# עבודת הגשה – ניתוח פתרונות לבעיית VC – מבוא לאופטימיזציה (89589)

מגישים: נועם רוט (212450431), טל סיגמן (322888389)

במסגרת עבודה זו נציג את בעיית כיסוי הקודקודים (vertex cover) בגרף ונסביר למה היא בעיה לא טריוויאלית. נציג את הפתרון הפשוט (brute force) ונסביר את הבעייתיות שבו. לכן, נציג את האלגוריתמים שלנו לפתרון בעיה זו ע"י (1) פתרון בעזרת תכנות לינארי (2) פתרון באמצעות אלגוריתם גנטי.

לאחר הצגת האלגוריתמים, נציג את התוצאות מההרצות שלהם ונשווה ביניהם.

## תיאור הבעיה

בעיית vertex cover (בעיית כיסוי הקודקודים) היא בעיה מתמטית מתחום תורת הגרפים, בה מחפשים תת-קבוצה לא-טריוויאלית של קודקודי הגרף, כאשר תת-הקבוצה מכילה לפחות קודקוד אחד הנמצא בקצה כל קשת בגרף. מכיוון שכל תת-קבוצה בגודל $n-1$ קודקודים מקיימת תנאי זה[1] (קבוצת כל הקודקודים היא טריוויאלית), נרצה לחפש את הקבוצה המינימלית המקיימת תנאי זה (כלומר, כיסוי קודקודים מינימלי). באופן דומה לתיאור הקודם, מציאת הקבוצה המקסימלית היא טריוויאלית (קבוצת כל הקודקודים).

נרצה להגדיר את הבעיה באופן פורמלי:

יהי גרף $G=(V,E)$ כאשר $V$ היא קבוצת הקודקודים ו-$E$ היא קבוצת הקשתות.

נגדיר קבוצה $V'$ המקיימת: $V' \subseteq V$, כלומר, תת-קבוצה מוכלת ממש בקבוצת קודקודי הגרף, כך ש:

$$\forall\ (u,v) \in E:\ u \in V' \ \vee\ v \in V'$$

זוהי הגדרת בעיית כיסוי קודקודים, אך לא מינימלית.

כדי להגדיר את הבעיה כמינימלית, צריכה להיות תת-קבוצה שמקיימת:

(1) הקבוצה $V'$ היא כיסוי קודקודים.

(2) הקבוצה $V'$ היא מינימלית.

כדי להגדיר מינימליות, נגדיר כי לא קיימת קבוצה $V''$ קטנה יותר שהיא גם כיסוי קודקודים.

לכן, נגדיר גם כי: $\forall\ |V''| < |V'|\ \exists\ (u,v) \in E: u \notin V'' \wedge\ v \notin V''$

[1] מכיוון שקבוצת כל הקודקודים בהכרח כוללת לפחות קודקוד אחד לכל קשת בגרף.

יהי קודקוד $v$, אזי הקבוצה $V \setminus \{v\}$ מכילה לפחות קודקוד אחד לכל קשת בגרף.

מכיוון שלכל קשת יש 2 קודקודים, אבל רק קודקוד אחד לא שייך לקבוצה ולכן הקודקוד השני בהכרח בקבוצה.

נשים לב כי זוהי בעיה חישובית מסוג NP-complete, כלומר, בעיית הכרעה שניתן למצוא רדוקציה פולינומית מכל בעיית הכרעה במחלקת סיבוכיות NP, אליה [לפי רדוקציה מבעיית SAT[2]].

כלומר, זוהי בעיה לא טריוויאלית לפתרון פשוט – מעבר על כל תת-קבוצה של הקודקודים, כלומר, פתרון בסיבוכיות מעריכית של מספר הקודקודים: $2^n$ תתי-קבוצות, ולכן הפתרון הפשוט ביותר שהוצג כאן (נקרא גם $brute\ force$) בהכרח ימצא את הפתרון הנכון, אבל באופן מאוד איטי ולא יעיל, במיוחד עבור גרפים גדולים מאוד, ולכן נרצה למצוא פתרונות יעילים יותר בעזרת הכלים שהכרנו בקורס.

