בניית שני וקטורים בינאריים מתתי הוקטורים שלהם

נעם מוסבא, איתמר קרויטורו

תקציר- דף זה חוקר את הבעיה של בניית שני וקטורים בינאריים מתתי הוקטורים שלהם. תחת תבנית זו, מניחים שכל תתי הוקטורים הם מגודל קבוע וידוע מראש ושלא מתרחשות שגיאות מכל סוג, והמטרה היא, בעזרת קבלת תתי הוקטורים האלה, לשחזר מחדש את שני הוקטורים המקוריים. בעוד ישנם מאמרים ואלגוריתמים לבניית וקטור בינארי יחיד מתתי הוקטורים שלו מגודל מסוים, בין אם עם שגיאות ובין אם בלי שגיאות, בדף זה אנחנו ננסה להכליל את הבעיה לשני וקטורים ולפתור אותה בסבירות גבוהה. נבדוק איך שלושה סוגי edit distance משפיעים על הצלחת ההפרדה- נתחיל ממרחק המינג בין שני הוקטורים, נמשיך ל-(LCS)longest common subsequence ונסיים עם מרחק לוינשטיין, לאחר מכן נבדוק אילו סוגי וקטורים יעלו או יורידו את הסבירות להצלחה ולבסוף נציע פתרון שמטרתו תהיה להצליח להפריד בין תתי וקטורים ששייכים לוקטור א' לתתי וקטורים ששייכים לוקטור ב', כאשר פתרון זה יהיה מוכוון למיקסום הסתברות ההצלחה.

מוטיבציה- הבעיה שמוצגת בדף זה היא בעיה מוחלשת של בעיה מורכבת יותר- להצליח לבנות מחדש שני גדילים של DNA מקריאה רנדומלית משני הגדילים. לכן, דף זה יקדם את הדרך לפתרון בעיה מורכבת זו, ובכך בעתיד אולי לקצר את זמני הקריאה והכתיבה לגדילי DNA, שכן זאת אחת הבעיות המרכזיות בשמירת זיכרון בגדילי DNA.

סימונים- אורך וקטור יסומן בתור n. תתי הוקטורים מהגודל הקבוע ייקראו "חלון" ואורך החלון יסומן בתור w.

פרמטרים המשפיעים על הצלחת פתרון הבעיה- גודל כל חלון, גודל ההדבקה של שני חלונות ליצירת תת וקטור גדול יותר, מרחק edit על שלושת סוגיו בין שני הוקטורים, וכמות הקריאות הרנדומליות שנרצה לקרוא מכל וקטור.

אינטואיציה לבעיה

ננסה קודם לתת אינטואיציות לבעיה, מבחינת מרחק המינג בין שני הוקטורים בעיקר, לגבי האם מרחק מסוים מעלה את סיכויי ההצלחה לבניית שני הוקטורים או מוריד.

א. מרחק המינג מקסימלי לאו דווקא מבטיח הצלחה(ואפילו מעלה סיכויים לכישלון), לדוגמה- 0101…01 עם 1010…10- מרחק המינג של n אבל סיכויי הצלחה 0. כנ"ל גם מקרים כמו- 1…01…0 עם 0…10…1. לעומת זאת, אם שני הוקטורים הם 0…0 עם 1…1 אז ניתן לראות שבוודאות נצליח, וגם פה מרחק המינג של שני הוקטורים הוא n.
ב. מרחק המינג מינימלי כמו שהאינטואיציה אומרת לנו גם כן פוגע בסיכויי ההצלחה, לדוגמה- 0…01 עם 0…00.
ג. לכן התיאוריה שלנו(שנתעמק בה בהמשך) היא שמרחק המינג באזור ה-n/2 מתקרב למקסימום סיכויי הצלחה.

השפעות מרחק המינג בין זוג וקטורים על הצלחת ההפרדה:(כאשר מחברים וקטורים כמו בגרף דה-ברויין[1], בהנחה שכל החלונות של כל וקטור הגיעו)

בחלק זה נבדוק איך מרחק המינג בין זוג הוקטורים משפיע על פתרון הבעיה, כאשר נבדוק מדגם של מרחקים- מרחק המינג גדול מאוד, קטן מאוד, ובאמצע, ולבסוף נגיע לטענה שתסכם את החלק הזה.

א. כאשר מרחק המינג הוא n (כאורך הוקטורים).
נרצה לבדוק את הטענה הבאה- אם משקל המינג של אחד הוקטורים שואף ל-0 או ל-n(כרגע עבור חלון בגודל logn), ומרחק הוקטורים שואף ל-n או n בעצמו, אז סיכויי ההצלחה שואף ל-100%. נציג דוגמאות:
   a. 0…0 עם 1…1. ניתן לראות שנתוני הטענה מתקיימים ואכן נצליח בוודאות.
   b. 0101…01 עם 1010…10. משקל המינג של כל אחד מהוקטורים הוא n/2 ואכן ההסתברות להצליח להרכיב מחדש שני וקטורים אלה ע"י החלונות שלהם הוא אפסי.

[1] נקרא לזה גם "הדבקה שמרנית", כלומר, נדביק חלון w עם חלון u אם ורק אם סיפא בגודל של (w-1) של חלון w זהה לחלוטין לרישא בגודל של (w-1) של חלון u. בהמשך נתייחס גם איך הדבקות פחות שמרניות משפיעות על פתרון הבעיה.

c. 1…01…10…01…0 עם 0…10…01…10…1. כמו ב-b, גם כאן משקל המינג של כל אחד מהוקטורים הוא n/2 כאשר יש סיכוי מאוד גבוה ליצירת תת וקטור שלא שייך לשני הוקטורים בכלל.

d. 0…0110 עם 1…1001. כאן כבר לא ברור לגמרי מה הם סיכויי ההצלחה וזה תלוי ב-n.

ניתן לראות כי נראה שהטענה היא אכן נכונה, כאשר הרף שבו משקל המינג כבר לא נחשב "שואף ל-n" הוא כאשר משקל המינג של אחד מהוקטורים גדול מגודל החלון, ואז סיכויי ההצלחה קטנים משמעותית. כלומר, עבור מרחק המינג ששווה ל-n, הדבקה שמרנית, וחלון שגדול שווה מ-logn(ככל שמגדילים את החלון ההצלחה עולה), ככל שמשקל המינג של אחד הוקטורים קטן מאוד(והשני גדל מאוד בהתאמה) כך ההסתברות להצליח להפריד בין חלונות הוקטורים עולה.

ב. כאשר מרחק המינג הוא 0.

שני הוקטורים זהים ולכן או שסיכויי ההצלחה הם 0 אם האלגוריתם פועל בצורה שאם החלון הנוכחי שייך לוקטור שנבנה כרגע אז הוא ילך לשם. וסיכויי הצלחה שואפים ל-100% אם האלגוריתם מחליט באופן רנדומלי אם הוא ממשיך לבנות את הוקטור הנוכחי או לנסות להתחיל וקטור חדש(וכמובן שברגע שיש 2 תתי וקטורים שנבנים, הוא לא ינסה להתחיל עוד תת וקטור חדש).

עבור מרחקי המינג נמוכים מאוד(אך שונים מ-0).

קשה להפריד בין שני הוקטורים, הפעם באופן בלתי תלוי למשקלי המינג של 2 הוקטורים- כי הבעיה פה היא שהוקטורים דומים מדי אחד לשני(ונגדיר זאת באופן מוחלט בסעיף ג'), מה שגורם לכך שחלון שקיים רק בוקטור א' יחובר עם חלון שקיים רק בוקטור ב' ליצירת תת וקטור שלא שייך לאף אחד מהוקטורים.

