פתרון בעיית צביעת הגרפים בעזרת אלגוריתם גנטי ואלגוריתמי Beam-Search

פרוייקט סוף

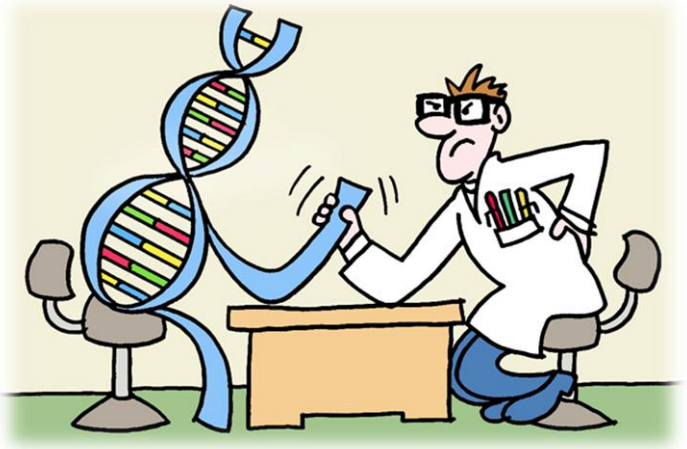
בינה מלאכותית

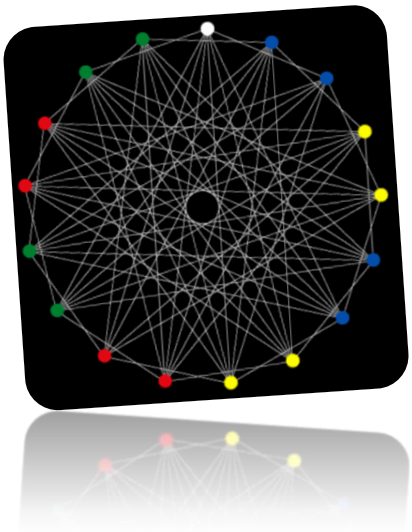
מגישים

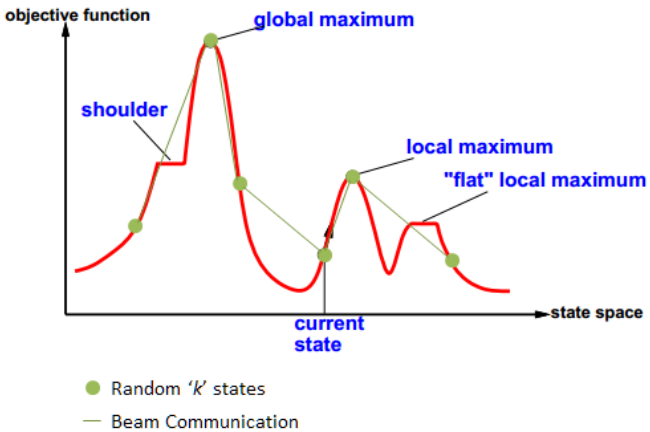
יונתן איתי

תמר בר-אילן

מרץ 2013







הקדמה

במאמר זה נעסוק בנושא של אלגוריתמי חיפוש מקומיים. באמצעותם ננסה לפתור את בעיית צביעת הגרפים, עליה נרחיב בהמשך. האלגוריתמים בהם נשתמש יהיו Local-Beam-Search, Stochastic-Beam-Search ואלגוריתם גנטי. בהמשך נחקור את הפרמטרים השונים והשלבים השונים בכל אחד מן האלגוריתמים הללו, והראה כיצד הם משפיעים על פתרון הבעיה שלנו.

אלגוריתמי חיפוש מקומיים באים לעזרתנו בפתרון בעיות מסובכות – כלומר בעיות שפתרונן בדרכים אחרות הוא בסיבוכיות זמן גבוהה – ובעזרתם אנחנו מנסים להגיע לפתרון בזמן מועט יותר. עם זאת, נזכור כי אלגוריתמים אלו לא מבטיחים לנו אף פעם פתרון מלא של הבעיה. השימוש בחיפוש מקומי מתאפשר כאשר אנחנו מחפשים איזשהו "מצב מטרה", אך הדרך אליו אינה מעניינת אותנו.

הדיון שלנו יתבסס על נתונים אמפיריים מהרצת התוכנה שבנינו, דרכם ננסה לעמוד על ההבדלים בין האלגוריתמים השונים שהצענו, ועל הפרמטרים האופטימליים עבורם לשם פתרון הבעיה. הקורא מוזמן להתרשם ע"י תפעול התוכנה המצורפת, בעלת ממשק משתמש נוח המאפשר לבדוק ולשנות מספר מאפיינים ולראות את הפתרונות המסופקים בכל פעם.