## גישות הפתרון

### פתרון 1 – נאיבי brute force[3]

הפתרון הנאיבי שאנו מציגים, מקבל גרף כקבוצת קודקודים ומטריצת שכנויות (adjacency matrix) המייצגת את הקשתות.[4] נשים לב שאם הגרף ריק, כלומר, אין בגרף קשתות, אזי בהכרח כיסוי הקודקודים הוא ריק:

$$0 \le |V'| \le |V| = |\emptyset| = 0 \Rightarrow |V'| = 0 \Rightarrow V' = \emptyset$$

כעת, נניח כי: $\emptyset \subset V' \subset V$ ולכן, נבדוק לכל גודל אפשרי [של תת-קבוצה] בתחום: $1 \le |V'| \le |V| - 1$ ולכל גודל נבדוק האם קיימת תת-קבוצה של קודקודים באותו הגודל שהיא כיסוי קודקודים.

מכיוון ומתחילים מגודל הקבוצה הקטנה לגדולה, בהכרח תת-הקבוצה הראשונה שנמצא שהיא כיסוי קודקודים (לפי ההגדרה), היא המינימלית (הרי לא מצאנו אף קבוצה קטנה ממנה שמקיימת תנאי זה).

לכל גודל, נמצא את כל תתי- הקבוצות בגודל $k$ ולכל אחת נבדוק האם לכל קשת לפחות אחד מהקודקודים נמצא בקבוצה. נשים לב כי סיבוכיות הריצה ביחס למס' הקודקודים בגרף היא:

$$\boxed{O(|V|)_{brute\ force} = \underbrace{2^{|V|}}_{\#\ of\ sub-groups} \cdot \underbrace{|E|}_{\#\ of\ edges} = 2^{|V|} \cdot |V|^2}$$

סיבוכיות זו מאוד גדולה שגדלה מאוד מהר עבור גרפים גדולים, למשל, עבור גרף המקיים: $|V| = 40$, נקבל כי הסיבוכיות היא: $O(40)_{brute\ force} = 2^{40} \cdot 40^2 = 1{,}759{,}218{,}604{,}441{,}600$.

---

[2] בעיית SAT היא בעיית הכרעה מעולם הלוגיקה המתמטית להכרעה האם עבור פסוק מסוים C קיימת השמה בוליאנית למשתנים שבפסוק, כך שהיא ספיקה, כלומר, ערכו הכולל של הפסוק הוא אמת.

[3] Brute force (תקיפה כוחנית) הוא אלגוריתם שפועל באופן של ניסוי וטעיה של כל האפשרויות לפתרון בעיה נתונה עד למציאת הפתרון הנכון [מויקיפדיה].

[4] מטריצת שכנויות $M$ היא מטריצה המייצגת קשתות בגרף, המקיימת: $M[i,j] = \begin{cases} 0 & (v_i, v_j) \notin E \\ 1 & (v_i, v_j) \in E \end{cases}$

## פתרון 2 – תכנות לינארי (linear programming)

ראשית, נשים לב כי הבעיה שלנו היא בעיית תכנות לינארי[5], ולכן נרצה למדל את הבעיה ע״י הגדרת המשתנים, האילוצים והמטרה:

נגדיר כל קודקוד $x_i$ כמשתנה בינארי, כלומר, $x_i = 1$ אם הקודקוד שייך לכיסוי הקודקודים המינימלי, ו-0 אחרת. נשים לב כי יש להגדיר את $0 \leq x_i \leq 1$ כמשתנה **שלם** מכיוון שאי אפשר לבחור ״חצי״ קודקוד.

