# מיני פרויקט בניתוח נתונים

ינואר 2019

נכתב על-ידי גל אברמוביץ'

מוגש לידי ד"ר סיון סבתו

## חלק ראשון - האלגוריתם המשופר

ניסיתי רבות למצוא קורלציה בין נתוני הסרטים ו/או נתוני המשתמשים לבין "התאמה" בין סרטים, שתיתרגם לעלות נמוכה יותר, אולם ללא הצלחה משמעותית.
בסופו של דבר החלטתי לעבוד בגישת top-down ולשפר את אלגוריתם החלוקה לקבוצות על סמך פונקציית העלות בלבד.
משום שהפרמטרים בפונקציה זו הם ההסתברויות שצופה אקראי צפה בסרט מסוים, $p(m)$, או בצמד של שני סרטים מסוימים, $p(m_1, m_2)$, בניתי אלגוריתם חמדן שלוקח רק את אלה בחשבון:

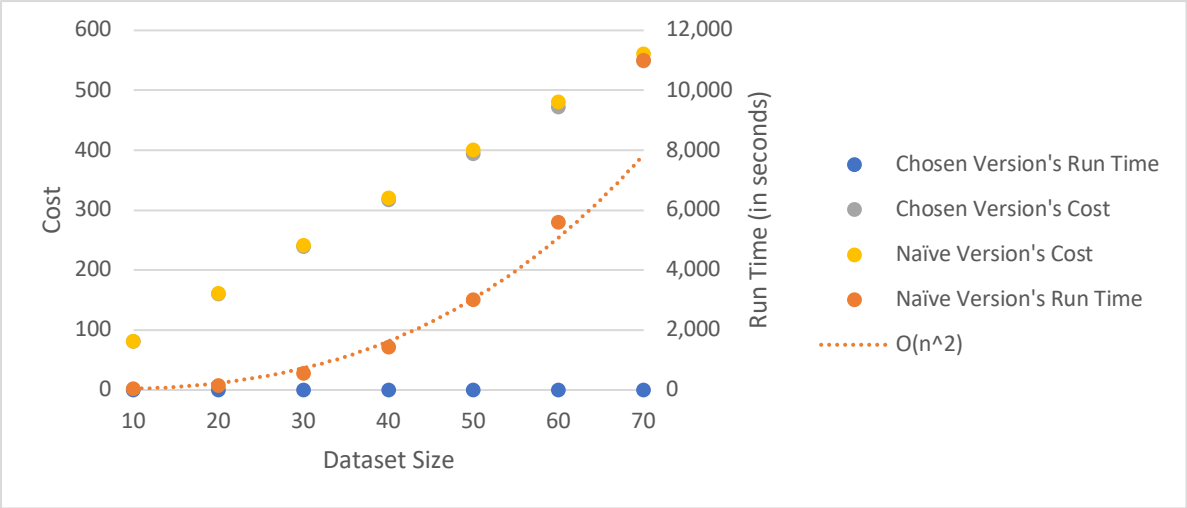
בהינתן רשימת קודקודים $V$, פונקציה $p : V \times V \to \mathbb{R}_{>0}$ (באופן ספציפי, הן לצרכים פורמליים והן בקוד הגדרתי כי $p(m,m) = p(m)$ ) וכן פונקציית עלות לכל קבוצת קודקודים $C \subseteq V,\ c : \mathrm{C} \to \mathbb{R}_{>0}$, כפי שהוגדר בהנחיות הפרויקט:

א. הכנס כל קודקוד $v \in V$ לקבוצה נפרדת $\{v_i\}$, כך ש- $\bigcup_{i=1}^{|V|}\{v_i\} = V$
ב. כל עוד $|V| > 0$ בצע:
   i. בחר את קבוצת הקודקודים בעלת העלות הגבוהה ביותר[1], נסמנה בתור הקבוצה ה-$i$ .
   ii. בדוק לכל קבוצה $j \neq i$ האם $c_i + c_j > c_{i \cup j}$, כלומר האם סכום העלויות של שתי הקבוצות גדול מעלות קבוצת האיחוד של שתיהן. אם כן, אחד את הקבוצה $i$ עם הקבוצה שעלות קבוצת האיחוד שלהן מינימלית[2].
   iii. אם לא נמצאה אף קבוצה מתאימה בצעד הקודם, הוצא את הקבוצה $i$ מתוך $V$ והוסף אותה ל-$V'$ .
ג. החזר את $V'$, חלוקה לקבוצות זרות (clusters) של קבוצת הקודקודים $V$.

למעשה זהו אלגוריתם clustering היררכי: בכל שלב הוא מאחד את שתי הקבוצות ה"דומות" ביותר (כלומר, עם העלות הנמוכה ביותר) וכך מקטין את העלות הכוללת, המוגדרת כסכום עלויות הקבוצות. קל לראות שהאלגוריתם אכן מתכנס, שכן בכל איטרציה מספר הקבוצות ב-$V$ קטן והאלגוריתם עוצר כאשר $|V| = 0$.

