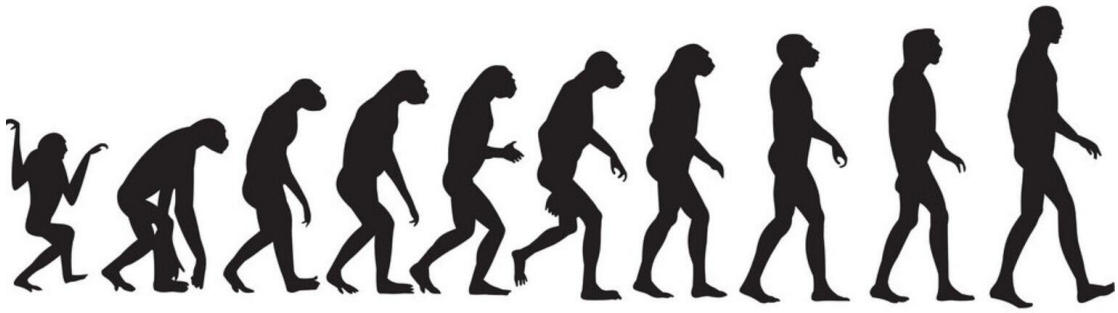
אוניברסיטת בן-גוריון בנגב

הפקולטה למדעי הטבע

המחלקה למדעי המחשב

דו"ח מסכם

נושאים בחישוב מונחה טבע



מגישים: יונתן ששוני 205916265

יוסי כרמלי 204752406

אופיר מינץ 309988731

אייר תשפ"א

מאי 2021

1. מבוא

הבעיה אותה ניסינו לפתור היא בעית הכיסוי בקודקודים.

<u>הגדרה:</u> יהי גרף $G = (V, E)$. נקרא ל-C כיסוי בקודקודים של הגרף G אם מתקיימים התנאים הבאים:

1. $C \subseteq V$
2. לכל צלע $(u, v) \in E$ מתקיים: $u \in C$ או $v \in C$

<u>בעית כיסוי בקודקודים:</u>

בהינתן גרף $G = (V, E)$ יש למצוא כיסוי בקודקודים בגודל מינימלי.

<u>חשיבות הבעיה:</u> בעית הכיסוי בקודקודים אומנם עלולה להישמע כבעיה מתמטית טהורה, אך היא מופיעה גם בעולם האמיתי בצורות שונות ומגוונות. לדוגמא, בעיה בה נרצה לקבוע את מיקומן של מצלמות מהירות, באופן בו נעשה שימוש במספר מינימלי של מצלמות אשר יצליחו לכסות את כל השטחים הרלוונטיים.

הבעיה הוכחה כבעיה NP קשה, כלומר לבעית הכיסוי בקודקודים לא קיים פתרון יעיל. לכן ננסה לקרב את הפתרון האופטימלי בעזרת אלגוריתם אבולוציוני.

<u>מהו אלגוריתם אבולוציוני?</u>

- אלגוריתם אבולוציוני כללי מורכב מחמש פונקציות עיקריות:
  1. *Initialization* – מציאת פתרון התחלתי (בדר"כ רנדומלי) המייצג את הדור הראשון.
  2. *Fitness* - מתן ציון עבור פתרון כלשהו.
  3. *Selection* - בחירה של המופעים אשר יעמידו צאצאים לדור הבא.
  4. *Crossover* - שילוב הפתרונות שנבחרו בשלב 3, ויצירת צאצאים אשר יהוו את הדור הבא.
  5. *Mutation* - יצירת מוטציה בקרב הצאצאים.
- האלגוריתם פועל באופן הבא:
  1. בעזרת הפונקציה *Initialization* ניצור את הפתרונות ההתחלתיים שלנו, דור 0.
  2. לכל פתרון התחלתי נחשב איכותו בעזרת הפונקציה *Fitness*.
  3. במשך מספר דורות שנקבע מראש נבצע:
     - 3.1 נבחר את המופעים שיעמידו צאצאים לדור הבא בעזרת *Selection*.
     - 3.2 נחשב את הדור הבא בעזרת המופעים שנבחרו בשלב הקודם, בעזרת *Crossover*.
     - 3.3 נבצע מוטציה על המופעים שהתקבלו בדור החדש.
     - 3.4 נמדוד את איכות המופעים של הדור החדש בעזרת *Fitness*.
     - 3.5 נחזור ל-3 .
  4. נחזיר את המופע שקיבל את ה *Fitness* האופטימלי מבין כל הדורות שחושבו במהלך ריצת האלגוריתם.