לכן, האילוץ הראשון שנגדיר הוא שכל המשתנים שלמים: נרצה להגדיר גם כל קשת $z_j$ כמשתנה בינארי, כלומר, $z_j = 1$ אם הקשת נמצאת בגרף (התא המתאים במטריצה הוא 1), כדי שנוכל לבדוק עבור כל קשת $(i, j)$ את האילוץ: $x_i + x_j \geq 1$ [בהכרח סכום הקודקודים גדול מ-1].

מכיוון שהמשתנים של הקודקודים בינאריים, אזי כדי שיתקיים אי-השוויון: $x_i = 1 \ \vee \ x_j = 1$ , בחירה של לפחות אחד הקודקודים (או שניהם). נשים לב כי זוהי בדיוק הגדרת הבעיה שלנו.

כעת נרצה להגדיר את המטרה (objective) שהיא: $min(\sum_{i=1}^{|V|} x_i)$ , המינימום של סכום הקודקודים בגרף – בחירת תת-הקבוצה המינימלית של הקודקודים שמקיימת את שאר האילוצים שהוגדרו.

לסיום, נרצה לוודא שכמות הקודקודים שנבחרה אכן מכסה את כל הגרף ולא משתמשת ב-״חלקי״ קודקודים, כך שעדיין עומדים בכל הכללים. כלומר, ההגדרה של המשתנים היא: $0 \leq x_i \leq 1$ שהיא: **linear programming relaxation**, כלומר, ״הקלה״ של האילוצים לצורך החישוב, ולכן נרצה לוודא שאכן מתקיים כיסוי קודקודים בכך שנבחר לאחר הפתרון, את הקודקודים המקיימים: $x_i > 0.9$, כלומר, קודקוד שקיבל ערך גבוה וקרוב ל-1.

נפתור זאת ע״י ספריית pulp ע״י הגדרת האילוצים, המשתנים וההגדרות:

```
# define the objective function
prob += pulp.lpSum(x)
# define the constraints
for (u,v) in G.edges():
   # each edge is covered
   prob += x[u] + x[v] >= 1
# solve
prob.solve()
```

[5] בעיית אופטימיזציה עם מטרה (objective), משתני החלטה (variables) ואילוצים (constraints) שהם כולם לינאריים (שוויוניים ואי-שוויוניים).

פתרון 3 – גנטי (genetic algorithm)

אלג' גנטיים מבוססים על התאוריה של דרווין, והם משמשים בעיקר בכדי למצוא קירוב לפתרון אופטימלי, עבור בעיות אופטימיזציה רבות וקשות, אשר פתרון פולינומי אינו בר ביצוע.

**לאלג' הגנטי שלנו יש 5 שלבים:**

1. Initialization– בשלב זה נגריל כמה פתרונות אפשריים ל-VC.
2. Fitness Score– הגדרנו פונקציית הערכה כפונקציה של כמות הקשתות המכוסות. כלומר אנו נרצה שלכל קשת בגרף, יש לפחות קודקוד אחד, בקבוצה שמיועדת להיות VC. ככל שכמות הקשתות שאין להן אף קודקוד ב-VC גדולה יותר, אזי הציון של הפתרון, יהיה גדול יותר (שזה לא טוב), כלומר אנו נרצה ציון כמה שיותר נמוך, כאשר 0 מציין, שיש לנו פתרון אשר לכל קשת בגרף יש לפחות קודקוד אחד בקבוצה.
3. Selection – כל הפתרונות שמעל סף מסוים, יעברו לדור הבא, ואילו נהרוג את כל שאר הפתרונות.
4. Crossover – ניקח שני פתרונות, וניצור מהם צאצא. בפועל, אנחנו לוקחים את הקודקודים שנמצאים בפתרון אחד, ולא נמצאים באחר, ובאופן דומה לוקחים את הקודקודים שנמצאים בפתרון השני ושלא נמצאים בפתרון הראשון, ומשתי הקבוצות האלה מחליפים חצי מהקודקודים, כלומר מכניסים אל הפתרון הראשון, 50 אחוז מהקודקודים שהיו בפתרון השני ולא היו אצלנו במקור.
5. Mutation – ניקח את הפתרון החדש שיצרנו מקודם, ונבצע לו מוטציה (שמשנה 5 אחוז מהפתרון). יש לנו 2 סוגי מוטציות אפשריות (הסיכוי לכל אחת מהמוטציות זהה):