<u>אופטימיזציות לאלגוריתם</u>
לאלגוריתם שני צעדי בחירה (המסומנים בספרות עיליות: [1,2]), אשר נקבעו לאחר בדיקות שערכתי על סטים אקראיים של סרטים. כל בדיקה כללה למעשה 1000 איטרציות בהן הוגרלו קבוצות סרטים אקראיות מתוך כל הסרטים החוקיים (בעלי לפחות 10 דירוגים). בהצגת הנתונים התייחסתי לממוצע תוצאות הבדיקה.

**בדיקה ראשונה - בחירת הקבוצה הראשונה**:
מיון הקבוצות לפי עלות בכל פעם שהאלגוריתם מבצע את שלב (ב1) הביא לתוצאות טובות יותר (ראו תרשים 2).
באופן מפתיע, גרסה "נאיבית" יותר של האלגוריתם - בה בשלב ב' נבחר בכל פעם זוג הקבוצות שאיחודן מינימלי מבין כל הזוגות הרלוונטיים וכאשר לא קיים זוג כזה, האלגוריתם עוצר - הציגה ביצועים פחות טובים: עם clustering בעלות מעט גבוהה יותר וכמובן בזמן ריצה גרוע יותר, $O(n^2)$. (ראו תרשים 1).

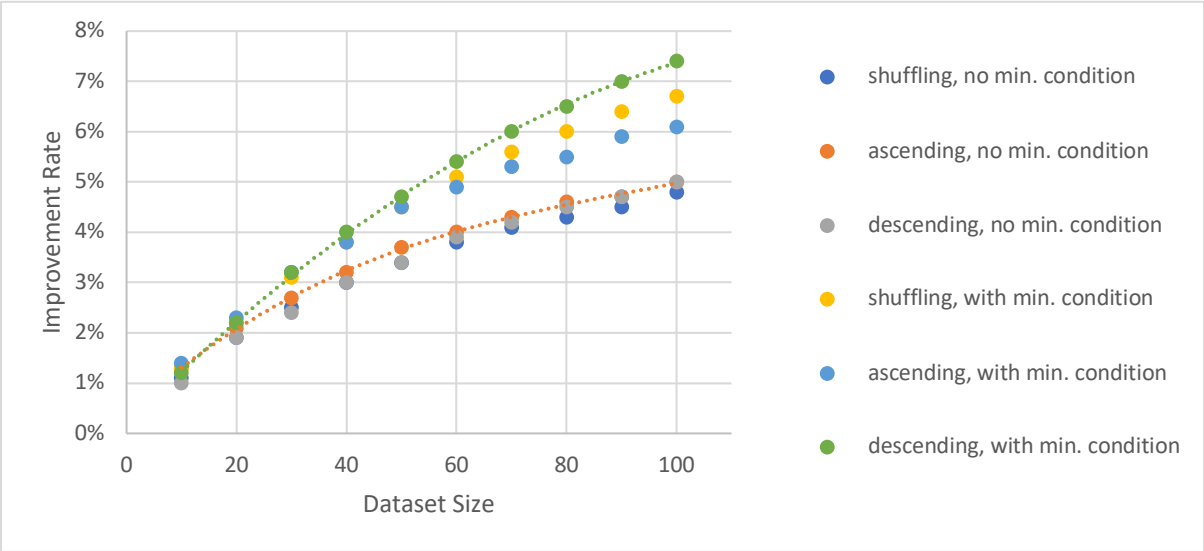



תרשים 1: השוואה בין זמני הריצה ועלויות האלגוריתם בגרסה ה"נאיבית" ובגרסה שנבחרה

**בדיקה שניה - בחירת הקבוצה השנייה לאיחוד:**
בחירת הקבוצות שאיחודן מינימלי השיגה את העלות הנמוכה ביותר.
להלן אחוזי השיפור בעלות ה-clustering שהשיג האלגוריתם המשופר, ביחס לזו שהשיג האלגוריתם המקורי, עבור קלט המונה 100 סרטים:

| | התחלת איטרציה עם קבוצה אקראית | התחלת איטרציה עם קבוצה בעלת עלות מינימלית | התחלת איטרציה עם קבוצה בעלת עלות מקסימלית |
|---|---|---|---|
| איחוד עם קבוצה אקראית | 4.8% | 5.0% | 5.0% |
| קבוצת האיחוד בעלת עלות מינימלית | 6.7% | 6.1% | 7.4% |

בגרף הבא ניתן לראות כי אחוז השיפור מתוצאות האלגוריתם המקורי עולה ממש ככל שמגדילים את גודל הקלט (מספר הסרטים) של האלגוריתם, מכך שהאופטימיזציות יעילות לכל גודל קלט:




תרשים 2: אחוזי השיפור של האלגוריתם המשופר ביחס לאלגוריתם המקורי, לפי פרמטרים שונים