ג. עבור מרחק המינג n/2.

סיכויי ההצלחה תלויים <u>*בטענה 1*</u> הבאה- עבור $a, b \in \{0,1\}$ ועבור חלון $a\bar{x} \in \bar{u}$, אם קיים גם $\bar{x}b$, אז שיהיה קיים רק בוקטור u. טענה מוחלשת תוסיף גם שאם $\bar{x}b \in \bar{v}$ אז שיהיה קיים גם בוקטור u, וגם זה בתנאי שגם מתקיים $a\bar{x} \in \bar{v}$.

קודם, נשים לב שטענה זו מכילה בתוכה פתרון לבעיות הקודמות-

אם מרחק המינג של זוג הוקטורים מאוד קטן, סבירות גבוהה שהתנאי הזה אכן יתקיים, אבל הבעיה הפעם היא שכמות הפעמים ש-$\bar{x}b$ משותף ל-2 הוקטורים היא גבוהה מדי, ולכן יהיה קשה להחליט לאן החלון ילך, כפי שהוסבר בסעיף ב'.

אם מרחק המינג של זוג הוקטורים מאוד גדול, ראינו שאנחנו תלויים במשקל המינג, וראינו שככל שמשקל המינג של אחד הוקטורים גדול מאוד, אז ההסתברות להצליח עולה, וזה אכן מסתדר עם התנאי הזה: אם משקלי המינג של שני הוקטורים הם קיצון של שני הצדדים(שואפים ל-0 או ל-n), זה מגדיל את השונות בין שני הוקטורים ולכן ההסתברות שיהיה קיים $\bar{x}b$ כזה משותף לשניהם או רק לוקטור השני v הולך וקטן.

כלומר, בסה"כ, עבור מרחק המינג בין זוג וקטורים שגדול מ-k מסויים(k תלוי לגודל החלון), ככל שהתנאי הזה מתקיים עבור יותר $a\bar{x}$ *and* $\bar{x}b$, כך ההסתברות <u>להפריד</u> בין החלונות בצורה נכונה גדלה יותר.(נשים לב שבמאמר של איתן[2] על הרכבה של וקטור מהחלונות שלו, אחת הטענות(שדומה לטענה שלנו) שהוכחו הייתה שאם כל תת וקטור בגודל של חלון פחות 1 יחידה במינה, אז ניתן להרכיב את הוקטור. בטענה שלנו, אנחנו משתמשים בטענה כדי להגיד שניתן יהיה להפריד בין חלון ששייך לוקטור א' לחלון ששייך לוקטור ב', כאשר העיקרון לשתי הטענות זהה).

נראה שקיים חסם תחתון על גודל החלון, על מנת שהטענה תהיה אפשרית:

כמות ה- $a\bar{x}$ שקיימים בוקטור הראשון u הוא n-w+1. ייתכנו חזרות, ולכן $\#(a\bar{x}) \leq n - w + 1$.

על כל $a\bar{x}$ כזה, קיימים שני $\bar{x}b$: עבור *b=0* או *b=1*.

כל $\bar{x}b$ כזה לא קיים ב-*v*, ולכן מספר ה-$\bar{x}b$ האסור מקיים $\#(\bar{x}b) = 2 * \#(a\bar{x}) \leq 2 * (n - w + 1)$.

לכן, כמות החלונות שאסור שיהיו קיימים בוקטור *v* הוא $\#(\bar{x}b)$. נבדוק חסם תחתון(לא הדוק) לגודל החלון כך שניתן יהיה בכלל ליצור וקטור *v* שעומד בתנאים האלו:

[2] https://arxiv.org/pdf/1912.11108.pdf

עבור *w=logn*, יתקיים כי מספר החלונות הכללי הוא $2^w = n$ ואילו מספר החלונות האסורים הוא $2n - 2w + 2$
ועבור $n \geq 2^3 = 8$ מתקיים כי $2n - 2w + 2 > n = 2^w$, כלומר, לא קיימים בכלל *v*'ים כאלה.
עבור *w=2logn*, יתקיים כי מספר החלונות הכללי הפעם הוא $2^w = n^2$, ולכן יתקיים כי $2n - 2w + 2 < n^2$ לכל $n \geq 3$.
לכן החסם התחתון על גודל החלון על מנת שיהיו קיימים בכלל *v*'ים רצויים הוא $w \geq 2logn$.

ננסה <u>לכמת את מספר זוגות הוקטורים שמקיימים את התנאי הזה</u>:
עבור u קבוע כלשהו, נבדוק את כמות ה-v'ים הרעים כך שיש לפחות $x\bar{b}$ אחד כזה, כלומר אם $a\bar{x}$ קיים, אז $\bar{x}b$ קיים ב-v.
נבחר חלון $a\bar{x}$ אחד כזה מ-u: n-w+1. לכל חלון כזה יש שתי אפשרויות עבור וקטור $\bar{x}b$. נבחר אקראית איפה בוקטור v החלון שנבחר יהיה: n-w+1. לבסוף, נבחר אקראית את שאר הוקטור- 2^{n-w}.
בסה"כ נקבל שיש $(n-w+1)^2 * 2 * 2^{n-w}$ כאלה.
לכן מתקיים: $\#(bad\ v's) \leq (n-w+1)^2 * 2 * 2^{n-w}$
לכל u קבוע כלשהו, מספר ה-v'ים הכלליים שקיימים הוא 2^n ולכן יתקיים-

$$\#(good\ v's) > 2^n - (n-w+1)^2 * 2 * 2^{n-w}$$

ניתן לראות שככל ש-w גדל, כך כמות ה-v'ים הרעים קטן, כלומר, הסתברות ההצלחה גדלה.

<u>השפעות גודל Longest Common Subsequence(LCS) בין זוג וקטורים על הצלחת ההפרדה</u>:(כאשר מחברים וקטורים כמו בגרף דה-ברויין, בהנחה שכל החלונות של כל וקטור הגיעו)

בחלק הקודם בדקנו את השפעות מרחק המינג בין שני הוקטורים וראינו שההשפעה שלו לא מוחלטת, אלא הוא תלוי בתנאים נוספים ובעיקר ב*טענה 1*. בעצם, *טענה 1* רוצה להגיד שאם לוקטור ב' אין תת וקטור מאורך מסויים, הקיים בוקטור א', אז הסיכוי להצלחה גדל. נוכל להכליל זאת באמצעות LCS- הוא מוצא את תת הוקטור הכי גדול(שהוא לאו דווקא רציף) שמשותף לשני הוקטורים, וככל ש-LCS גדול יותר, כך שני הוקטורים יותר דומים אחד לשני ולכן הסתברות ההצלחה קטנה. לדוגמה, ראינו שעבור הוקטורים 0...0 ו-1...1 הסתברות ההצלחה היא מקסימלית, ואכן גם ה-LCS פה הוא 0. דוגמה נוספת היא שני וקטורים alternating 01...01 ו- 10...10, וניתן לראות שה-LCS הוא בגודל n-1 שזה כמעט הכי גדול, ואכן הסתברות ההצלחה במקרה זה היא מינימלית אם לא אפסית.

באמצעות LCS ניתן למצוא את ה-edit distance בין שני וקטורים, כאשר מרשים אך ורק deletions & insertions, והנוסחה ניתנת ע"י [3]- $d'(x,y) = n + n - 2 * |LCS(x,y)|$. כלומר, ראינו מקודם שככל ש-LCS גדול יותר, כך הסתברות ההצלחה קטנה. מהנוסחה ניתן לראות שככל ש-LCS גדול יותר, כך מרחק edit בין שני הוקטורים קטן. לכן ניתן להסיק, שככל מרחק edit קטן, כך סיכויינו להצליח להפריד בין שני הוקטורים קטן.