   סוג ראשון - לוקח 5 אחוז מהקודקודים של הפתרון, ומחליף אותם ב-5 אחוז של **קודקודים אחרים שלא היו בפתרון הזה לפני.**

   סוג שני - לוקח 5 אחוז מהקודקודים של הפתרון, ומחליף אותם ב-5 אחוז של **קודקודים, שיכולים לכסות קשתות שהפתרון הנוכחי לא מצליח לכסות.**

נק' חשובות:

- אנו מגדירים גודל התחלתי של אוכלוסייה וגודל מקסימלי של אוכלוסייה שאנחנו מסכימים לה, וכן נגדיר גם מספר איטרציות מקסימלי. נעצור את האלג' כאשר יגיע אל גודל האוכלוסייה המקסימלי שהגדרנו או עד שיגיע למס' האיטרציות המקסימלי שהגדרנו.

- כמו כן חשוב לציין, כי האלג' הגנטי מבצע חיפוש בינארי בהתחלה לגודל ה-VC, שכן האלג' צריך ליצור פתרונות בגודל מסוים – כלומר ליצור מועמדים פוטנציאלים ל-VC, אך הוא אינו יודע מה גודל ה-VC המינימלי, ולכן נבצע חיפוש בינארי על הגודל ה-VC, ועבור כל ערך, נריץ את האלג' הגנטי, ולבסוף ניקח את ה-VC הכי קטן, בדיוק לפי ההגדרת הבעיה שהצגנו בהתחלה.

## ניתוח תוצאות

### ניתוח תוצאות – השוואת זמן ריצה

בגרף זה מוצג זמן הריצה בשניות כפונקציה של מספר הקודקודים בגרף בתחום $12 \leq |V| \leq 24$ .

בגרף ניתן לראות כי עבור מס׳ קודקודים קטן (למשל 12), זמני ריצת האלגוריתמים השונים די דומים. ככל שמגדילים את מס׳ הקודקודים, זמן הריצה עולה בקצב שונה לגמרי:

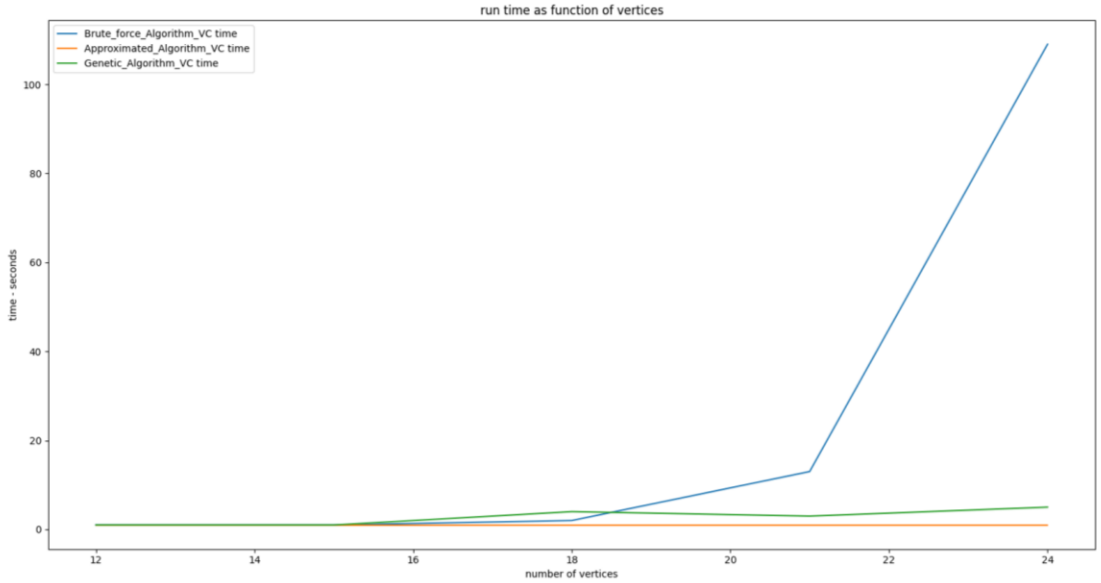
בשיטת ה-brute force, זמן הריצה נראה כגרף מעריכי החל מ: $|V| = 18$ כך שעבור 24 קודקודים, זמן הריצה יהיה מעל 100 שניות (לעומת פחות מ-20 שניות ריצה עבור גרף עם 21 קודקודים).