## חלק שני - מקרה המדגים שיפור משמעותי

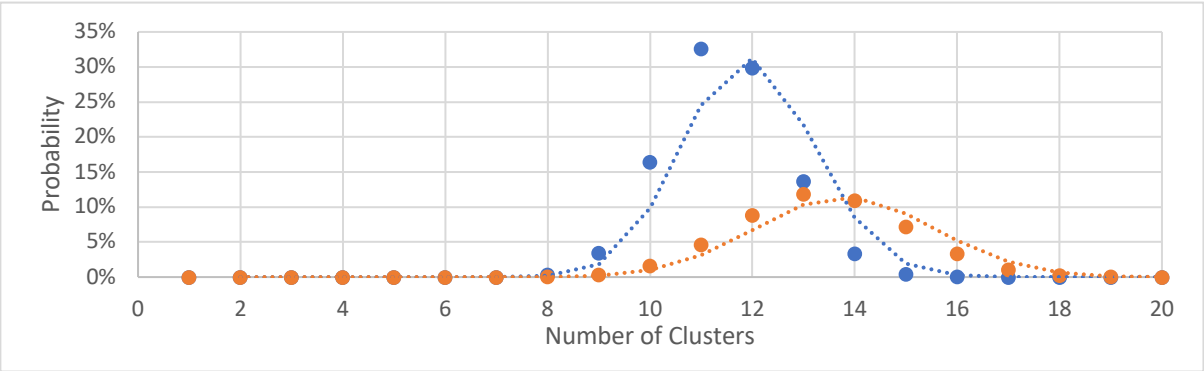
משום שאלגוריתם Clustering Correlation המקורי אינו דטרמיניסטי, בחרתי סט סרטים שעבורו גם בניתוח המקרה הממוצע (Average Case Scenario) ניכר שיפור בין הרצת האלגוריתם המקורי לבין האלגוריתם המשופר.
כדי למצוא סט סרטים כנדרש, הרצתי את שני האלגוריתמים על סטים אקראיים של סרטים והשוויתי בין עלויות הפלטים. לשם התייחסות, כפי שמתואר בגרף בסעיף הקודם, השיפור הממוצע עבור סט המונה 10 סרטים הינו 1.3%.
לסט הסרטים הבא שיפור **ממוצע** של 1.8% ושיפור **מרבי** של 5.8%:

| | | Original algorithm clustering | Improved algorithm clustering |
|---|---|---|---|
| 87 | Dunston Checks In | 8 | 4 |
| 175 | Kids | 4 | 0 |
| 375 | Safe Passage | 9 | 3 |
| 742 | Thinner | 6 | 1 |
| 743 | Spy Hard | 5 | 4 |
| 964 | Angel and the Badman | 3 | 2 |
| 1007 | Apple Dumpling Gang | 2 | 2 |
| 2175 | Déjà Vu | 7 | 3 |
| 2841 | Stir of Echoes | 1 | 1 |
| 3040 | Meatballs | 0 | 4 |

כאשר השיפור המשמעותי ביותר היה בין הרצת האלגוריתם המקורי, שבנה 10 קבוצות - המכילות סרט בודד כל אחת - בעלות של 88.69; לעומת האלגוריתם המשופר, שבנה את הקבוצות הבאות, בעלות של 83.48:

Cluster 0: 175 Kids
Cluster 1: 2841 Stir of Echoes, 742 Thinner
Cluster 2: 1007 Apple Dumpling Gang, The, 964 Angel and the Badman
Cluster 3: 2175 Déjà Vu, 375 Safe Passage
Cluster 4: 87 Dunston Checks In, 3040 Meatballs, 743 Spy Hard

זו אומנם דוגמה קיצונית, אולם מעניין להבחין כי גם באופן כללי האלגוריתם המשופר נטה לחלק את הקלט לפחות Clusters מאשר האלגוריתם המקורי:




תרשים 3: ההסתברות לקבלת כמות קבוצות בתוצאת האלגוריתם המקורי (בכתום) והאלגוריתם המשופר (בכחול) עבור קלט בגודל 20 סרטים, לפי 100 מיליון קלטים אקראיים

## חלק שלישי - אתגרים בפרויקט

בראש ובראשונה הפרויקט כלל לא מעט אתגרים מקצועיים, שכן התחלתי אותו ללא ידע רלוונטי לניתוח נתונים. לשמחתי, קיים ברשת מידע רב על ניסויים דומים שנעשו וסיפקו לי קצוות חוט רבים להתמודדות עם הבעיה.

מנגד, ריבוי הרעיונות הדומים, בשילוב גודלו של מאגר הנתונים הבסיסי, היוו מכשול בפני עצמם, שכן נדרשו לי ניסויים רבים כדי להגיע להחלטה (הקשה) להתעלם מנתוני הסרטים והמשתמשים ולהתייחס רק ל-Metadata בבניית האלגוריתם.

משום שתכננתי וכתבתי את הפרויקט ב-Java, כדי לערוך את הבדיקות הנ"ל העדפתי לכתוב את רובו מחדש ב-Python, שכן קיימים עבורה מימושים ידידותיים בהרבה לאלגוריתמים שמבצעים Clustering, שאפשרו לי לבדוק יחסית בקלות גישות שונות לכתיבת האלגוריתם. ראוי לציון נוסף הוא אלגוריתם PCA, שגם לו קיים מימוש ב-Python וגם הוא הציג תוצאות מעניינות, בעיקר משום שאפשר לי ויזואליזציה אינטואיטיבית של הנתונים, גם אם הדבר לא הביא לשיפור בביצועי האלגוריתם (שכן קל יותר להבין את פעולת אלגוריתמי ה-Clustering בצורה חזותית).