ב-1975, Chvátal & Sankoff חקרו את המקרה בו שני וקטורים נשלפים באופן יוניפורמי מאותו אלף-בית קבוע, ומצאו את Chvátal–Sankoff constants, ואותם קבועים בעצם מסמלים את תוחלת ה-LCS בין שני הוקטורים האלה חלקי אורך הוקטור(כאשר אורך הוקטור שואף לאינסוף). כלומר, הם מצאו לקבוע הזה חסם עליון וחסם תחתון, שהוא מסמל את החלק יחסי בשני הוקטורים שמשותף. כלומר, אפשר להסתכל על זה "עד כמה שני הוקטורים דומים" כאשר הם נשלפים באופן יוניפורמי מאותו אלף-בית קבוע. כלומר, שליפת הוקטורים האלה היא באותה צורה של הבעיה שלנו, ולכן נראה שהקבוע הזה אכן רלוונטי גם לבעיה שלנו, ואותם חסמים יכולים לרמוז לנו מה הם הסיכויים להצליח להפריד בין שני הוקטורים בלי שנעשו צעדים נוספים של קידוד. כפי שמתואר כאן[4], החסם התחתון כאשר האלף-בית בינארי הוא 0.788~ והחסם העליון הוא 0.826~. כלומר, כאשר שני וקטורים בינארית נשלפים יוניפורמית, הם דומים בכמעט 80%, ולכן ההסתברות להצליח לפתור את הבעיה כמו שהיא היא אפסית. לכן נראה שכדי לפתור את הפתרון, חייב להשתמש בכלים ואמצעים כמו קידוד על מנת להצליח לשפר את הסיכויים שלנו כדי לפתור את הבעיה.

[3] https://en.wikipedia.org/wiki/Longest_common_subsequence_problem#Relation_to_other_problems
[4] https://en.wikipedia.org/wiki/Chv%C3%A1tal%E2%80%93Sankoff_constants#Bounds

<u>השפעות מרחק לוינשטיין בין זוג וקטורים על הצלחת ההפרדה</u>:

בהמשך לחלק הקודם, מרחק לוינשטיין הוא כאשר בנוסף ל-deletions & insertions מרשים גם substitutions. נשים לב שאומנם substitutions יכולים להקטין את מרחק edit(שכן במקום לעשות delete+insert אפשר לעשות ישירות sub), אך מרחק לוינשטיין לא יוסיף לנו פיענוח נוסף על זה הנאמר קודם, שכן עדיין יתקיים שככל שמרחק edit, בין אם עם substitutions ובין אם בלי, קטן יותר, כך סיכויי ההצלחה שלנו קטנים גם הם.

לכן, נראה שמתוך סוגי מרחקי ה-edit, בעוד מרחק המינג לא פועל בצורה חד משמעית, מרחק לוינשטיין כן פועל בקורולציה עם סיכויי ההצלחה, ולכן כדאי להתמקד דווקא בו.

<u>השפעות סוגי וקטורים על הצלחת ההפרדה בין חלונות השייכים לוקטור אחד לשני</u>: (כאשר מחברים וקטורים כמו בגרף דה-ברויין, בהנחה שכל החלונות של כל וקטור הגיעו)

בחלק זה נבדוק האם *טענה 1* יכולה להיות מנוסחת שונה- הפעם לא נסתכל על תתי חלונות, אלא על החלונות עצמם- איך שוני בין החלונות של שני הוקטורים משפיע על ההסתברות שלנו להצליח להפריד בין חלונות ששייכים לוקטור אחד לשני, בהינתן שלא ידוע מראש משהו על הוקטורים האלה.

א. תחילה, היינו רוצים ש-2 הוקטורים יהיו שונים בחלונות שלהם אחד מהשני(כדי להצליח להפריד ביניהם) וכן בתוך כל וקטור(כדי להצליח להרכיב את הוקטורים) -
כמות חלונות אפשריים: 2^{w}
כמות חלונות בוקטור יחיד: $\mathrm{n}-\mathrm{w}+1$
לכן, כדי שלכל וקטור יהיו חלונות שונים אחד מהשני- $2*(\mathrm{n}-\mathrm{w}+1) \;=\; 2\mathrm{n}-2\mathrm{w}+2$.
ראינו בחלק הקודם שאנחנו צריכים שיתקיים $w \geq 2logn$.

ב. אם נרצה רק שלכל וקטור, החלונות ששייכים אליו יהיו שונים לגמרי מהחלונות של הוקטור השני אז- החישוב יהיה דומה לחסם על "good v's" שחושב בחלק הקודם.

אבל, עבור שני הסעיפים, עדיין אנחנו עלולים לבצע הרכבה של חלון של וקטור א' לחלון של וקטור ב'.
לדוגמה-

$$\mathrm{v}_1 = \cdots 010010 \ldots \rightarrow windows: 0100,1001,0010$$
$$\mathrm{v}_2 = \ldots 001100 \ldots \rightarrow \mathrm{windows}: 0011,0110,1100$$

עדיין אפשר לקחת את חלון מוקטור 1 שהוא 1001 ולהרכיב עם 0011 ו-0110 שהם חלונות מוקטור 2 ולקבל 100110 שהוא תת וקטור שונה משני תתי הוקטורים שאיתם התחלנו.
לכן נראה שהתנאי הזה לא מספיק, ו*טענה 1* תספיק לנו לבינתיים.

<u>השפעות גודל החלון</u>:

עד עכשיו חיפשנו תמיד גודל חלון מינימלי על מנת שנצליח. עכשיו נעשה סקירה קצרה על איך גודל החלון יכול לעזור לפתרון הבעיה, אך במחיר.

אחד הקשיים בבעיה שאנו מנסים לפתור הוא כאשר יש רצפים זהים לחלוטין שמופיעים במקומות שונים לאורך הוקטור. לדוגמה, בוקטור 0011010011, הרצף 0011 מופיע פעמיים, ולכן מקשה על להחליט איפה יהיה כל אחד מהמופעים האלה, ואולי בכלל קראנו את אותו חלון פעמיים? לכן, הגדלת גודל החלון יכולה לעזור- אם נגדיל את החלון, נראה שבמקום 0011 נקבל 00110, ועבור החלון השני נקבל 10011- 2 חלונות שונים שעכשיו ניתן יהיה להפריד ביניהם. לכן ככלל אצבע- חלון גדול יותר עוזר להתמודד עם קונפליקטים בצורה יותר טובה. אבל, נשים לב שקריאה גדולה יותר נחשבת יקרה יותר(במיוחד בתחום ה-DNA) ולכן בכלל קיימת הבעיה שאנחנו מנסים לפתור(אחרת היינו קוראים חלונות בגודל n וסיימנו).

<u>השפעות גודל ההדבקה</u>:

בדומה לגודל החלון, עד עכשיו ניסינו להיות כמה שיותר שמרניים בנוגע לגודל ההדבקה של שני חלונות נתונים, וכך יהיה גם בהמשך. אך נשים לב שזה גורר הדבקה איטית יותר- כל פעם אנחנו מגדילים את הוקטור הנבנה רק ב-1, ולכן נצטרך לעשות n-w+1 צעדים כאלה. אך ככל שנרשה להדביק בצורה מקלה יותר- לדוגמה, מספיק w/2 מהביטים שיהיו זהים- כך מצד אחד נרכיב את הוקטור מהר יותר, ואילו מצד שני הסיכוי לטעות גדל. לכן קיים פה trade off בין מהירות פעולת האלגוריתם לבין הסתברות ההצלחה.