בשיטת התכנות הלינארי, אנחנו משתמשים בספרייה ה-python, pulp (שפותרת את הבעיה בשיטת branch and bound), זמן הריצה די קבוע - שנייה אחת לכל גדלי הגרף בתחום זה.

בשיטה הגנטית, ניתן לראות עליות וירידות בגלל שתהליכי ה-*crossover* וה-*mutation* שלה הסתברותיים (אם כי המגמה הכללית היא עליה איטית), אך ניתן לראות בבירור כי זמן הריצה קצר בהרבה משל השיטה הנאיבית (החל מ-18 קודקודים), אך ארוך יותר עבור שיטת התכנות הלינארי. עם זאת, עבור גרפים "קטנים" עם פחות מ-18 קודקודים, זמן ריצת האלגוריתם הגנטי הוא הארוך ביותר.

הערות:

- עבור גרפים מאוד קטנים (מס׳ חד ספרתי של קודקודים), כל השיטות פותרות את הבעיה בזמן קצר (ואפילו הגישה הנאיבית היא המהירה ביותר), ולכן לא ניתן להסיק מהם תובנות על ההבדלים בביצועים.
- עבור גרפים עם יותר מ-24 קודקודים, זמן הריצה של השיטה הנאיבית גבוה מאוד, ועקב מגבלות כוח החישוב, לא הצגנו את תוצאותיו.



ניתוח תוצאות – השוואת איכות הפתרון

כעת, בכל שלב יצרנו גרף רנדומלי, עם מספר קודקודים מסוים, ובדקנו את ה-VC שכל אחד מהאלג' שלנו החזיר. נשים לב, כי האלגוריתם הנאיבי תמיד יחזיר את ה-vertex cover המינימלי (שהוגדר בתחילת הקובץ), ולכן נשווה את אלג' התכנות הלינארי, ואת האלג' הגנטי אליו.

עבור גרפים קטנים יחסית:

| מס' הקודקודים בגרף | נאיבי | תכנות לינארי | אלגוריתם גנטי |
|---|---|---|---|
| 3 | 1 | 1 | 1 |
| 5 | 2 | 2 | 2 |
| 7 | 4 | 4 | 5 |
| 9 | 6 | 6 | 6 |
| 11 | 8 | 8 | 8 |
| 13 | 9 | 9 | 9 |
| 15 | 10 | 10 | 11 |

עבור גרפים גדולים יחסית:

| מס' הקודקודים בגרף | נאיבי | תכנות לינארי | אלגוריתם גנטי |
|---|---|---|---|
| 20 | 14 | 14 | 16 |
| 21 | 15 | 15 | 17 |
| 22 | 17 | 17 | 18 |
| 23 | 18 | 18 | 19 |
| 24 | 19 | 19 | 20 |
| 25 | 20 | 20 | 21 |
| 26 | 21 | 21 | 23 |

ניתן לראות בתוצאות שלנו, כי אלגוריתם התכנות הלינארי תמיד החזיר את כיסוי הקודקודים המינימלי, בעוד שהאלגוריתם הגנטי החזיר כיסוי קודקודים שאינו מינימלי (במרבית הגרפים), אך לא גדול משמעותית מהכיסוי המינימלי, כלומר, האלגוריתם הגנטי שלנו הינו אלג' המעניק קירוב טוב לפתרון.