במהלך הניסיונות למצוא את הפרמטרים הרלוונטיים לפיתוח אלגוריתם יעיל יותר נתקלתי גם בסוגיית עיבוד הנתונים לטובת הזנתם לאלגוריתם. משום שהנתונים ייצגו משתנים מטיפוסים שונים, הייתי צריך לבנות מבני נתונים שידעו להכיל ולהשוות משתנים בוליאניים (מיני הצופים), בדידים (למשל שנת הסרט, גילאי הצופים), רציפים (למשל ההסתברויות המשותפות לצפייה בזוגות הסרטים) ונומינליים (כמו שמות הסרטים).

אתגר משמעותי נוסף נגזר גם הוא מהעבודה עם מאגר מידע גדול יחסית. הדבר גרם לכך ש"עלות ההקמה" של התכנית בכל הרצה הייתה גבוהה ודרשה כ-15 שניות של קריאה מהקבצים בהם אוחסנו הנתונים.
כדי להתמודד עם הסוגיה השתדלתי להכניס כמה שיותר פרמטרים דינמיים ולבדוק אותם בצורה איטרטיבית: למשל, בניסיון לשפר את אלגוריתם Correlation Clustering המקורי ניסיתי לבחור קבוצות סרטים לפי הפרשי השנים בהם יצאו לאקרנים. את הבדיקה הזו עשיתי בצורה אוטומטית. הדבר דרש משמעת וסדר בפלט התכנית, שכן - כאמור - זמני הקריאה מהקבצים דרשו זמן רב בתחילת כל הרצה.

דרך נוספת לצמצם את גודל הקובץ שעליו התבססתי הייתה למיין את הסרטים לפי האינדקסים שלהם ולאחסן עבור כל סרט רק את רשימת השכנויות של הסרטים בעלי האינדקסים הגדולים ממנו. כך הקטנתי כמעט פי 2 את גודל בסיס הנתונים, שכן מטריצת השכנויות הינה סימטרית.

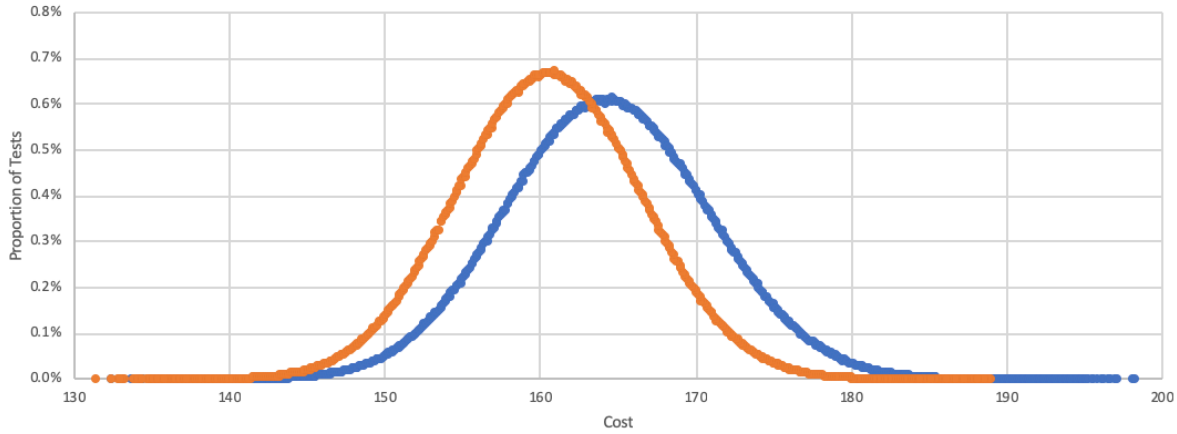
לבסוף, כדי להקטין את משקל הקובץ לטובת ההגשה, כיווצתי את הקובץ באמצעות פרוטוקול gzip. כמובן שהכיווץ לא שיפר את זמן הקריאה, אך מבחינה פרקטית חשיבותו ניכרת לא פחות.

## חלק רביעי - סטים מיוחדים

לשם אחידות וקריאות, כל תוצאות האלגוריתמים מעומדות *בדומה* לפורמט הפלט הנדרש: כל cluster בשורה נפרדת, אולם בין הסרטים השייכים לאותו cluster מפרידים טאבים.
חשוב לציין כי יש לקחת את הקביעות בחלק זה בערבון מוגבל, שכן הן תוצאה של מספר איטרציות גדול יחסית, אולם אינן מספקות הוכחה מתמטית לאופטימום כלשהו. עם זאת, הן אכן סיפקו ראייה מעניינת על תוצאות האלגוריתמים.

### סט ראשון: העלות הנמוכה ביותר

מתוך הנחה (ראו תרשים 4) שהאלגוריתם המשופר מניב פתרונות בעלי עלויות זולות יותר, בדקתי מה הסט שמניב את העלות הנמוכה ביותר (בשתי הבדיקות של סט זה עיגלתי את העלויות לפי דיוק של 0.1).




