- נניח שיש מקום שאתה מעוניין שהאיש ישאה למשל "חנות" בדרך. אתה פשוט מוסיף את מקום החנות בפונקציה עצמה ( $P$  או  $Q$  במקרה זה). אז הם היו נשאיםשוב על הנקודות הקרובות ביותר לציר  $U$  עד שיגיעו ל"חנות" ומשם ממשיכים רק עם הכלב בשביל (הולכים במקביל לציר  $-X$ ). זה אפשר לאיש לשחות הabi הרבה זמן בחנות כשהוא מורד את הכלב מדרך הרצואה (הרצואה תהיה מספיק ארוכה גם בשיטה זו).
- אנו יכולים לבחור נתיב עם המרחק המינימי המשמש בפונקציות שלנו או המרחק המרבי, זה שימושי יותר לביצוח נתונים ובדיקות (לודא שהדברים מנוצלים בצורה אופטימלית)

Fréchet view מעניק לנו ייצוג ויזואלי מעוניין למרחקים בין העקומות שלנו, המתאר את הדמיון בין העקומות, אך זה לא מאד שימושי בשמהדר ב- 4 עיקומות או יותר.

בנושא המחקר העיקרי שלנו, בגלל מספר הצירים, התוצאה תהיה צורות במספר רב של מידדים, זה ללא ספק יוצר מחרזה ויזואלי מתקשים עם כמה עיקומות שאנו ממליצים לכל מי שיחקอร אותו בעתיד, שינסה לראות.



## מסקנות

כiom כFI שהזכרנו משתמשים באלגוריתמים לחישוב מרחק Fréchet של עוקומות לשימוש בחישוב במקביל, בلومר UIBOD בו זמינות של בעיה מסוימת ע"י מספר מעבדים או מספר ליבוט על מנת להגיע לתוצאה מהירה יותר.

דוגמא זו ודוגמאות נוספות שהזכרנו מהוות מוטיבציה למציאת המרחק עבור חעוקומות.

הפתרון שהציגנו עבור בעיה זו הינו פתרון למקרה הבודד, אך פתרון הבעיה עבור המקרה הרציף יהיה שימושי בתחוםים נוספים שהזכירנו, למשל עבור הזרמת מידע (streaming).

ראינו כי לפתרון שמצאנו עלות חישוב גובה. עלות החישוב הממוצעת של מציאת המרחק עבור חעוקומות, כאשר נסמן את גודל העוקמה ה  $\alpha$  ב  $\omega$ , הולות עבור  $\alpha$  עוקומות הינה:

$$O(n_1 * \dots * n_m * \log(n_1 * \dots * n_m))$$

אלגוריתם החישוב עבור שתי עוקומות מתבצע באופן רקורסיבי, لكن עלות החישוב עבור חעוקומות, 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 העוקומות עד כה וכן בוחרים את מרחק Fréchet המתאים לכל הנקודות עד שלב זה. لكن כדי לחשב את המרחק יש לפתור את הבעיה על ידי חלוקתה לתחום בעיות הנפתרות בתורן על ידי בעיות קטנות יותר.

לא ניתן לחשב את המרחק על ידי חישוב נאיבי בשיטת הפרד ומשול וכן יש צורך בתכנון דינאמי הדורש רקורסיה. בתוצאה מכך חישוב המרחק במספר רב של עוקומות הוא בעל עלות חישוב יקרה.

עם זאת, החיסכון שיכולים להעניק מרחק Fréchet בשימושים רבים, יכול "לפצות" על הולות הגבהה שלו, למשל בחזרה לעיבוד במקביל, ניתן לkürוץ תהליכיים שקרים בכמה מעבדים במקביל כך שיישתלים לשימוש במרחקי Fréchet למרחות עלות החישוב הגבוהה.

מרחקי Fréchet מתארים את הדמיון בין העוקומות ועל כן תורמים במקרים בהם יש להשוות בין פעולות שונות שקוות במקביל וمتוארכות ע"י עוקומות. עלות החישוב עבור מספר רב של עוקומות גבוהה ועל כן יש להשתמש במרחקי Fréchet לצורך התאמת 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](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...](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  $\varepsilon$ .
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  $\varepsilon$  of types a) and b) and apply onto them the technique of Algorithm 2. This gives two values  $\varepsilon_1, \varepsilon_2$  with  $\delta \in [\varepsilon_1, \varepsilon_2]$  and  $\varepsilon \notin [\varepsilon_1, \varepsilon_2]$  for any critical value  $\varepsilon$  of type a) or b) other than  $\varepsilon_1$  or  $\varepsilon_2$ .
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  $\varepsilon$  of Step 1 with  $\varepsilon \leq \varepsilon_1$ ). Use Cole's variant of parametric search based on sorting the values in  $A$  to find the actual value of  $\delta$ .

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

```
Function dF( $P, Q$ ): real;
  input: polygonal curves  $P = (u_1, \dots, u_p)$  and  $Q = (v_1, \dots, v_q)$ .
  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  $ca(i, j)$ 
      elseif  $i = 1$  and  $j = 1$  then  $ca(i, j) := d(u_1, v_1)$ 
      elseif  $i > 1$  and  $j = 1$  then  $ca(i, j) := \max\{c(i-1, 1), d(u_i, v_1)\}$ 
      elseif  $i = 1$  and  $j > 1$  then  $ca(i, j) := \max\{c(1, j-1), d(u_1, v_j)\}$ 
      elseif  $i > 1$  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 שהתבסנו עליו עברו חישוב מרחק בין שלושה עקומות או יותר באמצעות סכום מרחקי הפרשה בינהם, בנוסף הוכח הטענה 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  $\alpha_1, \dots, \alpha_m$  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

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

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.

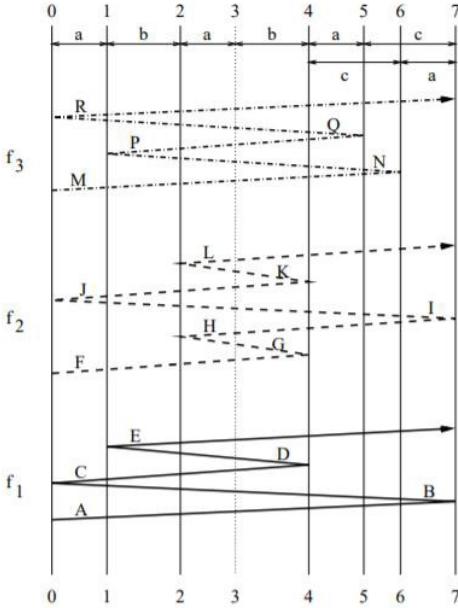


Figure 1: A set of three polygonal curves used in the proof

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  $\varepsilon$ , and shaded areas are forbidden areas where the distance is too big. A solution with Fréchet distance at most  $\varepsilon$  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  $\alpha_1-\alpha_2$  plane.) One sees that certain passages are about to become blocked if  $\varepsilon$  is decreased below  $a+b$ , for example, point 1 in the  $\alpha_2-\alpha_3$  plane. The segment labeled 2 in the  $\alpha_2-\alpha_3$  plane and in the  $\alpha_1-\alpha_3$  plane will be unpassable because it is no longer monotone when  $\varepsilon$  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  $\alpha_1-\alpha_3$  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  $\alpha_1-\alpha_2$  plane and on the  $\alpha_2-\alpha_3$  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