הסיבות לאי-הדיוק של האלגוריתם הגנטי (ביחס לאלגוריתם הנאיבי):

- האלגוריתם הגנטי משתמש ב-crossover ו-mutation, שאופן מימושם הוא תהליכים סטוכסטיים (הסתברותיים), ולכן ישנה הסתברות לטעות בניגוד לאלגוריתם הנאיבי (שהוא אלגוריתם דטרמיניסטי).
- חסימת גודל האוכלוסיה המקסימלית שאנחנו מסכימים לה, וכן חסם על מס' האיטרציות המקסימלי. למשל, אם נגיע לגודל האוכלוסיה המקסימלית לפני שנגיע למס' האיטרציות המקסימלי, אזי נעצור. (וכן להפך).

חשוב לציין כי אלגוריתם התכנות הלינארי גם הוא אלגוריתם מקורב (על אף שמחזיר תוצאות מדויקות במקרים שהוצגו). הדיוק הרב נובע מכך שזהו קירוב טוב, וכן כי אנו משתמשים בספריית pulp שמשתמשת באופטימיזציות מתקדמות רבות (למשל, branch and cut).

## מסקנות ותובנות אישיות

### מסקנות

ניתן להבחין כי, עבור גרפים קטנים יחסית, אנו עשויים להעדיף את האלג' הנאיבי, שכן הוא עשוי להיות מהיר יותר מהאלג' הגנטי, ואלג' התכנות הלינארי. ניתן אף לבצע שיפור קל לאלג' הנאיבי, בכך שאנו לא נבחן כל תת-קבוצה של קודקודים, שכן אין חשיבות לסידור הפנימי בתוך הקבוצה, כלומר אין הבדל בין תת-הקבוצה $v_1, v_2, v_3$ לבין תת-הקבוצה $v_3, v_1, v_3$, ולכן עבור גרפים קטנים, שיפור זה יכול לסייע בהורדת זמן הריצה.

לעומת זאת, עבור גרפים קטנים ראינו כי תוצאות זמן הריצה של האלג' הגנטי היו הכי פחות טובות מבין כל האלג' שבדקנו. זאת משום שבאלג' הגנטי אנו מבצעים חיפוש בינארי עבור גודל ה-VCהרצוי, ועבור כל גודל אנו נריץ את האלג' מחדש, שכולל סדרת שלבים ארוכה, וביצוע מס' מוטציות. כאשר הגרף קטן יחסית, תהליך זה עשוי להיות מהיר יותר, לסקור את כל האפשרויות, ולא לנסות להגריל פתרונות ולשפר אותם.

כמו כן, ככל שמורכבות הבעיה גדלה, כלומר הגרף גדל, בלתי סביר להריץ את האלג' הנאיבי, וכפי שראינו בחלק של ניתוח התוצאות, הקפיצות בזמן הריצה שלו גבוהות במיוחד. מבדיקה שעשינו, עבור 25 קודקודים האלג' הנאיבי רץ קרוב ל-4 דקות, ואילו עבור 26 קודקודים קיבלנו כי זמן הריצה הינו 7-8 דק', וזה רק עבור הגדלה של קודקוד בודד, ואילו זמני הריצה של האלג' הגנטיים ואלג' התכנות הלינארי נמדדו בכמה שניות בודדות. חשוב להדגיש, כי האלג' הגנטי מכיל בתוכו תהליכים סטוכטיים, כמו crossover, mutation, ואלו עשויים להשפיע על הפתרון המקורב.