תרשים 4: ההסתברויות לקבלת עלויות שונות עבור קלטים אקראיים בגודל 20 סרטים (בכחול - האלגוריתם המקורי; בכתום - האלגוריתם המשופר)

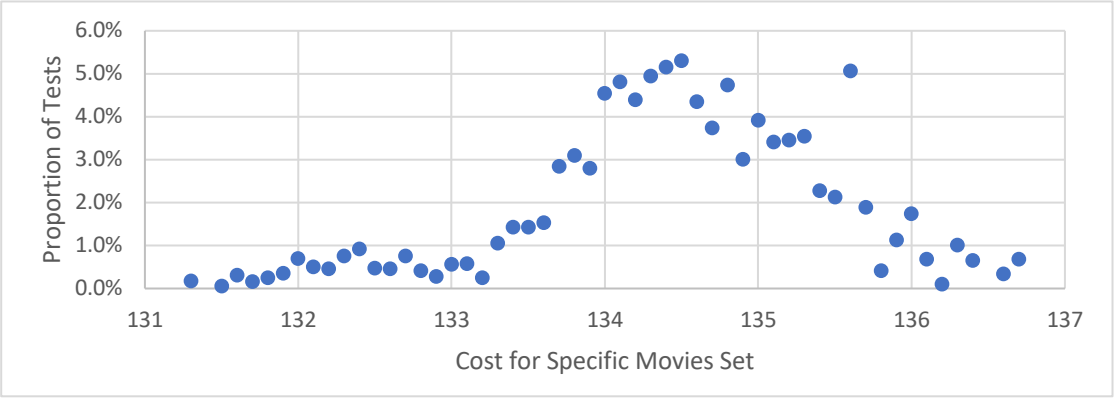
להלן הפלט שבו האלגוריתם המשופר הניב את העלות הנמוכה ביותר:

**Original Algorithm's Output (Cost: 136.34)**
3152 Last Picture Show, The
2694 Big Daddy, 3593 Battlefield Earth
3052 Dogma
1252 Chinatown
2875 Sommersby
3893 Nurse Betty
3396 Muppet Movie, The
260 Star Wars: Episode IV - A New Hope
3100 River Runs Through It, A
771 Vie est belle, La (Life is Rosey), 1221 Godfather: Part II, The, 1233 Boat, The (Das Boot), 1623 Wishmaster
2791 Airplane!
539 Sleepless in Seattle
339 While You Were Sleeping
2662 War of the Worlds, The
2494 Last Days, The
2788 And Now for Something Completely Different

**Improved Algorithm's Output (Cost: 131.51)**
260 Star Wars: Episode IV - A New Hope
3052 Dogma
3893 Nurse Betty
2662 War of the Worlds, The

3100 River Runs Through It, A
2494 Last Days, The
1221 Godfather: Part II, The, 1252 Chinatown
2694 Big Daddy, 3593 Battlefield Earth
1233 Boat, The (Das Boot), 3152 Last Picture Show, The
1623 Wishmaster, 771 Vie est belle, La (Life is Rosey)
3396 Muppet Movie, The, 2791 Airplane!, 2788 And Now for Something Completely Different
2875 Sommersby, 539 Sleepless in Seattle, 339 While You Were Sleeping

זו בדיקה מתבקשת וכמעט טריוויאלית, אולם בנוסף לכך בדקתי האם גם התוצאה של הפעלת האלגוריתם *המקורי* על אותו הסט הינה בעלת העלות המינימלית מבין הפעלות האלגוריתם המקורי. משום שהאלגוריתם המקורי אינו דטרמיניסטי, השוויתי בין $10^7$ איטרציות שונות שלו על אותו קלט (הסט שנבחר):



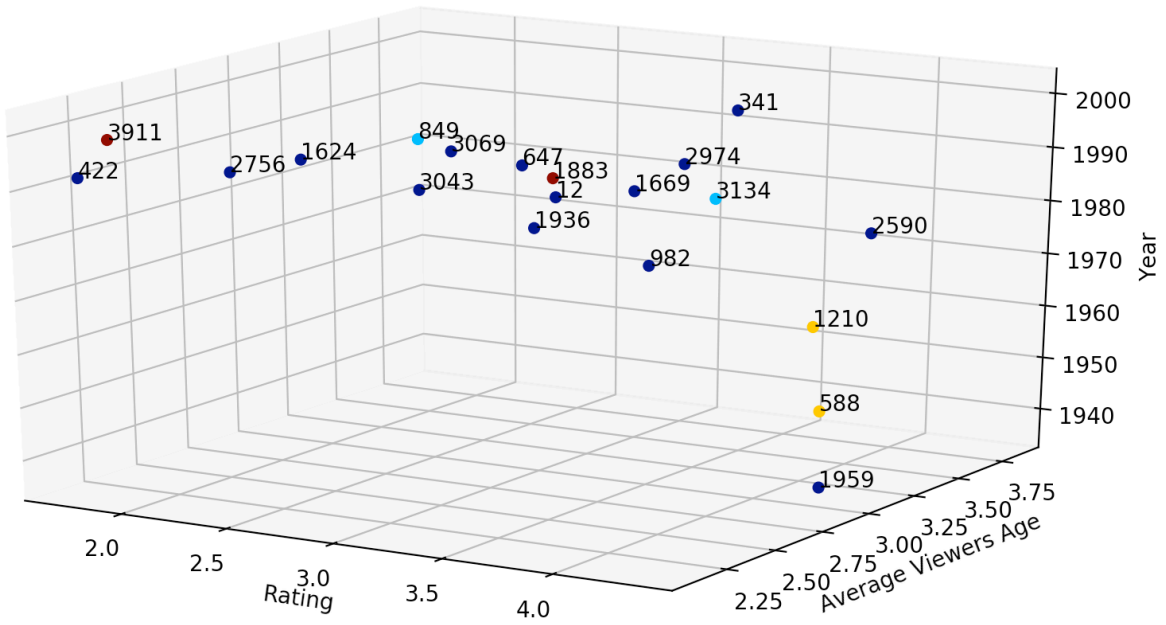

תרשים 5: ההסתברויות לקבל עלויות שונות באמצעות האלגוריתם המקורי, עבור הסט שנבחר בסעיף זה