<u>הצעות לפתרונות הבעיה בעזרת קידוד</u>: (כאשר מחברים וקטורים כמו בגרף דה-ברויין)

1. נתחיל מפתרון נאיבי(שיעבוד רק אם הבעיה שלנו מתקבלת בסגנון ה-DNA – כל וקטור אנחנו אלה שמפרקים אותו לחלונות ושמים אותו במאגר, ואחרי זה קוראים אותם רנדומלית. הוא לא יעבוד עבור וקטור שקוראים ממנו ממש רנדומלית כל הזמן[5]) הוא לאנדקס כל חלון ולהוסיף ביט שקובע האם החלון שייך לוקטור א' או ב'. בעוד הפתרון הזה יביא להצלחה של 100%(בהנחה שכל החלונות יגיעו), העלות של הפתרון גדולה מדי. אם נניח אפילו שגודל של חלון הוא n/2, זה אומר שיש כפי שראינו
$$\#(windows) = n - w + 1 = \mathrm{n} - \frac{\mathrm{n}}{2} + 1 = \frac{\mathrm{n}}{2} + 1$$
חלונות שונים לוקטור מסוים. על מנת להצליח לאנדקס את כמות החלונות הזאת, נקבל $\log(\frac{\mathrm{n}}{2} + 2)$ ביטים רק לאינדוקס. וזה יהיה עבור כל חלון, כלומר, כמות הביטים הנוספים שנצטרך רק בשביל הקידוד הזה הוא- $\left(\frac{\mathrm{n}}{2} + 1\right) * \log(\frac{\mathrm{n}}{2} + 2)$.
כדי לקבל סדר גודל, אם נניח ש- $\mathrm{n} = 2^{20}$ אנחנו נקבל שכמות הביטים רק עבור הקידוד הנוסף גדול מ-
$$2^{19} * \log(2^{19}) = 19 * 2^{19} > 16 * 2^{19} = 2^{23}$$
כלומר, כמות תאי היתירות היא הרבה יותר מדי גדולה, ולכן נזניח את הפתרון הזה.
2. פתרון נוסף(שעובד עבור כל סוג של שמירת וקטורים- בין אם כמו DNA ובין אם קוראים אקראית מוקטור שאנחנו לא יודעים מהו), הוא <u>קידוד</u>:
<u>ההצפנה</u>: לפני כל ביט בוקטור א' נוסיף אפסים, ונוסיף אחדות לפני כל ביט בוקטור ב'.
מטרת פתרון זה הוא להצליח להפריד בין חלונות ששייכים לוקטור א' לשל וקטור ב', בהנחה כי ניתן יהיה להרכיב כל וקטור בנפרד עם החלונות ששייכים אליו, על סמך קיום אלגוריתמים קיימים הפותרים בעיה זו.
<u>*הבחנה*</u>- לוקטור א' המוצפן אין שני 1ים רצופים, ולוקטור ב' המוצפן אין שני 0ים רצופים. הוכחה- עבור מוצפן א', בין כל ביט בוקטור המקורי שמים 0 לפניו וגם אחריו(אם הוא לא אחרון), ולכן או שנקבל 010 עבור ביט מקורי שהוא 1, או 000 עבור ביט מקורי שהוא 0. אם הוא בהתחלה אז האפשרויות הן 00 או 01 ועבור ביט שהוא אחרון האפשרויות הן 00 או 01. לכן מתקיימת ההבחנה. באופן סימטרי עבור מוצפן ב'.
לכן, נקבל שהחלונות של מוצפן א' שונים מכל החלונות של מוצפן ב', אלא אם כן יש alternating, כלומר 010101 או 101010.
קודם נסביר למה שאר החלונות בהכרח שונים- אם החלונות הם לא ה-alternating, זה אומר שיש או שני 0ים לפחות רצופים בחלון או שני 1ים לפחות רצופים בחלון. אם יש שני 0ים רצופים- החלון שייך לוקטור א', אם יש שני 1ים רצופים- החלון שייך לוקטור ב', וזה על סמך ה*הבחנה* שהוכחנו קודם.
חלון alternating כזה מתקבל מוקטור א' אם היה בוקטור המקורי(לפני ההצפנה) תת וקטור של 11...1 ומוקטור ב' אם היה בוקטור המקורי תת וקטור של 00...0.
לכן הבעיה המרכזית של הפתרון היא להתמודד עם חלון שהוא alternating.
לפני שננסה לפתור את הבעיה, נשים לב כי למרות שיש "המון" תתי וקטורים שלא שייכים לאף וקטור מוצפן- כל חלון שיש לו לפחות שני 1ים רצופים **וגם** לפחות שני 0ים רצופים, לא מצאנו איך להשתמש בהם בלי לפגוע בהצפנה(שכן צריכים לבחור תת וקטור ש"יסתדר" גם עם שאר הוקטור המוצפן שנמצא לפני ואחרי תת הוקטור הזה) ועם שמירה על חלונות שונים בין שני הוקטורים(כלומר, שנצליח בוודאות להפריד בין חלונות ששייכים לוקטור א' לחלונות ששייכים לוקטור ב').

[5] אחרי קריאת הפתרון המלא, ניתן לראות כי האינדקסים לאו דווקא יתחברו לוקטורים שבאים אחריהם.

ננסה לפתור את הבעיה, מכיוון שאלה בדיוק 2 חלונות בעיתיים, נניח שאם מקבלים אותם, אנחנו נעביר 0...0 לוקטור א' ו-1...1 לוקטור ב'. פתרון נוסף הוא לבצע הפוך(וכך אכן מימשנו את הפתרון)- נתעלם מהוקטורים הבעייתיים- לא נסווג לאף אחד מקבוצות הוקטורים את החלון הזה, על סמך המאמר[6] שמראה שניתן להרכיב מחדש וקטור במקרה שחסר בו חלון.

<u>פענוח</u>:
סיווג החלון- כל חלון בו יש לפחות שני 0ים רצופים, יהיה שייך לוקטור א', כל חלון שיש בו לפחות שני 1ים רצופים יהיה שייך לוקטור ב'. הנכונות של הסיווג מסתמכת על ה*בחנה*.
פענוח החלון- ישנן שתי שיטות שניתן באמצעותן לפענח את תת הוקטור המקורי, לאחר סיווג נכון:

א. לשמור מראש טבלה בעת ההצפנה, כך שלכל חלון יהיה החלון המקורי המפוענח.
ב. עבור תת-וקטור ששייך לוקטור א':
עבור i=0,1:
עבור j=i, נתחיל מהאינדקס ה-i, ולכל j נבדוק האם הביט ה-j הוא 0, כאשר אחרי כל בדיקה נבצע j=j+2. אם ענינו על כל ה-j כן- אז הוקטור המקורי יתחיל מהביט ה- (1-i) וימשיך בקפיצות של שתיים מהביט הזה. אחרת, נעבור ל-i הבא.
בוודאות עבור אחד מה-i יתקיים התנאי, אחרת, סתירה לאופן ההצפנה(לשים 0 לפני כל ביט מקורי).
באופן סימטרי בין 0 ל-1 עבור תת-וקטור ששייך לוקטור ב'.

<u>יתירות</u>: נשים לב כי היתירות כאן היא כאורך הוקטור המקורי, שזאת יתירות גבוהה מאוד שמשלמים על מנת שנצליח בכמעט 100% להצליח להפריד בצורה נכונה בין חלונות ששייכים לוקטור א' לחלונות ששייכים לוקטור ב'. הבעיה היא ששיטת סיווג החלון המוצעת עובדת רק עבור שיטת הצפנה זו, ואילו אם נשנה את ההצפנה שתהיה "לשים 0ים כל logn ביטים" וכדומה, נצטרך להחליט בצורה מסוימת בעת קבלת חלון איפה ביט הסיווג הזה נמצא, והשיטה הנוכחית לא תעבוד בכלל, נבחן זאת בהמשך.

נשאר לבדוק- כמה חלונות נצטרך לקרוא אקראית כדי שבהסתברות גבוהה נצליח להוציא את כל החלונות השייכים לכל אחד מהוקטורים?
נשים לב שקיימים 2 חלונות- שני ה-alternating- שיפגעו בהסתברות ההצלחה(כי אנחנו לא משייכים אותם לאף וקטור), אך מכיוון שהם קבועים(תמיד יש בדיוק שניים כאלה), ואילו כמות החלונות גדלה בלפחות פי 2 כל פעם שמגדילים את הוקטור פי 2, 2 החלונות האלה זניחים מבחינת ההשפעה שלהם וייבלעו בהסתברות לאי בחירת חלון מסוים.
הסתברות לאי בחירת חלון מסוים- $\frac{n-w}{n-w+1}$.
כמות הפעמים שנצטרך לקרוא היא לפחות n-w+1, ולכן עבור k טבעי יתקיים- $\left(\frac{n-w}{n-w+1}\right)^{k*(n-w+1)}$.
מתקיים:
<u>גודל הוקטור</u> **לא** משפיע על אחוזי ההצלחה(אלא על כמות הזיכרון והזמן הכללי של האלגוריתם),
<u>גודל החלון</u> **לא** משפיע על אחוזי ההצלחה(גם כן משפיע בעיקר על כמות הזיכרון והזמן הכללי של האלגוריתם).
לכן נתמקד בכמות הקריאות.
נשים לב שזה רק עבור וקטור מוצפן אחד, ולכן נצטרך פי 2 קריאות מהחישוב הזה(עבור שני הוקטורים), כלומר אנחנו בעצם נקרא $2k(n-w+1)$ חלונות, ולכן יתקיים שהסתברות לאי בחירת חלון מסוים- $\left(\frac{n-w}{n-w+1}\right)^{2k*(n-w+1)}$.

[6] https://arxiv.org/pdf/1912.11108.pdf

על ידי הצבת k ומספרים, תיאורטית יתקיים:

הסתברות לאי בחירת החלון בהינתן ה-k	k
0.135	1
0.018	2
0.002	3
0.0003	4

ניתן לראות מהטבלה שקריאה של פי k=4 מכמות החלונות בוקטור יחיד נותנת סיכוי של רק 0.0003 שיהיה קיים חלון שלא נקרא.
מבחינת גודל החלון, נצטרך פי 2 מגודל החלון הדרוש כדי להצליח לבנות וקטור מהחלונות שלו. לכן במינימום, גודל החלון יהיה 4logn(ראינו בחלקים הקודמים ש-2logn הוא חסם תחתון על גודל החלון על מנת *שטענה 1* תתקיים, ולכן סביר שכל אלגוריתם ידרוש גודל חלון של לפחות 2logn ביטים).

נשים לב שנקבל שכל תת-וקטור מקורי יופיע פעמיים, לדוגמה אם היה קיים תת וקטור 1101 בוקטור א' המקורי(לפני ההצפנה) ואחרי ריפוד נקבל 01010001, אז נקבל בחזרה את 110 גם כשנקרא(עבור w=6) את 010100 וגם כשנקרא 101000, לכן אפשר להתחשב בזה בהסתברות לאי בחירת חלון מסוים, כלומר ההסתברות לאי בחירת חלון מסוים הוא בעצם $\frac{n-w-1}{n-w+1}$, וזה יכול להשפיע גם על כמות ההעתקים. בנוסף, נשים לב שהאלגוריתם צריך לדעת שהוא עלול לקבל מכל חלון לפחות שני העתקים.
נבדוק מה סיבוכיות הזמן והמקום של הפתרון:
<u>סיבוכיות זמן</u>- עבור n שהוא אורך הוקטור **המקורי**, ההצפנה היא O(n) צעדים, בפענוח, הסיווג לוקח O(log(2n)) צעדים, וכן שיחזור תת הוקטור המקורי לוקח O(log(2n)) צעדים, בסה"כ O(log(2n)). כמות החלונות הכללית הנדרשת היא k(2n-w+1), ולכן הפענוח מתבצע k(2n-w+1)*log(2n) פעמים, כלומר, מכיוון ש-k מספר טבעי נמוך(קטן מ-10) הפענוח בסה"כ לוקח O(nlog(2n)) צעדים. ולכן בסה"כ, סיבוכיות הזמן היא O(nlogn).
<u>סיבוכיות מקום</u>-
אם בוחרים בפתרון שבו נשמור טבלה, אז במקרה הכי גרוע הטבלה תהיה בגודל של 2(2n-w+1).
לאחר מכן, בונים 2 קבוצות בגודל של כל החלונות האפשריים, נקבל שסך כל החלונות שלנו יהיו k(2n-w+1), כלומר O(n). לכן עבור הפתרון עם הטבלה נקבל סיבוכיות של O(k*10n)=O(n) ואילו לפתרון בלי הטבלה נקבל סיבוכיות O(k*2n)=O(n).

<u>מימוש הפתרון</u>-
לקחנו אורכי מילה מגודל 2^x, אורכי חלון מגודל 2logn, גודל k משתנה בין 1 ל-4.
אנו מייצרים זוג מילים בינאריות באורך מסוים באמצעות פונקציה ()rand. לאחר מכן מקודדים אותן עם הריפוד אפסים ואחדות לפני כל ביט בהתאם למילה, ואז בצורה רנדומלית חותכים $2k(n-w+1)$ חלונות בגודל 4logn משתי המילים ביחד (פי שתיים מגודל חלון מקורי מכיוון שאורך מילה מקודדת גדול פי שתיים). כדי שנוכל לבדוק את מקור כל חלון, אנו יוצרים שני מאגרים של כל החלונות שקיימים בכל אחת מהמילים המקוריות לפני הקידוד, וממיינים את המאגרים.
האלגוריתם יעבור על כל החלונות המקודדים, יבדוק לפי הקידוד שלהם מאיזו מילה הם הגיעו, ויבדוק במאגר החלונות המקוריים של המילה שהם אכן קיימים שם ושעוד לא מצאנו אותם (על מנת להימנע מכפילויות). האלגוריתם יאסוף בשני מאגרים את כל החלונות שהוא מצא על מנת להעביר אותם לשלב הבא של בניית המילים מהחלונות שלהן. וגם יסכום את כל החלונות מכל מילה שהוא מצא ויחשב את אחוז הכישלון – כמה חלונות לא הצלחנו למצוא.