לסיכום, ניתן להסיק כי אלגוריתמים גנטיים הינם אלגוריתמים מקורבים, המתכנסים במהירות, ומסקנה דומה עבור אלג' התכנות הלינארי שלנו. למעשה, ראינו כי ניתן לפתור בעיות NP קשות בעזרת שיטות אופטימיזציה שראינו בקורס, למשל אלגוריתמים גנטיים (When Suboptimality is Essential) או שיטות של תכנות לינארי (branch and bound). זוהי מסקנה חשובה, שכן כפי שראינו בחלק של ניתוח התוצאות, לעיתים ההבדלים באיכות הפתרון הינם זניחים יחסית, ואילו הבדלי הריצה משמעותיים. כלומר במגוון רחב של מצבים, אנו בהחלט עשויים להעדיף פתרון מקורב, כלומר כזה שאינו צודק במאה אחוז, אך כזה שניתן לקבלו במהרה, ולכן יותר ריאלי להשתמש באלג' הזה במשימות בחיי היום-יום. שכן לעיתים אין לנו את הזמן, או את היכולת, לחכות פרק זמן משמעותי בכדי לקבל פתרון נכון, ונרצה לקבל פתרון מהיר ומקורב, בסיכוי גבוה.

### תובנות אישיות

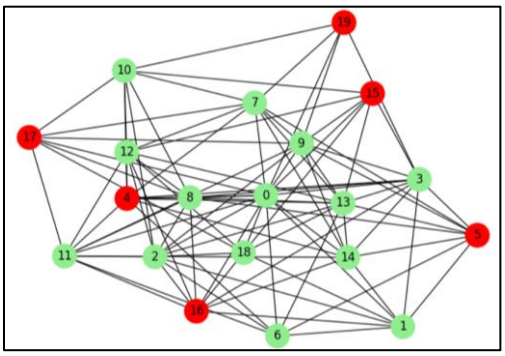
למדנו בפרויקט איך לממש דוגמאות של שיטות שונות שנלמדו בהרצאה ולפתור בעזרתן בעיות שונות ולהשתמש באלגוריתמים השונים שראינו בקורס ובשיטות השונות לפתרון בעיות לא טריוויאליות. למדנו איך להשוות בין האלגוריתמים השונים ולהעריך את ביצועיהם, ולהכריע באיזה מקרה להשתמש בכל אחד מהם, ומה הסיבות להבדלים ביניהם, מכיוון שלא תמיד יש את הזמן והיכולת להמתין לסוף ריצת אלגוריתם מסוים.

למדנו במהלך העבודה על פרויקט זה ידע מקצועי ושיטות עבודה שנוכל להשתמש בהן בעתיד בפרויקטים שונים.

**<u>נספח 1 - דוגמאות הרצה (גרפים שהוגרלו באופן רנדומי) כאשר ה-VC הם הקודקודים הירוקים</u>**

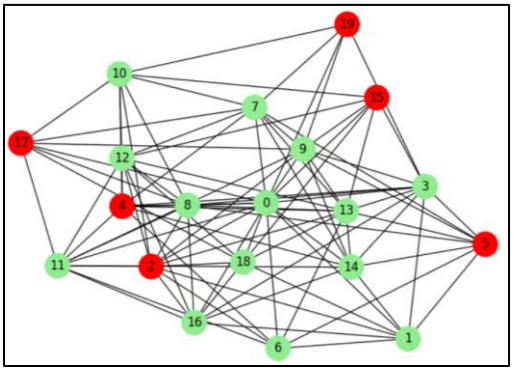
<u>אלגוריתם נאיבי:</u>

דוג' לפתרון בעזרת brute force עבור גרף של 20 קודקודים [מופיע ב- *Brute_force_Algorithm_VC*]:




<u>אלגוריתם תכנות לינארי:</u>

דוג' לפתרון בעזרת תכנות לינארי עבור גרף של 20 קודקודים [מופיע ב- *Approximated_Algorithm_VC*]:




<u>אלגוריתם גנטי:</u>

דוג' לפתרון בעזרת אלגוריתם גנטי עבור גרף של 20 קודקודים [מופיע ב- *Genetic_Algorithm_VC*]: