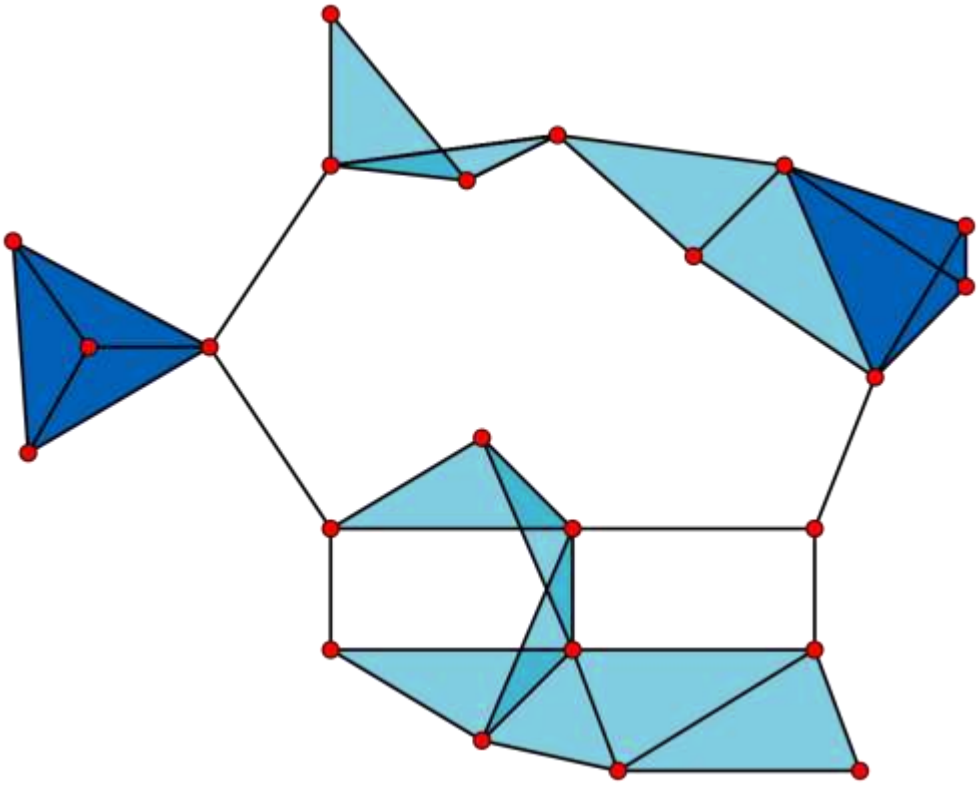
מיני פרויקט אלגוריתמים אבולוציוניים



קורס: נושאים ביישומים של מדעי המחשב
מרצה: פרופסור משה זיפר
מגישים: אלה בארי וגיא טורביץ
תאריך הגשה: 23.01.23

תוכן עניינים

מבוא

בעיות ידועות במדעי המחשב-

במדעי המחשב, ניתן לסווג בעיות שונים על פי פרמטרים רבים. החל מעולם התוכן של הבעיה, מידת חשיבותה, ועד זמני הריצה וכמות הזיכרון הנדרש לפתרון הבעיה.

אחד הפרמטרים החשובים ביותר הוא זמן הריצה. בתוך זה, קיים סיווג נוסף שמחלק את כל הבעיות הקיימות (שניתן למצוא להן פתרון) לשתי קבוצות:

בעיות P, ובעיות NP.

בעיה ששייכת לקבוצה P, היא בעיה שקיים וידוע לה אלגוריתם שפותר אותה בזמן ריצה פולינומי לכל היותר.

בעיה NP קשה היא בעיה שלא ידוע אלגוריתם לפתרונה בזמן פולינומי, אבל בהינתן 'עד' (פתרון כלשהו), ניתן לוודא את נכונותו בזמן קצר ביותר.

קליקות-

תורת הגרפים היא תחום מרכזי וחשוב בעולם המחשבים. בעיות גרפים רבות מייצגות בעיות אמיתיות מהחיים, ופתרונן יכול לעזור בתחומים רבים ושונים (וויז, עיבוד רשתות נוירונים ועוד).

בתוך זאת נתבונן בקליקה: קליקה היא קבוצת קודקודים בגרף בלתי מכוון, אשר כל זוג קודקודים שונים בה מחובר על ידי קשת. כלומר, תת-הגרף המושרה על ידה מהווה גרף שלם.

קליקת מקסימום בגרף בלתי מכוון היא קליקה, כך שלא קיימת בגרף קליקה בעלת מספר קודקודים גדול יותר.

אלגוריתמים אבולוציוניים-

ראינו כי קיימות בעיות שטרם נמצא להן פתרון מהיר, זאת אומרת שעד כה נמצאו להן אלגוריתמים בזמן מעריכי בלבד.

מאחר שחלק ניכר מבעיות אלו הן בעיות חשובות אשר קיים ערך גדול לפתרונן, נולד הביקוש למצוא סוג פתרונות אחרים (ולא בהכרח דטרמיניסטים),

על מנת להוריד משמעותית את זמן ריצת האלגוריתם.

אחד העזרים שנמצא יעיל לכך הוא אלגוריתמים אבולוציוניים.

העיקרון של אלגוריתם זה עובד לפי העיקרון האבולוציוני - "המתאים שורד". בכל דור, רק מי שחזק דיו יכול להתרבות, ואל הדור הבא נלקחים אלמנטים משני ההורים.

בנוסף, כל זן צריך להתמודד עם התאמה לסביבה והשינויים שחלים בה.

בעצם נבצע "ברירה טבעית" באופן מלאכותי, בשביל לבחור את המועמדים שיעברו לשלבים הבאים.

רעיון תכנותי בסיסי זה מושפע מההצלחה של האבולוציה בפתרון בעיות אמיתיות.

לכן אם ניקח אוכלוסייה של פתרונות ונבחר מתוכם רק את המתאימים ביותר לפתרון בעיה, נערבב אותם אחד עם השני ונוסיף קצת רעש,

נקבל דור חדש של פתרונות הקרוב צעד נוסף לפתרון הבעיה הנתונה.

נחזור על התהליך מספר רב של פעמים (דורות) ולבסוף נגיע לפתרון הקרוב ביותר.

נוכל להיעזר באלגוריתמים אבולוציוניים כדי לפתור בעיות אלו בזמן סביר,ומהיר יותר מהפתרון הדטרמיניסטי שלהן.

הגדרת הבעיה

הבעיה שנתמקד בה בפרוייקט זה היא הבעיה של מציאת קליקת מקסימום בגרף בלתי מכוון נתון.

קיימות בעיות רבות שניתן לייצג באמצעות בעיה זו נפרט על חלקן:

ניתוח רשתות חברתיות- ברשתות חברתיות, קליקות יכולות לייצג קבוצות של אנשים הקרובים מאוד אחד לשני.

זיהוי הקשרים הללו יכול לעזור לחוקרים לזהות קהילות מסויימות או מבנים חברתיים ולהבין כיצד המידע מתפשט בהם דרך הרשת החברתית.

רשתות ביולוגיות- ניתן להשתמש בקליקות כדי לזהות קבוצות של חלבונים או קבוצות של גנים שעשויים למלא תפקידים דומים ברשת ביולוגית.

התאמת פרסומות- ניתן להשתמש בקליקות לזיהוי קבוצות של פריטים הנרכשים לעיתים קרובות יחד, ובאמצעותן ניתן לקבל המלצות מותאמות אישית ללקוחות, כלומר מה לפרסם ולמי.

איתור הונאה- ברשתות פיננסיות, קליקות יכולות לשמש כדי לזהות קבוצות של יחידים המשתתפים בפעילויות הונאה.

בעיה זו היא בעיה NP-שלמה, ולכן לא ידוע אלגוריתם הפותר את הבעיה בזמן פולינומי. מאחר וראינו כי קיימים שימושים רבים במגוון תחומים לבעית הקליקות, פתרון יעיל לבעיה הוא בעל חשיבות.

נרצה למצוא דרך טובה יותר, כלומר מהירה יותר, לעשות זאת ולשם כך ניעזר באלגוריתם אבולוציוני.

הגדרת הפתרון

פתרון הבעיה הוא וקטור שמייצג קליקה. נקבל וקטור ביטים, במילים אחרות, וקטור בו הקורדינטות הן 0 או 1, כך שכל קורדינטה מייצגת קודקוד בגרף.

$$vector[i] \in \{0,1\}$$

נצפה כי בין כל שניים מהקודקודים שהקורדינטה שלהן היא 1, תהיה קיימת קשת. בנוסף, נצפה כי קליקה זו תהיה הקליקה הגדולה בגרף.

קודקוד אשר הקורדינטה המייצגת שלו היא 0 הוא קודקוד שלא שייך לקליקה, ואילו אם הקורדינטה שווה ל 1, הקודקוד כן שייך לקליקה.

קיימים פרמטרים שונים שניתן לקבוע, והם משפיעים על מהירות ודיוק האלגוריתם. ניסינו מגוון אפשרויות ובחנו איזה הרכב של נתונים רצים בצורה הטובה ביותר. כמובן שיש תלות בגרף הקלט (מספר קודקודים ומספר קשתות). מצאנו עניין בבחירת הפרמטרים, והחלטנו לבנות ממשק בו בכל ריצה נוכל לשנות את פרמטרים אלו ולבחון את תפקודו של האלגוריתם האבולוציוני. בנוסף, הגדרנו פרמטרים כברירת מחדל שאותם מצאנו יעילים. בעצם, בכל ריצה של האלגוריתם נוכל לשנות את ההגדרות בהתאם לרצוננו, ולבדוק איך הפרמטרים השונים משפיעים על זמן הריצה, מספר הדורות הנדרשים לפתרון, וציון הfitness הסופי.

פרמטרים אלו:

- מספר קודקודי הגרף
- הסתברות לקשת כלשהי להיווצר
- גודל האוכלוסייה
- אליטיזם
- הסתברות למוטציה באינדיבידואל
- בעת מוטציה, כמות הביטים המוחלפים
- גודל הטורניר
- הסתברות לטורניר
- שיטת הקרוס-אובר
- מס' דורות מקסימלי
- מס' דורות ללא שינוי (עד שנעצרת התוכנית)

<u>גרף הקלט:</u> נוכל לבחור בכל ריצה את מספר קודקודי הגרף – בין 5 ל 200. ואת הסיכוי של קשת כלשהי להיווצר.

<u>האוכלוסייה:</u> האוכלוסייה שלנו היא קליקות. כל קליקה מיוצגת על ידי וקטור ביטים אקראיים. ההסתברות של ביט מסוים לקבל את הערך 0 או לקבל את ערך 1 היא 0.5.

כל אינדקס בוקטור מייצג קודקוד, וערכו מצביע על השתייכותו או אי השתייכותו לקליקה. (0=שייך. 1=לא שייך.).
בנוסף, נקבע בתחילת הריצה את גודל האוכלוסייה.

תהליך האבולוציה:

חישוב הפיטנס: נבחן את האוכלוסייה שלנו. כל אינדיבידואל יעבור אוולואציה, כלומר, הקליקה תיבחן לפי מידת הנכונות שלה. מה מספר הקודקודים בקליקה הגדולה ביותר המיוצגת ע"י האינדיבידואל? מה היחס בין הקליקה הגדולה ביותר באינדיבידואל לבין כמות האחדות המופיעות בו? לפי פרמטרים אלה כל אינדיבידואל יקבל ציון fitness.
השתמשנו גם בשיטת "Caching" בכדי לא להעריך אינדיבידואל מסויים יותר מפעם אחת, ובכך לחסוך בזמן ריצה מיותר.

אליטיזם: בתחילת הריצה קבענו elitism_rate, נסמנו e. ניקח את e% האינדיבידואלים ה"מוצלחים ביותר" – אלו שקיבלו את הfitness הגבוה ביותר, ואותם נשלח אוטומטית לדור הבא.

טורניר: נחלק את האינדיבידואלים לקבוצות טורניר לפי הקלט של גודל הטורניר. מכל טורניר ניקח את האינדיבידואל ה"מוצלח" ביותר. כלומר, האחד שקיבל את הfitness הגבוה ביותר.

קרוס-אובר: מבין האינדיבידואלים שניצחו בטורניר: כל פעם ניקח שניים, ונבצע עליהם קרוס אובר לפי אחת מהשיטות שנבחרו-

one point - בוחרת אינדקס בין 0 לn. האינדיבידואלים יחליפו בניהם את האינדקסים 0 עד i.

two point - שיטה זאת בוחרת שני אינקסים באופן אקראי, ומחליפה את ערכי האינדקסים בין שני האינדיבידואלים.

uniform – לכל i: 0 =< i =<n בהסתברות 0.5 מגרילה באילו אינדקסים להחליף בין הערכים של שני האינדיבידואלים.

מוטציה: נעבור על כל האינדיבידואלים שאנו מחזיקים, לפי שיעור המוטציה שנקבע נגריל באילו אינדיבידואלים לבצע שינוי, ועבור כל אינדיבידואל שנבצע בו מוטציה נגריל אינדקסים לשינוי, לפי כמות הביטים להחלפה שקבענו.

תהליך זה הוא יצירת הדור הבא. לפי קביעתנו נבחר את מספר הדורות הרצוי. נבחין כי יש לצפות שבגרפים גדולים יותר, יהיה קשה יותר להגיע לפתרון מדויק, ועל כן נצטרך באופן יחסי מספר דורות גדול יותר. כאשר

נבחן גרפים קטנים יותר, יהיה קל יותר להגיע לפתרון ועל כן בשביל לצמצם זמן ריצה מיותר נבחר מספר קטן יותר של דורות.

ניתן לבצע "ניסוי וטעיה" כדי לבחון את הפרמטרים השונים, בעיקר לפי גודל הגרף וכמות הקשתות בו, וכך לבצע הרצות מהירות ואפקטיביות יותר בפעמים הבאות.

מבט על של התוכנה

נפרט על המחלקות והפונקציות השונות:

-CliqueCrossover.py
מחלקה זו אחראית על ביצוע הקרוס-אובר לפי השיטה שנבחרה. שלושת המטודות השונות פורטו למעלה.

-CliqueEvaluator.py
מחלקה זו מקבלת אינדיבידואל ומחשבת לו ציון פיטנס.

-CliqueMutation.py
מחלקה זו מקבלת איניבידואל ומבצעת בו מוטציה.

-EvolutionaryAlgorithm.py
המחלקה בה נמצא האלגוריתם SimpleEvolution, שאחראי על הרצת האבולוציה.

-Graph.py
מחלקת הגרפים. מאתחלת את מטריצת השכנויות של גרף הקלט.

-GraphVisualization.py
הויזואליזציה של הפתרון. שמנו דגש שנוכל לראות בפתרון באופן מודגש את הקודקודים השייכים לקליקה ואת הצלעות המחברות בניהם.

-RandomGraph.py
יצירת גרף הקלט. לפי מספר הקודקודים וההסתברות של קשת להיווצר- פרמטרים שנקבעים בתחילת הריצה, ניצור אקראית את הגרף שלנו. דיפולטיבית נעבוד עם גרף בעל 30 קודקודים בו הסיכוי של צלע כלשהי להיווצר הוא 0.7.

-TerminationCheckerChange.py
בחלק מההרצות, נראה שמדור לדור אין שיפור בפיטנס. במקרה כזה לא נרצה שהאלגוריתם ימשיך לרוץ לפי מספר הדורות שנקבע, אלא נרצה לסיים את הריצה מתוך הבנה שכבר קיבלנו את התוצאה הטובה ביותר ואין צורך לבזבז זמן ריצה מיותר. נבדוק אם הציון המקסימלי בדור הנוכחי הוא שיפור לעומת הדור הקודם, ואם הוא לא- בחלוף מס' דורות מסויים הנקבע מראש, נסיים את הריצה ונחזיר את האינדיבידואל הטוב ביותר (כלומר, בעל ערך הפיטנס הגבוה ביותר) הנוכחי.

-UI.py

הממשק של בחירת הפרמטרים. לפי ההסבר למעלה, בנינו ממשק לבחירת פרמטרי-ריצה שונים אותם נוכל להגדיר בכל ריצה מחדש. שם מוגדרים גם ערכי ברירת המחדל.

תוצאות הדגמות ריצה וגרפים

לאחר שסיימנו לכתוב את כל מחלקות האלגוריתם, ביצענו כמות גדולה של הרצות ובדיקות, עם פרמטרים שונים בכל פעם, בשביל לנסות להבין את יכולות האלגוריתם. בכל סדרת הרצות כזו ניסינו להבין מה הם הערכים האופטימליים אותם צריך לקבל האלגוריתם, על מנת שיוכל לפתור בצורה יעילה גרפים שונים.
תחילה גילינו באופן מובהק, אחרי ריצות מרובות על גרפים שונים ובפרמטרים שונים, ששיטת ה-"Uniform" הינה שיטת הקרוס-אובר היעילה ביותר מבין השיטות שמימשנו. אלו שתי ריצות לדוגמה, על גרפים בעלי 30 ו-50 קודקודים, כשעבור כל קונפיגורציה נלקח זמן הריצה הממוצע ומס' הריצות שהגיעו לתוצאה הנכונה-

```
Running Evolutionary Algorithm
Graph Size: 30
Graph Edge Probability: 0.5
Population Size: 120
Elitism Rate: 0.05
Mutation Probability: 0.1
Mutation Bit Flips: 1
Tournament Size: 4
Tournament Probability: 1
Crossover Type: Configuration 1 - uniform
                Configuration 2 - one_point
                Configuration 3 - two_point
Max Generation: 50
Number of Generations Unchanged: 15
Configuration 1- Average Runtime:  0.778961980342865  Successful Runs:  20 /20

Configuration 2- Average Runtime:  1.3710962653160095  Successful Runs:  20 /20

Configuration 3- Average Runtime:  1.4507500529289246  Successful Runs:  20 /20
```

```
Running Evolutionary Algorithm
Graph Size: 50
Graph Edge Probability: 0.75
Population Size: 300
Elitism Rate: 0.05
Mutation Probability: 0.1
Mutation Bit Flips: 1
Tournament Size: 4
Tournament Probability: 0.8
Crossover Type: Configuration 1 - uniform
                Configuration 2 - one_point
                Configuration 3 - two_point
Max Generation: 250
Number of Generations Unchanged: 35
Configuration 1- Average Runtime:  32.5650338768959  Successful Runs:  19 /20

Configuration 2- Average Runtime:  76.41654280424117  Successful Runs:  12 /20

Configuration 3- Average Runtime:  60.485918807983396  Successful Runs:  14 /20
```

ניתן לראות כי בשתי הריצות, בעלות פרטמרטים שונים, שיטת ה-"Uniform" הובילה לפתרון מהיר יותר משמעותית משאר השיטות, ובריצה על גרפים גדולים יותר גם לאחוז הצלחה גבוה יותר. יתר הבדיקות השתמשו גם הן בשיטת ה-"Uniform".

דוגמה נוספת לפרמטר שבחנו את השפעתו על יעילות האלגוריתם הוא אחוז המוטציה. כמו שצפינו, ככל שהגרף נהיה יותר גדול – בעל יותר קודקודים ויותר קשתות, גם אחוז המוטציה האופטימלי עולה. למטה ניתן לראות שלוש ריצות לדוגמה של האלגוריתם על גרפים בגדלים שונים, כאשר בשני הראשונים ההסתברות האופטמלית למוטציה היא 0.05, אך בגרף גדול בעל 90 קודקודים כבר עדיף 0.1-

```
Graph Size: 30
Graph Edge Probability: 0.5
Population Size: 120
Elitism Rate: 0.1
Mutation Probability: Configuration 1 - 0.05
                      Configuration 2 - 0.1
                      Configuration 3 - 0.15
Mutation Bit Flips: 1
Tournament Size: 4
Tournament Probability: 1
Crossover Type: uniform
Max Generation: 50
Number of Generations Unchanged: 15
Configuration 1- Average Runtime:  0.6111007118225098  Successful Runs:  50 /50

Configuration 2- Average Runtime:  0.9653213930130005  Successful Runs:  50 /50

Configuration 3- Average Runtime:  0.9211485624313355  Successful Runs:  50 /50
```

```
Graph Size: 60
Graph Edge Probability: 0.7
Population Size: 200
Elitism Rate: 0.1
Mutation Probability: Configuration 1 - 0.05
                      Configuration 2 - 0.1
                      Configuration 3 - 0.15
Mutation Bit Flips: 2
Tournament Size: 5
Tournament Probability: 0.8
Crossover Type: uniform
Max Generation: 150
Number of Generations Unchanged: 20
Configuration 1- Average Runtime:  16.85940951704979  Successful Runs:  40 /40

Configuration 2- Average Runtime:  19.91557698249817  Successful Runs:  40 /40

Configuration 3- Average Runtime:  17.375878804922102  Successful Runs:  39 /40
```

```
Graph Size: 90
Graph Edge Probability: 0.6
Population Size: 400
Elitism Rate: 0.1
Mutation Probability: Configuration 1 - 0.02
                      Configuration 2 - 0.05
                      Configuration 3 - 0.1
                      Configuration 4 - 0.15
Mutation Bit Flips: 3
Tournament Size: 8
Tournament Probability: 0.8
Crossover Type: uniform
Max Generation: 200
Number of Generations Unchanged: 35
Configuration 1- Average Runtime:  99.54627933502198  Successful Runs:  1 /30

Configuration 2- Average Runtime:  82.461554479599  Successful Runs:  13 /30

Configuration 3- Average Runtime:  77.8725649913152  Successful Runs:  30 /30

Configuration 4- Average Runtime:  75.78229561646779  Successful Runs:  23 /30
```

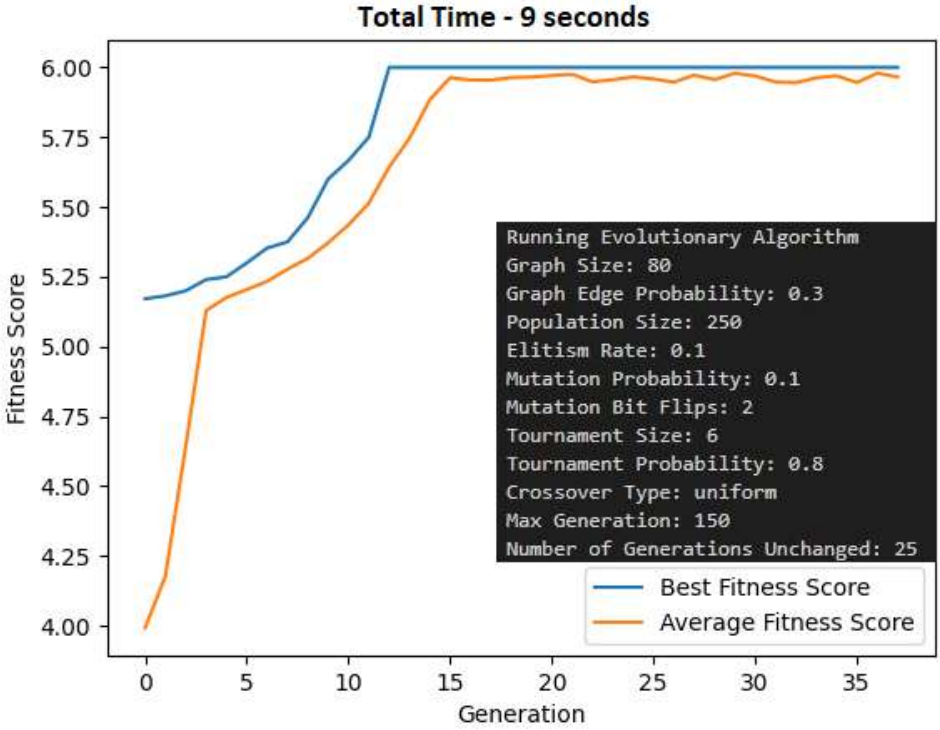
בדומה להסתברות למוטציה, גם הכמות האופטימלית של ביטים המוחלפים בעת מוטציה גדלה ככל שהגרף גדל- בגרף של 30 מספיק להחליף ביט אחד בעת מוטציה, בגרף של 60 קודקודים 2 ביטים, ובגרף של 90 3 ביטים, כמו שניתן לראות בדוגמאות הנ"ל-

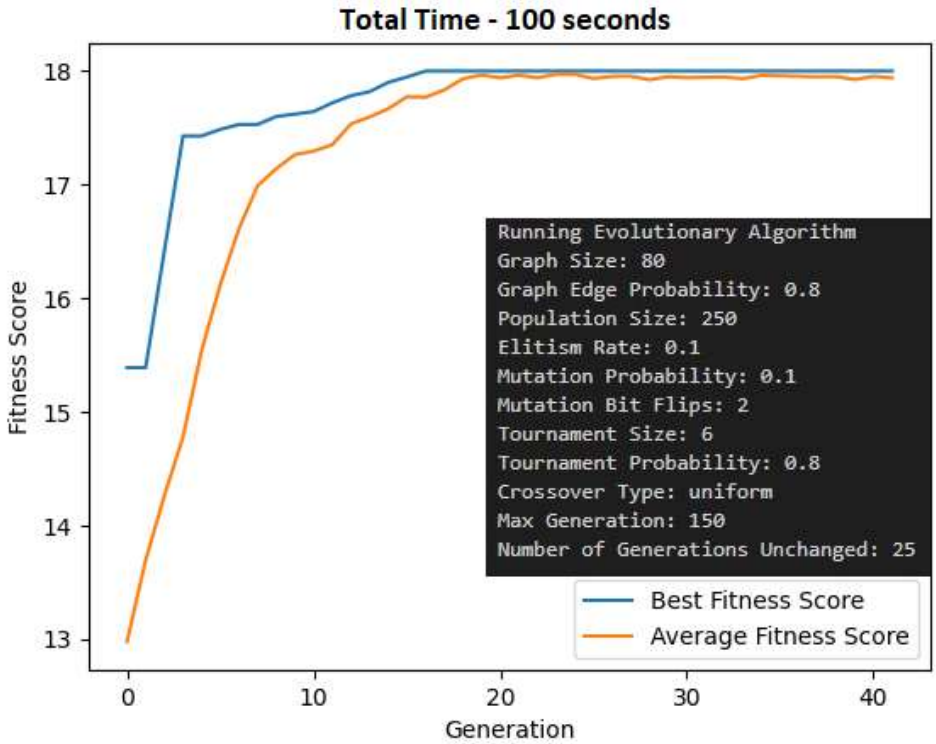
```
Graph Size: 30
Graph Edge Probability: 0.7
Population Size: 120
Elitism Rate: 0.15
Mutation Probability: 0.1
Mutation Bit Flips: Configuration 1 - 1
                    Configuration 2 - 2
                    Configuration 3 - 3
Tournament Size: 4
Tournament Probability: 1
Crossover Type: uniform
Max Generation: 50
Number of Generations Unchanged: 15
Configuration 1- Average Runtime:  1.7588358592987061  Successful Runs:  50 /50

Configuration 2- Average Runtime:  1.8968815422058105  Successful Runs:  50 /50

Configuration 3- Average Runtime:  1.9876499223709105  Successful Runs:  50 /50
```

```
Graph Size: 60
Graph Edge Probability: 0.7
Population Size: 200
Elitism Rate: 0.15
Mutation Probability: 0.1
Mutation Bit Flips: Configuration 1 - 1
                    Configuration 2 - 2
                    Configuration 3 - 3
                    Configuration 4 - 4
Tournament Size: 4
Tournament Probability: 0.8
Crossover Type: uniform
Max Generation: 160
Number of Generations Unchanged: 25
Configuration 1- Average Runtime:  20.790386724472047  Successful Runs:  38 /50

Configuration 2- Average Runtime:  19.400540890693666  Successful Runs:  50 /50

Configuration 3- Average Runtime:  21.933796801567077  Successful Runs:  50 /50

Configuration 4- Average Runtime:  26.537551350593567  Successful Runs:  50 /50
```

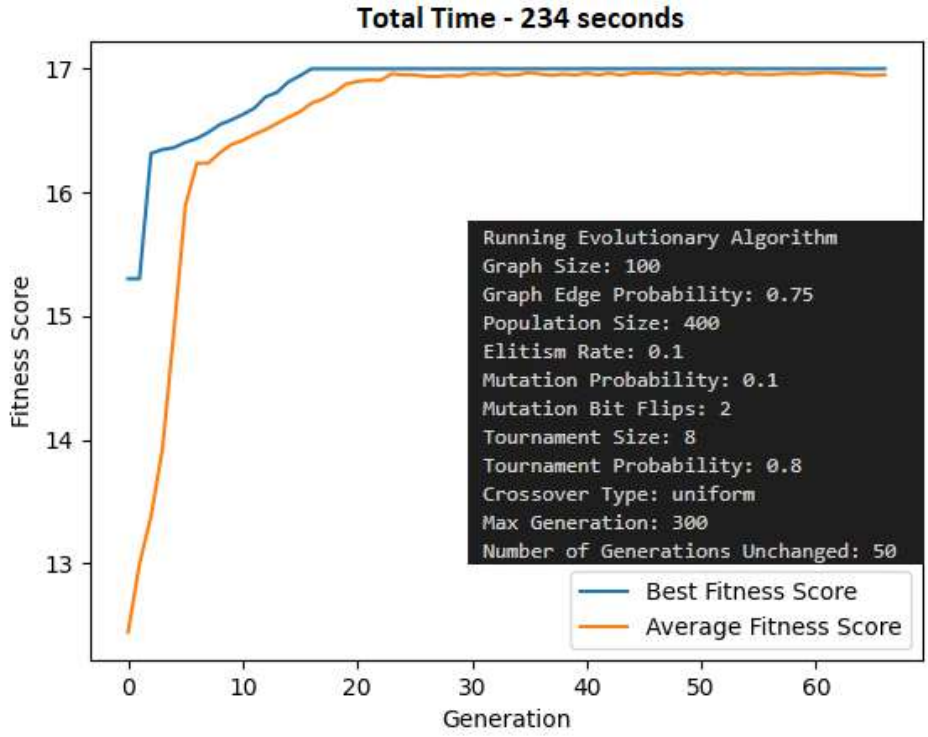
```
Graph Size: 100
Graph Edge Probability: 0.7
Population Size: 400
Elitism Rate: 0.1
Mutation Probability: 0.1
Mutation Bit Flips: Configuration 1 - 2
                    Configuration 2 - 3
                    Configuration 3 - 4
Tournament Size: 8
Tournament Probability: 0.8
Crossover Type: uniform
Max Generation: 200
Number of Generations Unchanged: 40
Configuration 1- Average Runtime:  174.34302391529084  Successful Runs:  31 /50

Configuration 2- Average Runtime:  160.22531460285188  Successful Runs:  49 /50

Configuration 3- Average Runtime:  163.6335218000412  Successful Runs:  2 /50
```

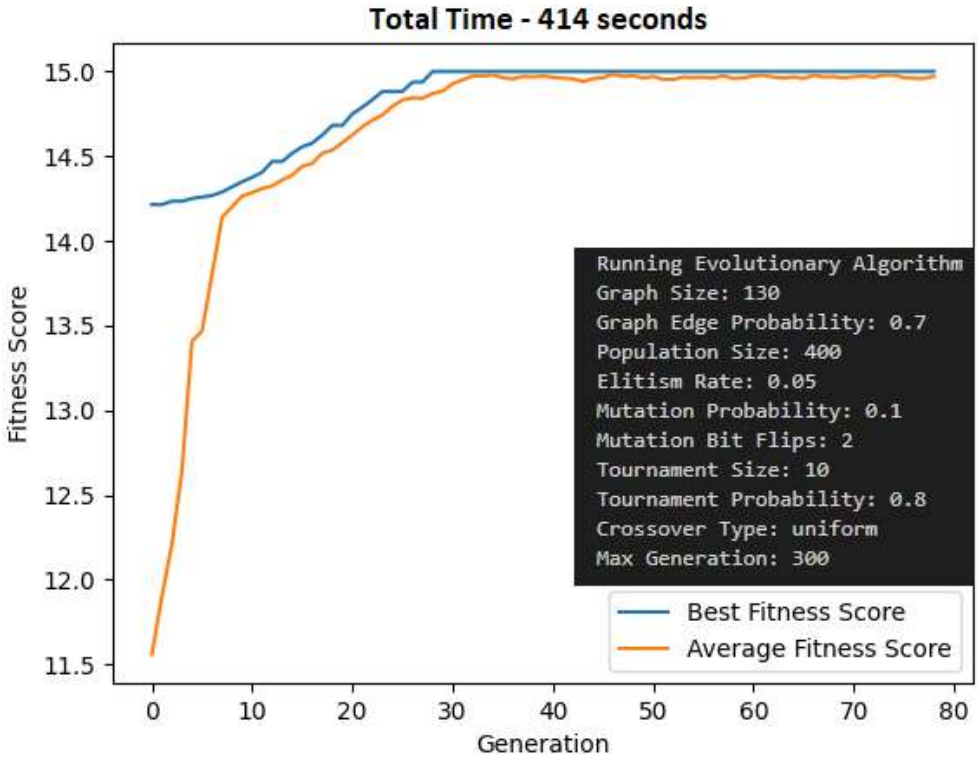
פרמטר שמשפיע בצורה ניכרת על זמן הריצה של האלגוריתם הוא ההסתברות להיווצרות קשת בגרף, כלומר כמות הקשתות בגרף. האלגוריתם עובד לאט יותר בצורה משמעותית עבור גרף בעל הרבה קשתות, כמו שניתן לראות בשתי הדוגמאות הבאות, בהן האלגוריתם קיבל פרמטרים זהים פרט להסתברות להיווצרות קשת.
הריצה בה ההסתברות לקשת הייתה 0.3 ארכה כ-9 שניות, לעומת הריצה עם הסתברות של 0.8 שארכה כ-100 שניות.

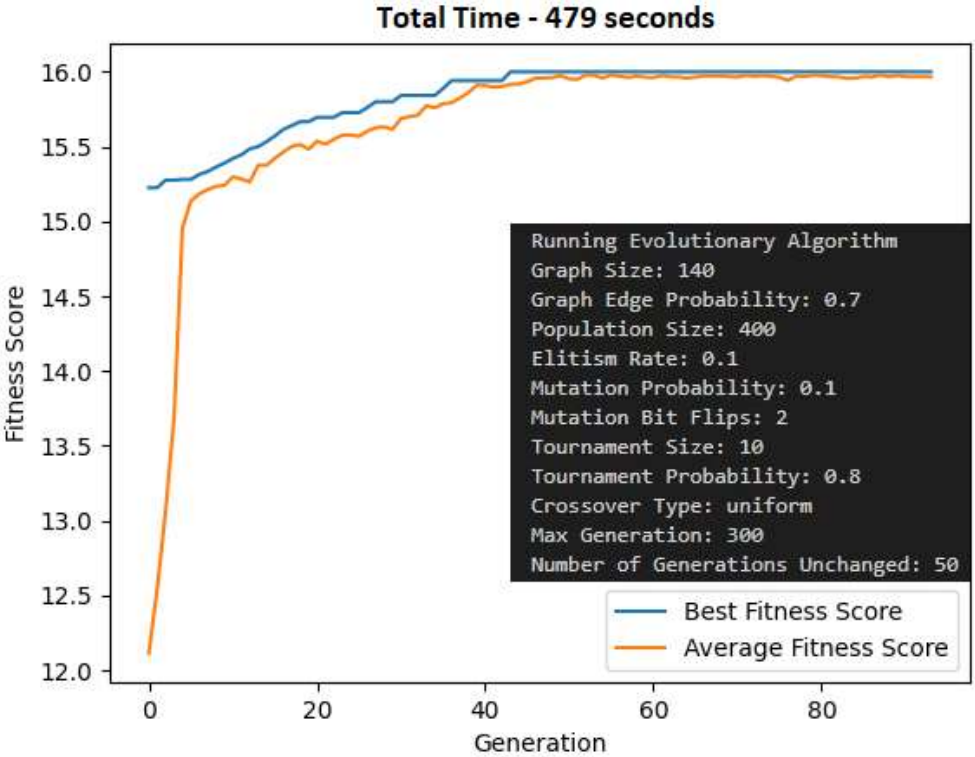




לסיום צירפנו שלוש דוגמאות לריצות על גרפים גדולים במיוחד, ניתן לראות כי בגדלים אלו האלגוריתם רץ זמן ארוך ונהיה פחות יעיל משמעותית בפתרון הבעיה.






Total Time - 479 seconds
Fitness Score
Generation
16.0
15.5
15.0
14.5
14.0
13.5
13.0
12.5
12.0
0
20
40
60
80
Running Evolutionary Algorithm
Graph Size: 140
Graph Edge Probability: 0.7
Population Size: 400
Elitism Rate: 0.1
Mutation Probability: 0.1
Mutation Bit Flips: 2
Tournament Size: 10
Tournament Probability: 0.8
Crossover Type: uniform
Max Generation: 300
Number of Generations Unchanged: 50
Best Fitness Score
Average Fitness Score

מסקנות

החלק הקריטי והמסובך באלגוריתם היה שיטת האוולואציה. ראינו כי לא מספיק לומר באופן כללי אם הקליקה חוקית או אם קודקוד מסויים קיבל ערך נכון או לא. נדרשנו לרדת לעובי הקורה ולנתח לעומק את הציון שניתן. כדי לעשות זאת בחנו את המקרים הבאים: כמה משמעותי אם קודקוד קיבל 1 אך היה צריך לקבל 0 וגם להפך. כמה חשובים מספר הקשרים של קודקודים מתוך הקליקה לבחוץ, ולהפך. וגם, כמה צריך להתחשב בגודל הקליקה הנתונה לעומת הגודל האמיתי אותו רצינו לקבל. את כל אלו 'כימתנו' ונתנו להם ערך מספרי. לאחר מחשבה רבה, ניסוי וטעייה, הצלחנו למצוא נוסחה שמייצגת בצורה טובה את טיב האינדיבידואל, מה שאכן גרם לדורות להשתפר מאחד לשני, ולהגיע לבסוף לתוצאה הנכונה.

תהליך האבולוציה עבד בצורה טובה ומצא יחסית מהר את הקליקה המקסימלית בגרף. גם בגרפים גדולים יחסית (200+ קודקודים) תוך מספר קטן יחסית של דורות קיבלנו את התוצאה, אך זמן האבלואציה בכל דור גדל.

שמנו לב שמאחר ושיטת אוולואציה עובדת נכון, כבר תוך דור אחד או שניים נקבל אינדיבידואלים עם ציון גבוה מאוד, קרוב ביותר לציון אותה הייתה מקבלת הקליקה המקסימלית.

אבחנה זאת עוזרת לנו בכך שקיימות בעיות רבות בהן אין הכרח לתוצאה מקסימלית, אלא מספיקה תוצאה קרובה במידה מסוימת לתוצאה האופטימלית. כלומר, במקרה שלנו גם אם נקבל קליקה גדולה מספיק, או לחילופין באינדיבידואל יהיו מספר קודקודים מסוימים שקיבלו את הציון 1 אך אינם שכנים (כלומר קליקה לא חוקית), אך עדיין תוצאות אלו יספקו אותנו. במקרים אלו, נוכל לבחור מספר קטן של דורות, לקבל תוצאה טובה מאוד (אך לא מדוייקת) ולעבוד איתה. כך נוכל לחסוך זמן ריצה וזיכרון רבים.