ומבדיקה זו ניתן ללמוד שתי מסקנות מעניינות:

1. תוצאת האיטרציה שהוצגה לעיל הייתה פחות טובה ביחס לביצועי האלגוריתם המקורי.
2. בהסתברות נמוכה מאד (0.2%) האלגוריתם המקורי יכל להניב תוצאה בעלת עלות מעט נמוכה יותר מזו של האלגוריתם המשופר (131.3).

## סט שני: הסרטים השונים ביותר

לצרכי השוואה בחרתי שלושה מאפיינים "בסיסיים" לסרטים: גיל הצופים הממוצע (כאשר הגילאים מנורמלים לטווח [0,6]), דירוג הסרט הממוצע (מ-1 עד 5) והשנה בה יצא הסרט לאקרנים.
סט הסרטים הראשון שבחרתי מונה 20 סרטים, שפיזור המאפיינים הנ"ל שלהם לא נראה יוצא דופן. למרות זאת, ייחודו של הסט בכך שהאלגוריתם המשופר חילק אותו ל-17 קבוצות, מספר הקבוצות המקסימלי שהצלחתי לדגום עבור קלט בגודל 20 סרטים (ראו תרשים 3).
בתרשים הבא ניתן לראות את פיזור הסרטים לפי שלושת הפרמטרים הנ"ל, כאשר הנקודות הכחולות מייצגות את קבוצות הסרטים השונות המכילות כל אחת סרט בודד; ואילו זוגות הנקודות בצהוב, אדום ותכלת מייצגים את זוגות הסרטים השייכים לאותו ה-cluster:




תרשים 6: פיזור על גבי שלושה צירים של סט הסרטים הנ"ל, מחולק לקבוצות המיוצגות לפי צבעים

להלן פלטי האלגוריתמים עבור סט הסרטים השני:

**Original Algorithm's Output (Cost: 167.05)**
647 Courage Under Fire, 1936 Mrs. Miniver, 1624 Thousand Acres, A, 2974 Bats
422 Blink
1669 Tango Lesson, The
3134 Grand Illusion (Grande illusion, La)
2756 Wanted: Dead or Alive
849 Escape from L.A.
3911 Best in Show
3069 Effect of Gamma Rays on Man-in-the-Moon Marigolds, The
3043 Meatballs 4
588 Aladdin
2590 Hideous Kinky
1959 Out of Africa
341 Double Happiness, 982 Picnic
1210 Star Wars: Episode VI - Return of the Jedi
12 Dracula: Dead and Loving It
1883 Bulworth

**Improved Algorithm's Output (Cost: 162.80)**
1210 Star Wars: Episode VI - Return of the Jedi
588 Aladdin
3911 Best in Show
1883 Bulworth
849 Escape from L.A.
3134 Grand Illusion (Grande illusion, La)
12 Dracula: Dead and Loving It
1669 Tango Lesson, The
3069 Effect of Gamma Rays on Man-in-the-Moon Marigolds, The
3043 Meatballs 4
422 Blink
2590 Hideous Kinky
341 Double Happiness
2756 Wanted: Dead or Alive

1959 Out of Africa,
647 Courage Under Fire,
1936 Mrs. Miniver,
1624 Thousand Acres, A
2974 Bats
982 Picnic

## סט שלישי: תוצאה בלתי צפויה

המיוחד בסט הסרטים השני הוא שבאופן נדיר האלגוריתם המקורי השיג תוצאה טובה משמעותית מהתוצאה של האלגוריתם המשופר. ליתר דיוק, סט זה הניב את ההפרש הגדול ביותר בעלויות ה-Clustering, לטובת האלגוריתם המקורי (ראו תרשים 7).

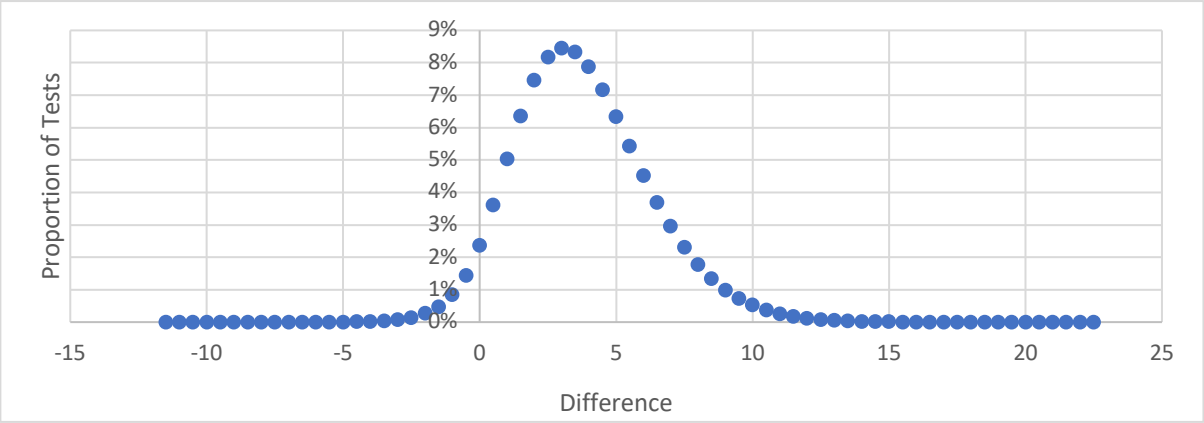
**Original Algorithm's Output (Cost: 154.03)**