<u>סיבוכיות זמן</u>-
סיבוכיות הזמן של הפונקציה ()rand היא לינארית O(1). ולכן עבור בניית המילים הרנדומליות וקידודן נקבל O(n). יצירת החלונות המקודדים תהיה ב-k*4logn*(2n-2w+2) אך מכיוון שכמו שציינו k הוא קבוע, נקבל O(nlogn). בניית מאגר החלונות המקורים יעשה בצורה זהה, ומיונו יעשה באמצעות פונקצית הספרייה ()sort למערכים שסיבוכיותה היא

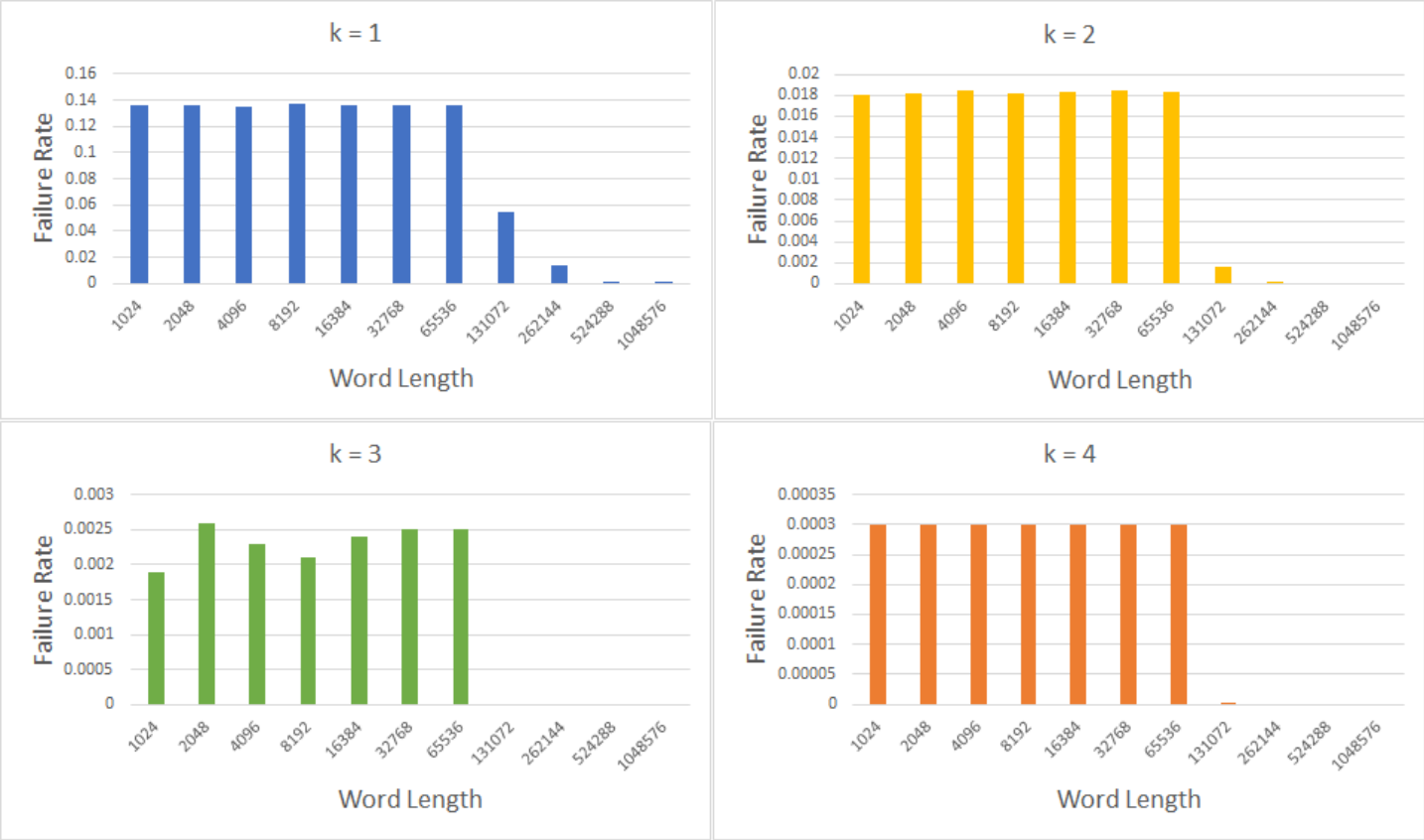
O(nlogn), ולכן גם עבור שלב זה נקבל O(nlogn). השלב האחרון של מעבר על מאגר החלונות המקודדים, מכיוון שיש לנו k*(2n-2w+2) כאלה, כאשר עבור כל חלון אנו מזהים לאיזו מילה הוא שייך ומפענחים אותו לצורתו המקורית ב-O(4logn), ומכניסים אותו למאגר החלונות שמצאנו. נקבל סה"כ ששלב זה יהיה O(nlogn) גם הוא, ולכן כל האלגוריתם רץ בזמן O(nlogn).

<u>סיבוכיות מקום-</u>

שומרים מחרוזות עבור המילים עצמן, ומערכי מחרוזות עבור כל החלונות במילים, וכל החלונות המקודדים שיצרנו, ולכן סיבוכיות המקום היא O((2n-2w+2)*k), מכיוון ש-k מספר טבעי נמוך, נקבל סך הכל סיבוכיות O(n).

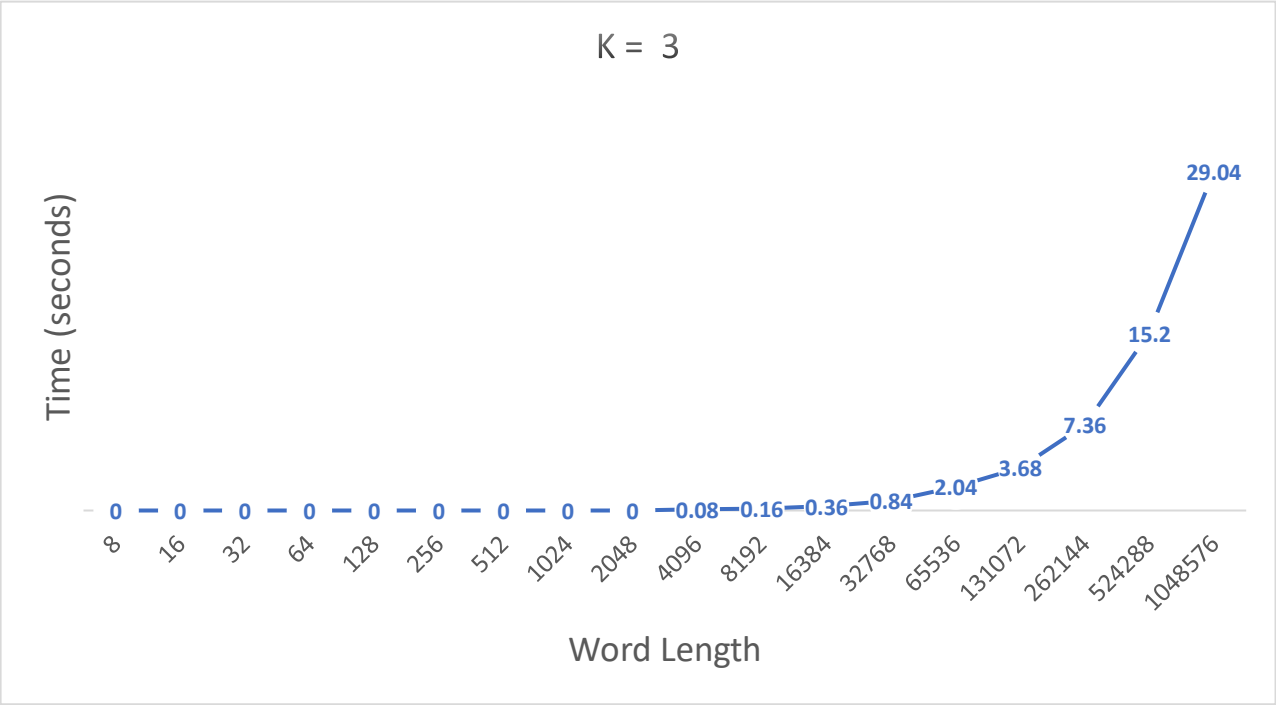
<u>גרף א' – אחוז כישלון האלגוריתם באורכי מילה שונים-</u>