2. השיטה: תיאור הגישה לפתרון הבעיה

<u>גישה ראשונה אלגוריתם *Custom*:</u>

- בהינתן גרף עם n קודקודים, בחרנו לייצג כיסוי בעזרת מחרוזת בינארית בגודל n, כך שהקודקוד v_i שייך לכיסוי אם"ם התו ה-i במחרוזת שווה ל-1. אחרת התו שווה לאפס כלומר הקודקוד v_i לא שייך לכיסוי.
- *Initialization* - מימשנו על ידי הגרלת מחרוזות בינאריות.
- *Fitness* - הענקנו ציון גבוה יותר למחרוזת המייצגת כיסוי חוקי, וציון גבוה עוד יותר כלל שהכיסוי קטן יותר.
- *Selection* - מימשנו על ידי *fitness proportionate*, גישה לפיה בוחרים את המופעים מהדור הקודם בהגרלה, והסיכוי של מופע להיות מוגרל תלוי ב *Fitness* שלו. כלומר ככל שה *Fitness* של מופע גדול יותר, כך גם הסיכוי של המופע להיות מוגרל גדול יותר.
- *Crossover* - בהינתן שתי מחרוזות בינאריות שרשרנו חלקים שונים מן המחרוזות זה לזה.
- *Mutation* - מימשנו על ידי הפיכה של ביט בודד במחרוזת.

<u>גישה שנייה אלגוריתם *Article*:</u>

על מנת לבצע *Fitness* ניתן להגדיר פונקצית שמחזירה עונש עבור כיסוי לא חוקי ואינה מענישה כיסוי חוקי, כמו כן הפונקצית תעניש כל קודקוד שנמצא בכיסוי וכך נוכל לתת עונש גדול יותר לכיסוי המכיל מספר רב יותר של קודקודים.

אנחנו לא מעוניינים שפונקצית ה *Fitness* תהיה תלויה גם בקשתות וגם במספר הקודקודים שמכיל כיסוי, אלא נרצה שהיא תהיה תלויה רק באחד מהמשתנים הללו. לשם כך נקבע את מספר הקודקודים.

על מנת לקבע את מספר הקודקודים, עלינו למצוא את מספר הקודקודים בכיסוי מינימלי. לשם כך אנו נבצע חיפוש בינארי אחר ערך מספר הקודקודים. כאשר נמצא כיסוי חוקי מגודל k נחפש כיסוי קטן יותר, וכאשר לא נמצא כיסוי מגודל k נחפש כיסוי גדול יותר.

בהינתן גודל כיסוי k נחפש אחר כיסוי חוקי על ידי הפונקציות הבאות:

- *Initialization* – נגריל מחרוזות בגודל k.
- *Fitness* – עבור מופע בעל k קודקודים בדיוק, נחזיר את מספר הקשתות שלא מכוסות על ידי מופע זה.
- *Selection* – נבחר את המופעים בעלי ה *Fitness* הגבוה ביותר.
- *Crossover* – בהינתן שני מופעים, מצאנו את הקודקודים שנמצאים במופע הראשון ולא בשני וגם את הקודקודים שנמצאים במופע השני ולא בראשון. לאחר מכן החלפנו חצי מהקודקודים שמצאנו בין שני המופעים.

- *Mutation* – עשינו שימוש בשתי מוטציות שונות. בכל פעם נבחר בהסתברות שווה להשתמש באחת מהן, עבור 5% מהמופעים שהתקבלו מפונקצית ה *Crossover*. הפונקצית הראשונה מגרילה 5% מהקודקודים במופע ומחליפה אותם בקודקודים שלא נמצאים במופע. הפונקצית השנייה גם כן מגרילה 5% מהקודקודים במופע ומחליפה אותם בקודקודים המכסים קשתות שעד כה המופע לא הצליח לכסות.