1955 Kramer Vs. Kramer,
1279 Night on Earth,
128 Jupiter's Wife,
3049 How I Won the War,
274 Man of the House,
3018 Re-Animator,
3893 Nurse Betty
3462 Modern Times
1288 This Is Spinal Tap
2669 Pork Chop Hill
2867 Fright Night
508 Philadelphia
2643 Superman IV: The Quest for Peace
612 Pallbearer, The
2730 Barry Lyndon
1421 Grateful Dead
997 Caught
3757 Asylum
1760 Spice World
3760 Kentucky Fried Movie, The

**"Improved" Algorithm's Output (Cost: 163.80)**

3893 Nurse Betty
508 Philadelphia
2643 Superman IV: The Quest for Peace
1288 This Is Spinal Tap,
3462 Modern Times,
1279 Night on Earth,
612 Pallbearer, The,
2867 Fright Night,
3757 Asylum,
997 Caught,
1760 Spice World
2730 Barry Lyndon
1421 Grateful Dead
274 Man of the House
3760 Kentucky Fried Movie, The,
2669 Pork Chop Hill,
1955 Kramer Vs. Kramer,
3018 Re-Animator
3049 How I Won the War
128 Jupiter's Wife

כדי לבדוק כמה באמת מקרה זה נדיר (מתוך הנחה שהאלגוריתם ה"משופר" אכן מניב תוצאות משופרות), השוויתי בין תוצאות 100 מיליון איטרציות של שני האלגוריתמים על קלטים אקראיים בגודל 20 סרטים. מבלי לפגוע בבדיקה, עיגלתי את ההפרשים לכפולות של חצי.



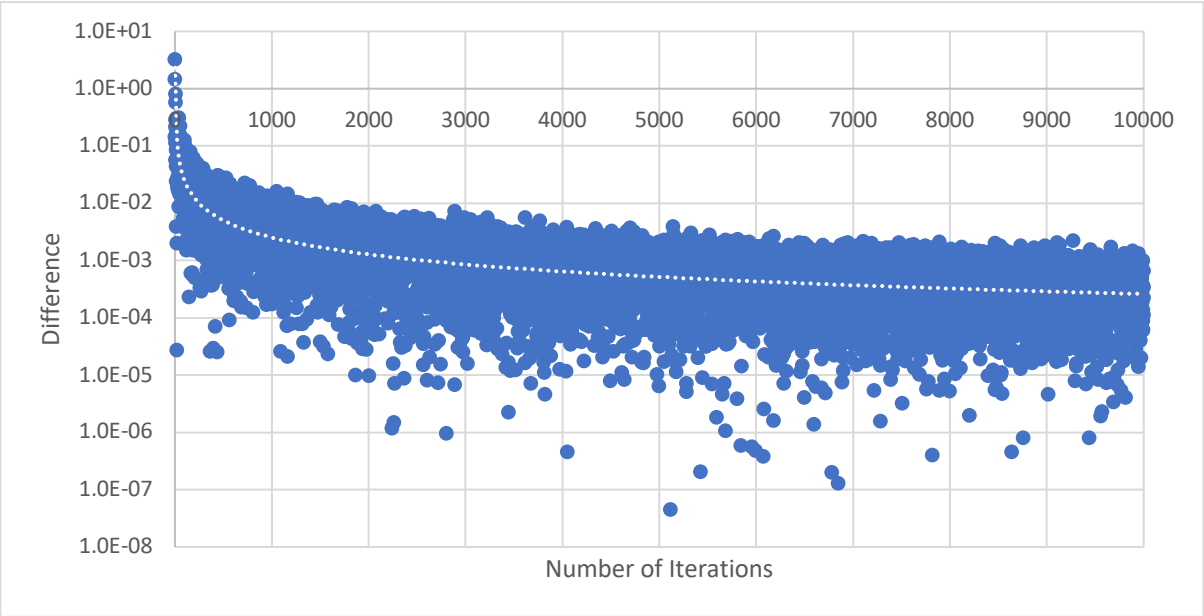

תרשים 7: ההסתברות לקבלת הפרשי עלויות של שני האלגוריתמים (לטובת האלגוריתם המשופר) עבור קלט בגודל 20 סרטים

חשוב לציין כי מספר הקונפיגורציות האפשריות לקלט האלגוריתמים גדול בהרבה מ-$10^8$ (ישנם 3883 סרטים חוקיים במאגר הנתונים ו-$3883^{20}$ הוא מספר הגדול אפילו מ-$10^{71}$, לכן גם מספר הקונפיגורציות המדויק גדול בכמה וכמה סדרי גודל ממספר הבדיקות שנערכו), אולם הדמיון הרב של התפלגות התוצאות להתפלגות נורמלית מתאים - לפי משפט הגבול המרכזי - לעובדה שהבדיקות בלתי תלויות ובעלות אותה התפלגות, ולכן היה מספיק אפילו מספר איטרציות קטן בהרבה כדי לזהות את הדפוס.

לסיכום, ניתן ללמוד כי הסיכוי שהאלגוריתם המקורי יניב תוצאה טובה יותר *כלשהי* מהאלגוריתם המשופר עומד בסך הכל על כ-3.3% ואילו הסיכוי להפרש המדויק שהוצג הינו קטן במיוחד: סביב 1 ל-$10^8$.

## חלק חמישי - סטים אקראיים לדוגמה

להלן מוצגות תוצאות הרצות האלגוריתמים על 20 סטים אקראיים של סרטים. את ממוצע העלויות של האלגוריתם המקורי חישבתי לפי $10^4$ איטרציות, שכן כבר לאחר כ-2000 איטרציות נוכחתי לגלות שהממוצע משתנה בפחות מ-$10^{-2}$ בין איטרציה לבאה אחריה.
האלגוריתם המשופר הינו דטרמיניסטי, לכן אין טעם בחישוב הממוצע של תוצאותיו.




תרשים 8: השינוי בין ממוצע תוצאות האלגוריתם המקורי כפונקציה של מספר האיטרציות

מכלל התוצאות עולה כי העלויות שהציג האלגוריתם המשופר אכן נמוכות מעלויות האלגוריתם המקורי. ספציפית לגבי 20 הסטים שנבדקו בחלק זה, השיפור הממוצע הוא של 7.3% עם חציון של 7.6%.

| Subset Index | Improved Algorithm's Cost | Original Algorithm's Cost | Average Original Algorithm's Cost |
|---|---|---|---|
| 1 | 782.78 | 832.26 | 839.36 |
| 2 | 750.07 | 805.56 | 811.86 |
| 3 | 775.16 | 843.62 | 839.89 |
| 4 | 781.56 | 844.43 | 841.69 |
| 5 | 788.70 | 851.10 | 847.71 |
| 6 | 770.83 | 838.26 | 829.30 |
| 7 | 757.69 | 799.94 | 802.57 |
| 8 | 767.15 | 841.83 | 831.98 |
| 9 | 770.54 | 835.32 | 834.06 |
| 10 | 800.32 | 881.25 | 874.38 |
| 11 | 789.11 | 858.33 | 856.31 |

| | | | |
|---|---|---|---|
| 12 | 785.04 | 858.41 | 855.19 |
| 13 | 759.24 | 810.88 | 808.58 |
| 14 | 767.24 | 818.94 | 830.47 |
| 15 | 787.44 | 860.75 | 856.71 |
| 16 | 768.48 | 830.30 | 828.81 |
| 17 | 784.51 | 848.93 | 849.19 |
| 18 | 779.49 | 837.58 | 839.67 |
| 19 | 775.35 | 832.99 | 831.28 |
| 20 | 787.97 | 848.93 | 852.21 |

## נספח - פירוט המבנה הכללי של הקוד

- **Algorithm**
  - **GraphGenerator** - A static class that generates new graphs
  - **VertexChoosingFunc** - Contains the three higher-class static functions that were used in testing new versions of the original algorithm. Given a graph, these functions choose a vertex by a certain method.
- **DataStructure** - A package that contains the classes that represent structures in the algorithm. Note that some of them might be described as objects rather than data structures.
  - **UserProps** - Contains representations of two Users' properties: age and gender.
  - **AlgorithmConfig** - A data structure that wraps the algorithm's configuration for a test: the amount of movies in the input, number of iterations, etc.
  - **Movie** - An object that wraps the properties and methods of a movie, given the basic dataset(s).
  - **NumArrayList** - Extends ArrayList<Double> mainly with the capability to calculate its elements' mean and variance.
  - **SubmissionMovie** - Extends Movie, just to meet submission format requirements
  - **User** - An object that wraps the properties and methods of a user, given the basic dataset(s).
- **Exceptions**
  - **MovieDoesntExistException** - Its name is self-explanatory
- **Graph**
  - **Cluster** - A data structure that represents a cluster, a set of vertices.
  - **ClustersGroup** - A data structure that represents an algorithm's solution: a group of clusters.
  - **Graph** - A data structure that represents a graph of vertices and edges.
  - **Vertex** - A data structure that represents a vertex in a graph.
- **Utils**
  - **Tester** - A static class that was used to run concurrent tests.
  - **Timer** - A static class that provides run-time measuring.
  - **Utils** - General-purpose static functions that I thought could be used in various contexts.
- **CorrelationClustering** - Static methods that run the two versions of the algorithm.
- **Database** - Contains static instances that hold the database during run-time.
- **Main** - The main class, which executes by default.
- **Parser** - A class that was mainly used in preparing the database that was used in the algorithms' code.
- **Reader** - A class with static methods that are used to read from external files.
- **Test** - The class that was used in order to run the tests and to build this report.
- **Writer** - A class with static methods that are used to write to external files.