בארבעת הגרפים הבאים ניתן לראות כי אחוז הכישלון של האלגוריתם (כמות החלונות שלא הצלחנו למצוא מתוך הכמות הכללית) הוא זהה בין הסימולציות השונות בלי תלות באורך המילה. כאשר החל מגדלים של 2^{17} (131072) הנתונים מתחילים לצנוח ושואפים לאפס מסיבות שלא הצלחנו לפענח. השערתינו לסיבה לכך היא שככל שאורך המילה גדל, יש יותר סיכוי לחזרה על חלונות במילה. בצורת המימוש הנוכחית, המיקום של כל חלון מקודד במילה המקורית אינו נשמר, ומכיוון שאנו חותכים את החלונות המקודדים באופן רנדומלי, אם קיים זוג חלונות זהים במיקומים שונים, נקבל שמצאנו את שניהם בשלושה מקרים שונים, כאשר באמת חתכנו את שניהם או כאשר חתכנו את אחד מהם פעמיים. וכתוצאה מכך נקבל עלייה בכמות ההצלחות של האלגוריתם. השוני בין ארבעת הגרפים הוא בגודל k, כאשר הנתונים תומכים בחישוב התיאורתי המוצג בעמוד 7. נציין שלמרות שנראה שהנתונים לא משתנים בין הגרפים, סקלת אחוז הכישלון בציר ה-y היא אחרת וקטנה בערך פי עשר בין גרף לגרף, ולכן ניתן לראות כי בכל הגדלה של k ב-1, התקבל אחוז הצלחה גבוה פי עשר. הנתונים המוצגים הם ממוצעים של 100 סימולציות.



גרף ב' – זמן הרצת האלגוריתם כאשר k=3 באורכי מילה שונים-

בגרף הבא התמקדנו בk=3, החלטנו כי אחוז הכישלון שהוא מספק – 0.0025 הוא גבוה מספיק והכי יעיל אל מול זמן ההרצה הדרוש. הגרף מציג כמה זמן לקח לאלגוריתם לרוץ עבור אורכי מילים משתנים, הנתונים המוצגים הם ממוצעים של 100 סימולציות. ניתן לראות כי בכל הכפלת אורך המילים, כמות הזמן שתיקח לאלגוריתם היא בערך פי שתיים, שכן הנתונים מסתדרים עם סיבוכיות הזמן של האלגוריתם nlogn.



שיפור למימוש הפיתרון-

על מנת לשפר את יתירות הקידוד, ניתן לבצע הצפנה שונה במקצת. ניתן להוסיף בוקטור א' אפסים כל $\lfloor logw \rfloor$ ביטים, ובוקטור ב' להוסיף אחדות כל $\lfloor logw \rfloor - 1$ ביטים. עבור הפענוח, נחפש בחלון המקודד את כל תת סדרות הביטים בו המתחילות ב-$\lfloor logw \rfloor$ המיקומים הראשונים בחלון בקפיצות שמתאימות לסדר הכנסת האחדות או האפסים. אם הצלחנו לבנות שרשרת אפסים לאורך כל המילה בקפיצות של $\lfloor logw \rfloor + 1$, או שרשרת אחדות לאורך כל המילה בקפיצות של $\lfloor logw \rfloor$, אז החלון שייך לוקטור המתאים.

נראה אינטואיציה באמצעות דוגמא, נניח כי אורך חלון מקודד הוא 20, $\lfloor logw \rfloor = 4$, $\lfloor logw \rfloor + 1 = 5$. חלון אפשרי שקודד מהמילה הראשונה יהיה מהצורה-

$$\ldots x_{i-8}x_{i-7}||0x_{i-6}x_{i-5}x_{i-4}x_{i-3}0x_{i-2}x_{i-1}x_ix_{i+1}0x_{i+2}x_{i+3}x_{i+4}x_{i+5}0x_{i+6}x_{i+7}x_{i+8}x_{i+9}||0\ldots$$

נבחר את החלון המוצג בין הסמנים ||, ניתן לראות כי גם אם כל הביטים המקוריים בחלון לפני הקידוד היו אחדות, החלון המקודד היה נראה כך:

$$\ldots 11||01111011110111101111||0\ldots$$

כל ניסיון לבנות שרשרת אחדות בקפיצות של $\lfloor logw \rfloor = 4$, ייכשל:

$$\ldots 11||01111011110111101111||0\ldots \quad \ldots 11||01111011110111101111||0\ldots$$

$$\ldots 11||01111011110111101111||0\ldots \quad \ldots 11||01111011110111101111||0\ldots$$

יתירות השיטה החדשה-

עבור וקטור א' שבו יש r ביטים בין כל ביט יתירות, נקבל שהיתירות היא $\frac{r}{n+r}$, ומכיוון ש-$r > 0$, נקבל כי $\frac{(n+r)}{r} > \frac{1}{2}$, כלומר, מבחינת יתירות הפתרון הזה הרבה יותר טוב מהפתרון המקורי.

על מנת שתמיד נצליח לסווג את החלון המוצפן לקבוצה המייצגת את הוקטור המקורי ממנו הוא הגיע, נצטרך שתתקיים "התנגשות"- עבור מיקום מסוים בחלון, הצפנת 0 כל logw ו-1 כל (logw)-1 ביטים תגרום לכך שתתקיים התנגשות וודאית בין ההצפנות כל ((logw -1) * logw) לכל היותר. כך יקרה המצב שלא יכול להיות חלון שלא נדע לאיזה וקטור מקורי הוא שייך ולכן הסיווג תמיד יצליח.
הבעיה בפתרון הזה הוא שעבור סיווג של חלון מסוים, יכול לקרות מקרה בו יהיו שני פענוחים אפשריים לחלון. לדוגמה, עבור החלון המוצפן 01000101 ועם קפיצות של 3 בין ביטי הצפנה של 0, נוכל להוריד את ה-0 הראשון והחמישי ונקבל חלון מקורי של 100101, או שנוכל להוריד את ה-0 השלישי והשביעי ונקבל 010011, ולא נוכל לדעת איזה מהחלונות הוא החלון המקורי ששייך לוקטור המקורי.
לכן, למרות שסיווג החלונות מוצפנים יעשה בצורה חד משמעית ובלי טעויות, תהיה בעיה לפענח את החלון המוצפן בצורה חד משמעית.

בנוסף, נשים לב שאנחנו צריכים לשמור על הדרישה שהחלונות המפוענחים יהיו בגודל של לפחות 2logn, כי כפי שראינו קודם זה צורך בסיסי על מנת שיהיה סיכוי להצליח לבנות וקטור יחיד מהחלקים שלו. לכן, נרצה שגודל של חלון מוצפן יהיה גדול ממש מ-2logn. עבור כל גודל של מילה, נחשב עבורה את הגודל 2logn. אחרי זה, נשים לב כמה תאי יתירות נכנסים בכל גודל של חלון, ולפי זה נדע להגדיל את גודל החלון של המילה המוצפנת. אחרי חישובים פשוטים, נקבל את הטבלה הבאה:

2,3	3,4	4,5	5,6	6,7	7,8	8,9	9,10	10,11	11,12	12,13	13,14	14,15	15,16
8	15	24	35	48	63	80	99	120	143	168	195	224	255
12	20	30	42	56	72	90	110	132	156	182	210	240	272

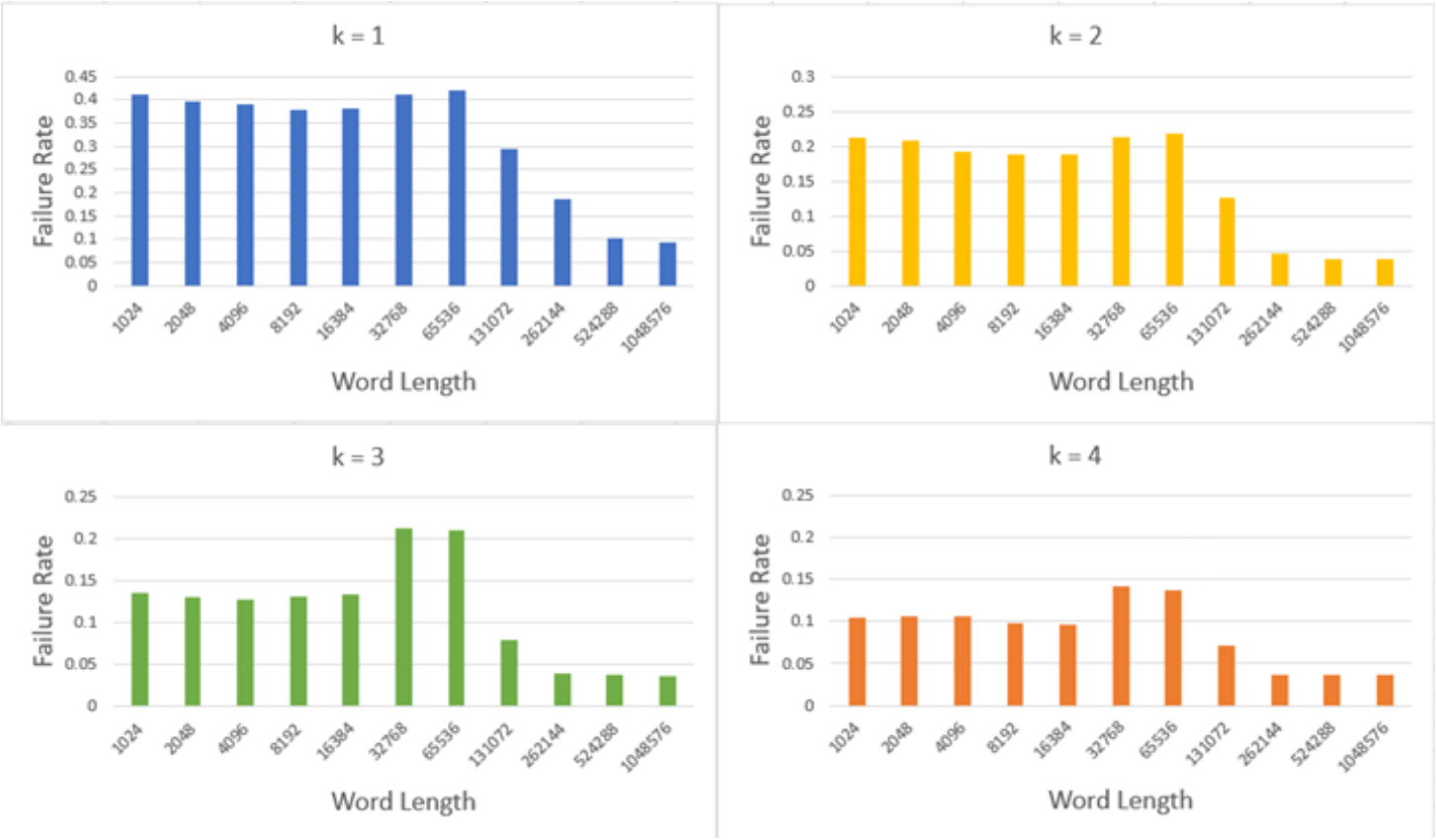
כאשר השורה הראשונה היא עבור מרווחים בין ביטי יתירות, השורה השנייה עבור גודל החלון המקורי שישאר לאחר פענוח, והשורה האחרונה עבור גודל חלון מקודד.

כלומר, ניתן לראות שכמות תאי היתירות משפיעה ישירות על גודל חלון מוצפן- ככל שיש יותר ביטים בין כל תא יתירות, כך גודל החלון גם גדל, הקשר נובע מהצורך לסיווג הוקטורים לוקטורי מקור, על מנת שתתרחש "התנגשות" נצטרך גודל חלון גדול יותר עבור רווחים גדולים יותר. מכיוון שגם גודל חלון גדול מדי זה יקר וגם יתירות גבוהה זה יקר, נבדוק בסימולציות איך זה משפיע על אחוזי הכישלון וכך כל אחד יוכל לבחור לו את האיזון בין גודל חלון, יתירות, ואחוזי הכישלון.

המימוש שלנו-

במימוש שלנו פעלנו לפי הטבלה שלעיל, ורצינו לבדוק את ההשפעה של הפתרון המשופר(מבחינת יתירות). בנוסף, כל חלון שניתן לפענח ביותר מצורה אחת, בחרנו להתעלם ממנו, כדי שבאחוזי הכישלון הוא ייספר בתור חלון שפוספס בגלל סיבה זו, וכך נראה עד כמה הפתרון הזה באמת משפר אם בכלל. יתר הקוד נשאר זהה למימוש הקודם.

בגרף הבא ניתן לראות את הקשר בין אחוזי הכישלון לגודל k באורכי מילה משתנים. כמו שראינו בפתרון המקורי- ישנה צניחה של הנתונים החל מוקטורים באורך 131072, ההשערה שלנו לסיבה לכך זהה למקרה הקודם. ניתן לראות כי אחוזי ההצלחה לא משתנים באופן יחסי לאורך המילה, נציין כי ישנה שונות מאוד גדולה בנתונים שקיבלנו בין ההרצות, אנו מייחסים את הדבר לטיפול בוקטורים בעלי פענוח דו משמעי. מכיוון שכל הוקטורים רנדומלים, יהיה הבדל בתוצאות האלגוריתם עבור וקטורים עם תתי וקטורים שכאלה. הסיבה הזו היא גם הסיבה המרכזית לירידה באחוזי ההצלחה של האלגוריתם, מכיוון שאנו פוסלים תתי וקטורים בעלי פענוח דו משמעי, נאלץ לרדת באחוזי ההצלחה לטובת פתרון עם יתירות יותר טובה. הנתונים המוצגים הם ממוצעים עבור 100 סימולציות.



מבחינת זמנים בחרנו לא להראות את הזמנים מכיוון שאנחנו חושבים שהפתרון החדש אמור להיות יעיל ומהיר יותר, אך לא שיפרנו מספיק את הקוד של הסימולציה והוא כרגע רץ באותה המהירות כמו הפתרון המקורי.

בגרף הזה ניתן לראות איך גודל הקפיצה בין כל ביט יתירות וכתוצאה מכך גודל החלון, משפיעים על אחוזי ההצלחה עבור וקטור באורך $2^{20} = 1048576$, ו-k=3, הנתונים המוצגים הם ממוצעים עבור 100 סימולציות.
ניתן לראות כי ככל שניקח מרווחים גדולים יותר, וכתוצאה מכך ניצור יתירות קטנה יותר, נקבל אחוזי כישלון יורדים.
בנוסף נשים לב כי ישנו מינימום אפשרי לאחוזי הכישלון בגרף, והקטנת היתירות מעבר לכך לא משפרת את התוצאות.
לכן רואים פה trade-off בין כמות תאי יתירות וגודל חלון-
ככל שיש פחות תאי יתירות, כך גודל החלון גדל, ואחוזי הכישלון יורדים.
וככל שיש יותר תאי יתירות, וגודל החלון קטן, אחוזי הכישלון עולים.
על מנת להחליט מה היא היתירות הכי טובה לקידוד, נאלץ לאזן זאת אל מול גודל חלון, ולכן יש לקבוע קונבנציה לפיה מה יותר יקר, תאי יתירות או קריאה של גדלי חלונות גדולים, וניתן להסתמך על גרף זה על מנת למצוא את נקודת הביניים והיחסים ביניהם.