על מנת להעריך את ביצועי האלגוריתמים שלנו נשווה אותם לשני אלגוריתמים נוספים:

<u>אלגוריתם *Greedy*</u>: אלגוריתם חמדן המאתחל כיסוי C כקבוצה ריקה. לאחר מכן מבצע מעבר על צלעות הגרף, ועבור כל צלע $(u,v) \in E$: אם $u \notin C$ וגם $v \notin C$, אזי $C = C \cup \{u,v\}$.

<u>אלגוריתם *LP*</u>: תכנון לינארי הוא אלגוריתם אופטימיזציה שניתן למימוש בזמן פולינומי, אשר מנסה למזער פונקצית מטרה כלשהי תחת אילוצים אשר הוגדרו מראש. עבור בעית הכיסוי בקודקודים ניתן לעשות זאת על ידי מזעור פונקצית המטרה $min \sum_{v \in V} x_v$ תחת האילוצים הבאים: $\forall v \in V: x_v \in \{0,1\}$, $\forall (u,v) \in E: x_u + x_v \geq 1$. כאשר ניתן לחשוב על x_v כאינדיקטור למאורע האם v שייך לכיסוי. ניתן להעמיק בנושא <u>כאן</u>.

3. הגדרת הניסוי

הצעד הראשון שלנו היה ניסיון לפתור את הבעיה בצורה ישירה, כפי שלמדנו מסדרת הרצאות בנושא האלגוריתמים האבולוציוניים, ולשם כך ביצענו את השלבים הבאים.

- יצירת פתרונות התחלתיים המיוצגים על ידי מחרוזות בינאריות.
- הגדרת פונקציית *Fitness* שתפקידה להבדיל בין איכותו של כיסוי אחד לאחר. ניסינו להגדיר מספר רב של פונקציות מסוג זה, אשר ניתן לחלק אותן לשתי קבוצות. האחת קבוצת פונקציות אשר מחזירות ציון גבוה יותר לכיסוי טוב יותר, כאשר איכות הכיסוי נמדדת לפי הקריטריונים של חוקיות הכיסוי וגודלו. השנייה היא קבוצה של פונקציות עונש, כלומר הפונקציות מחזירות ציון נמוך יותר עבור כיסוי טוב יותר ומתחשבות באותם קריטריונים.
- מימוש פונקצית ה *Selection* המייצגת את תהליך "הבחירה הטבעית", מימשנו בהצלחה את *fitness proportionate*.
- מימוש שיטת הזיווג, פונקצית *Cross-Over*, לפיה הצאצאים יהיו כיסויים היברידיים המורכבים מהוריהם. להלן חלק מהפונקציות אותן מימשנו:
  - חציה – בהינתן שתי מחרוזות בינאריות P_1, P_2 חצינו את המחרוזת הראשונה לשני חלקים $P_{1,1}, P_{1,2}$ ובאופן זהה נחצה ונקבל $P_{2,1}, P_{2,2}$. נחזיר שני צאצאים המוגדרים באופן הבא $C_1 = P_{1,1}P_{2,2},\ C_2 = P_{2,1}P_{1,2}$.

    לדוגמא: בהינתן שני הורים $P_1 = 0110100010, P_2 = 1010001000$ נחזיר את הצאצאים $C_1 = 0110101000, C_2 = 1010000010$.

- נקודה רנדומלית – הפונקציה מגרילה אינדקס במחרוזת שמתקבלות ומבצעת את החלוקה לפיו, במקום לבצע את החלוקה באמצע המחרוזת, ומכאן מבצעת את אותו התהליך שמבצעת פונקצית החציה.
- חלוקה יוניפורמית – חילקנו את כל אחת ממחרוזות הקלט לתתי מחרוזות מאורך שתיים ושזרנו את חלקי המחרוזות אחת בשניה כדי להרכיב את הצאצאים.

- מימוש פונקצית המוטציה אשר תופעל בהסתברות שנקבעה מראש על כל אחד מהצאצאים שהתקבלו על ידי פונקצית ה *Cross-Over*. נציג את הפונקציות שמימשנו :
  - הפיכה - הגרלת אינדקס במחרוזת, והפיכה הביט באינדקס שהתקבל.
  - החלפה - הגרלה של שני אינדקסים שונים במחרוזת, והחלפת מיקומם. במילים אחרות במידה וקיבלנו אינדקס שערכו 1, ואינדקס נוסף שערכו 0 נחליף את הקודקוד שנמצא כעת בכיסוי בקודקוד שלא נמצא בכיסוי.

בצעד השני התבססנו על מאמר המציע פתרון נוסף ומורכב יותר לפתרון הבעיה. כעת נתאר את השלבים אותם מציע המאמר.

- תחילה נבצע קירוב לגודל הכיסוי המינילי על ידי האלגוריתם החמדן שהוצג לעיל, נסמן את גודל הכיסוי שהוא מחזיר ב g.
- לאחר מכן נתחיל בביצוע חיפוש בינארי אחר ערכו של הכיסוי המינימלי ונתחיל מהתחום $[\frac{g}{2}, g]$.

כפי שנאמר כך בעצם נקבע בכל שלב את מספר הקודקודים k. במידה ומצאנו כיסוי בקודקודים מגודל k, נמשיך בחיפוש הבינארי ונבדוק האם קיים כיסוי קטן יותר, אחרת נבדוק האם קיים כיסוי גדול יותר, עד אשר נגיע להתכנסות ונגלה את הערך המינימלי שקיים לו כיסוי בקודקודים.

- כעת בהינתן k נבדוק האם קיים כיסוי בעזרת הפונקציות הבאות.
- פונקצית ה *Initialization* המגרילה מופעים של כיסויים מגודל k, כלומר מחרוזות בינאריות בהן מספר התווים מסוג *'1'* הוא k.
- פונקצית ה *Fitness* מחזירה ציון נמוך יותר ככל שהכיסוי מכסה מספר רב יותר של צלעות, כאשר כיסוי חוקי יקבל ציון מינימלי שערכו אפס. לפונקצית זו תפקיד כחלק ממימוש האלגוריתם האבולוציוני אשר מחפש כיסוי מגודל קבוע k. אך יש לה תפקיד נוסף והוא לכוון את החיפוש הבינארי. נסביר, אם הציון של כיסוי שהתקבל בשלב הנוכחי הוא 0, נדע שמצאנו כיסוי בגודל זה ועלינו להמשיך לחפש כיסוי קטן יותר. לעומת זאת, אם נקבל כיסוי עם ציון גדול מאפס, יהיה עלינו לחפש כיסוי גדול יותר.
- פונקצית ה *Selection* בוחרת את הכיסויים הטובים ביותר, כלומר בעלי ציון ה - *Fitness* הנמוך ביותר.

- פונקצית ה *Cross-Over* ממומשת באופן הבא. עבור כל כיסוי שנבחר על ידי *Selection* נגריל הורה/כיסוי נוסף אשר גם כן נבחר על ידי *Selection*. נסמנם C_1, C_2. בנוסף נסמן את קבוצת הקודקודים ששייכים ל C_1 ולא ל C_2 ב S_1 , ובאופן דומה את
קבוצת הקודקודים ששייכים ל C_2 ולא ל C_1 ב S_2. עבור כמחצית מהקודקודים בקבוצות הללו נבצע בכיסוי C_1 החלפה בין קודקוד ששייך לכיסוי מ S_1 לבין קודקוד שלא שייך לכיסוי מ S_2.
- פונקצית ה *Mutation* בוחרת על ידי הטלת מטבע הוגן להפעיל אחת מהפונקציות הבאות.
  - הראשונה מוציאה קודקוד מהכיסוי בהסתברות 0.05 ומכניסה במקומו קודקוד שלא נמצא בכיסוי.
  - השנייה מוציאה קודקוד מהכיסוי בהסתברות 0.05 ומכניסה במקומו קודקוד שלא נמצא בכיסוי וגם מכסה צלע שלא מכוסה על ידי הכיסוי הנוכחי.

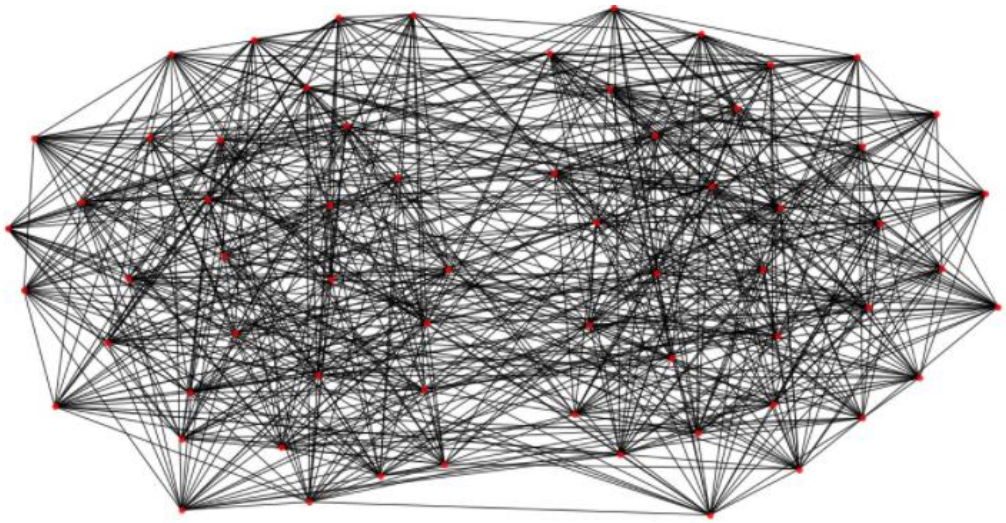
4. תוצאות

בחנו את ארבעת האלגוריתמים שתיארנו בחלק "השיטה: תיאור הגישה לפתרון הבעיה".

4.1. ניסוי ראשון

בניסוי הראשון עשינו שימוש בפרמטרים הבאים:

- *Hamming6-4 Graph* - גרף בעל 64 קודקודים ו704 צלעות אשר נלקח מ-http://networkrepository.com/dimacs.php.



- אוכלסייה בגדול 500
- מספר דורות 1000
- הסתברות למוטציה 0.05

4.1.1. תוצאות אלגוריתם *Custom*:

לאחר מספר ניסיונות נוכחנו לגלות שהתוצאות הטובות ביותר הגיעו לאלגוריתם זה משימוש בפונקצית *Fitness* המענישה לפי קריטריונים של חוקיות וגודל כיסוי מינימלי. פונקצית *Cross-Over* המגרילה את מיקום חציית המחרוזות. ופונקצית *Mutation* המבצעת החלפה.

<u>תוצאות הריצה</u>: לאחר כ 1000 דורות לא הצלחנו להגיע לכיסוי. התקשנו ביצירת פונקציית *Fitness* כך שתעודד יצירת כיסויים חוקיים מלכתחילה מה שגרם לכך שקיבלנו כיסויים לא חוקיים כפלט. הגענו למסקנה שהפונקציה לא מספיק טובה, כי היא אינה מבטיחה את חוקיות הכיסוי.

4.1.2. תוצאות אלגוריתם *Greedy*:

האלגרותים החמדן החזיר כיסוי בגודל 64, כלומר כיסוי הכולל בתוכו את כלל הקודקודים בגרף.

4.1.3. תוצאות אלגוריתם *LP*:

אלגוריתם התכנון הלינארי החזיר כיסוי בגודל 52.

4.1.4. תוצאות אלגוריתם *Article*:

החיפוש הבינארי מתחיל עם גבול עליון $h = 64$ וגבול תחתון שהתקבל על ידי האלגוריתם החמדן $\ell = \frac{g}{2} =$ 32, ולכן נתחיל את החיפוש שלנו מהערך $k = \frac{h}{\ell} = 48$.

מהלך הריצה: הכיסוי המיטבי שמצאנו לאחר כ 1000 דורות עבור $k = 48$ קיבל ציון 8 מפונקצית ה *Fitness*, כלומר הכיסוי אינו חוקי. לכן נחזור על החיפוש עבור $k = 56$, עבור ערך זה מצאנו לאחר כ 10 דורות כיסוי חוקי. הערך הבא הוא $k = 52$ גם הפעם לאחר קטן של דורות, כ 26 נמצא כיסוי חוקי. הערכים הבאים הם $k = 50,51$ עבורם לא נמצאו כיסויים חוקיים, ולבסוף עבור $k = 52$ אכן נמצא כיסוי חוקי אותו יחזיר האלגוריתם.

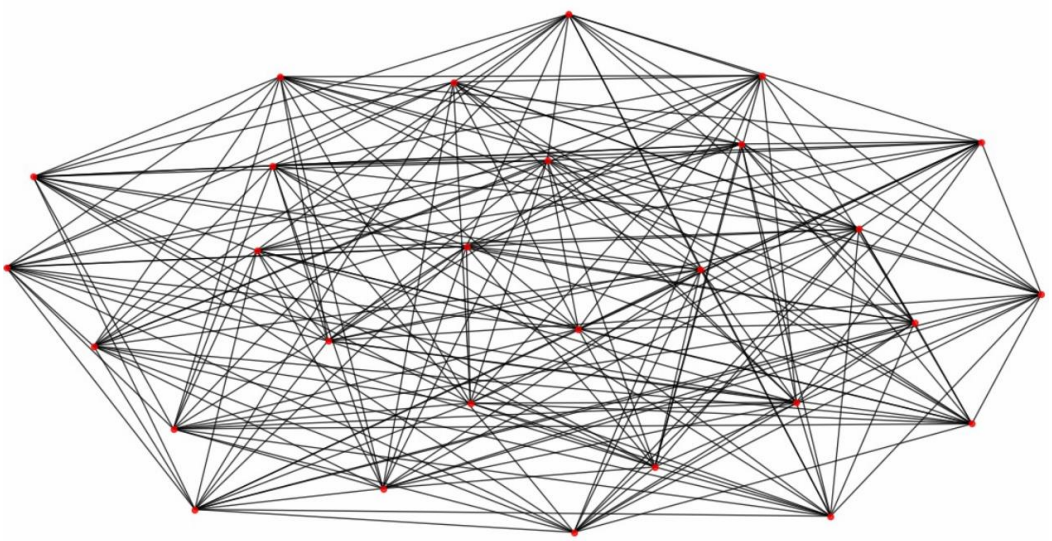
4.1.5. ריכוז תוצאות:

אלגוריתם / מאפיין	*LP*	*Greedy*	אלגוריתם *Custom*	אלגוריתם *Article*
מהירות	0:01 דקות	0:00:5 דקות	2:30 דקות	8:47 דקות
גודל כיסוי	52	64	34	52
כיסוי חוקי	*V*	*V*	*X*	*V*

4.2. ניסוי שני

בניסוי השני עשינו שימוש בפרמטרים הבאים :

- *Johnson8-2-4 Graph* - גרף בעל 28 קודקודים ו420 צלעות אשר נלקח מ- http://networkrepository.com/dimacs.php.



- אוכלסייה בגדול 500
- מספר דורות 1000
- הסתברות למוטציה 0.05

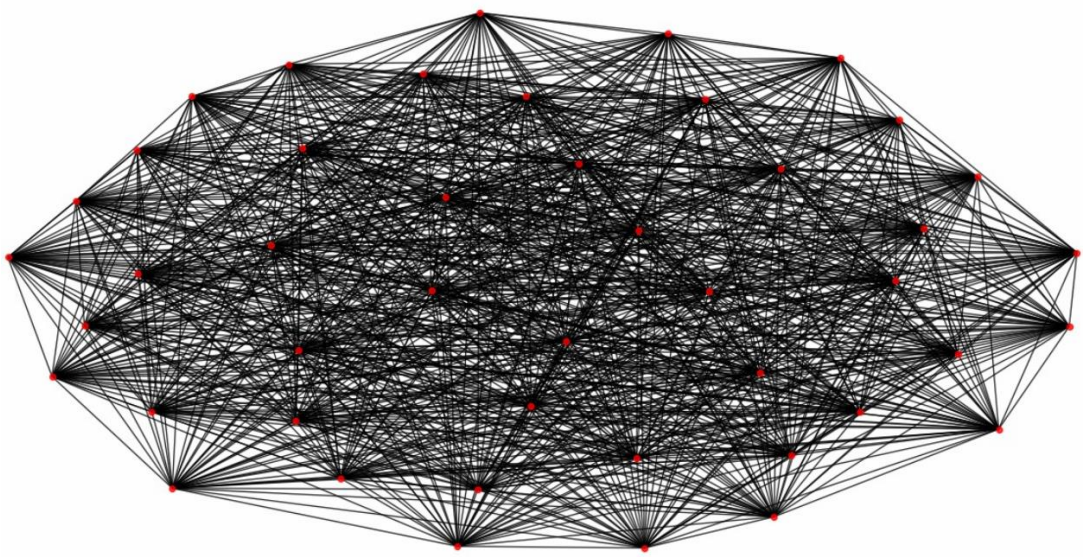
4.2.1. ריכוז תוצאות:

אלגוריתם / מאפיין	*LP*	*Greedy*	אלגוריתם *Custom*	אלגוריתם *Article*
מהירות	חצי שניה	4 מילי שניות	40 שניות	2 דקות
גודל כיסוי	21	26	14	21
כיסוי חוקי	*V*	*V*	*X*	*V*

4.3. ניסוי שלישי

בניסוי השלישי עשינו שימוש בפרמטרים הבאים:

- *MANN-a9 Graph* - גרף בעל 45 קודקודים ו918 צלעות אשר נלקח מ- http://networkrepository.com/dimacs.php.



- אוכלסייה בגדול 500
- מספר דורות 1000
- הסתברות למוטציה 0.05

4.3.1. ריכוז תוצאות:

אלגוריתם / מאפיין	*LP*	*Greedy*	אלגוריתם *Custom*	אלגוריתם *Article*
מהירות	חצי שניה	חצי שניה	2 דקות	4.5 דקות
גודל כיסוי	42	44	22	42
כיסוי חוקי	*V*	*V*	*X*	*V*

5. מסקנות, לקחים ומחשבות לשיפורים עתידיים

5.1. מסקנות:

1. מבחינת זמנים התכנון הלינארי והאלגוריתם החמדן הינם המהירים ביותר.
2. התכנון הלינארי והאלגוריתם *Article* הניבו תוצאות מיטביות זהות.
3. אלגוריתם *Custom* לא הצליח למצוא כיסוי חוקי.
4. אומנם החמדן הניב כיסוי חוקי ומהיר אבל הוא החזיר פתרון טריוויאלי המכיל את כלל הקודקודים בגרף.

5.2. לקחים:

למרות שפונקצית *Fitness* יכולה להשמע פשוטה למימוש, היא יכולה להיות תלויה במספר פרמטרים שונים ולא בהכרח תוביל לתוצאות הרצויות. כפי שראינו אלגוריתם ה *Custom* נכשל במציאת כיסוי חוקי, אך לאחר קביעה של אחד המשתנים אלגוריתם *Article* צלח ומצא כיסוי חוקי ומינימלי.

5.3. שיפורים עתידיים:

1. באלגוריתם *Article* בכל שלב בו נעשה חיפוש אחר כיסוי מגודל k, האלגוריתם האבולוציונים מגיע לתוצאה לאחר כ 20 דורות. לפיכך כדי לשפר את זמן ריצתו נוכל להקטין את פרמטר מספר הדורות ל 100.
2. כפי שכבר הזכרנו, נרצה למצוא פונקצית *Fitness* מתאימה עבור אלגוריתם *Custom*, אשר תגרום לו להחזיר כיסוי חוקי.
3. ביצוע תהליך *Cross-Validation* על מנת לקבוע את הערך המיטבי עבור הפרמטרים.
4. עבור אלגוריתם *Custom* כדי לעודד כיסויים חוקיים, נוכל להכניס לדור 0 מופעים חוקיים, לדוגמה הפתרון של האלגוריתם החמדן, או הפתרון הטריוויאלי הכולל את כל קודקודי הגרף.

6. רשימת מקורות

1. סדרת הרצאות – גיא כתבי - אלגוריתמים אבולוציוניים (חלק 1 מתוך 7)*YouTube* -
2. מאמר - *Experience the power of the Genetic Algorithm | by Abhishek Mungoli | Towards Data Science*
3. הקוד - *EvolutionaryAlgorithmVertexCover-/MiniProject at main · YonatanSasoony/EvolutionaryAlgorithmVertexCover- (github.com)*