בעיית צביעת הגרפים

נסקור בקצרה את הבעיה עבורה ננסה למצוא פתרון יעיל במאמר זה, ואת מאפייניה.

יהי גרף בלתי-מכוון G, בעל קבוצות קודקודים (Vertices) V, ובעל קבוצת צלעות (Edges) E. צביעה פירושה מתן ערכים שונים ("צבעים") לכל הקודקודים בגרף, כך שתישמר ההגבלה הבאה:

- ✓ תהי צלע $e \in E$, (x,y) כך ש-$x,y \in V$. אזי הצבע של x שונה מהצבע של y. כלומר: $C(x) \neq C(y)$.

מכאן כמובן ניתן לראות שכל גרף המכיל צלע שמחוברת לקדקוד אחד משני "צדדיה", לא ניתן למצוא לגביו צביעה חוקית, ולכן לא נדון בגרפים כאלה סביב בעיה זו.

מספר הגדרות בסיסיות:

- ✓ צביעה המשתמשת לכל היותר ב-k צבעים נקראת ***k*-צביעה**.
- ✓ מספר הצבעים הקטן ביותר הנדרש כדי לצבוע גרף G נקרא **המספר הכרומטי** שלו, ובד"כ מסומן כ-$\chi(G)$.
- ✓ גרף שניתן לצביעה תקינה ע"י k-צביעה ייקרא ***k*-צביע**.
- ✓ גרף ייקרא ***k*-כרומטי**, אם המספר הכרומטי שלו הוא בדיוק k.

סיבוכיות הזמן בפתרון הבעיה:

הבעיה של מציאת k-צביעה מסוימת, ע"י אלגוריתמים מדויקים (לא למשל אלגוריתמים הסתברותיים או אלגוריתמי קירוב), היא בסיבוכיות אקספוננציאלית. אם ננסה לפתור את הבעיה בחיפוש Brute-Force, אז עבור גרף בעל n קדקודים, נצטרך לעבור על k^n השמות אפשריות של צבעים לקדקודי הגרף. ישנם מספר אלגוריתמים מתוחכמים יותר, המתבססים על משפטים מתמטיים מורכבים בתורת הגרפים, או ספציפיים אך ורק ל-k מסוים, בעלי סיבוכיות טובה יותר. עם זאת, גם אלגוריתמים אלו אקפוננציאלים בגודל הקלט, ולכן מתאימים אך ורק עבור גרפים קטנים במיוחד.

עובדות אלו הביאו אותנו להתעסק בבעיה זו, ולנסות למצוא אלגוריתם הפותר אותה עבור מספר משמעותי של מקרים, ובזמן סביר.

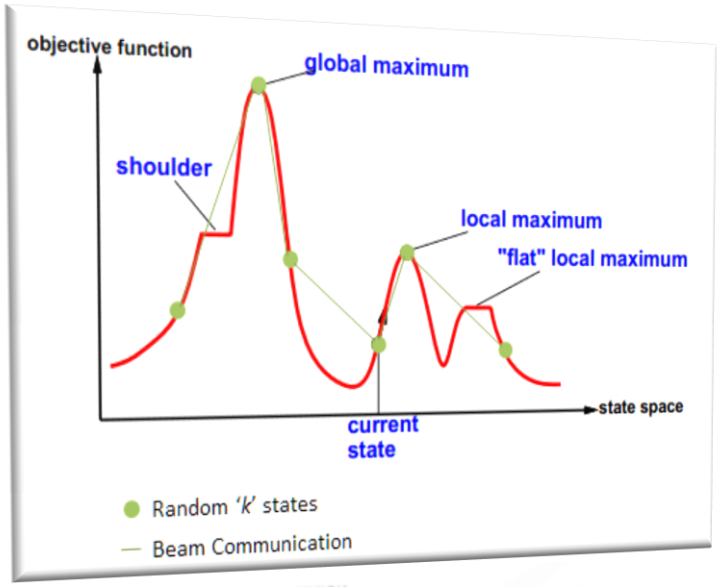
חיפוש מקומי

בחלק זה נסקור בקצרה את העיקרון לפיו פועל אלגוריתם חיפוש מקומי. באלגוריתם מסוג זה הפתרון שאנחנו מחפשים לבעיה מסוימת נמצא במרחב מצבים כלשהו, בדרך כלל מאוד גדול (ולכן מראש לא נלך בגישה היותר "סטנדרטית" לפתרון הבעיה). בחיפוש מקומי ניקח מצב התחלתי או מספר מצבים התחלתיים (תלוי האלגוריתם הספציפי), וננסה להגיע ממצבים אלו אל "מצב המטרה" שלנו, כלומר אל מצב שנמצא בקבוצת הפתרונות של הבעיה שלנו.

נוכל להבין זאת טוב יותר מהתבוננות בבעיה בה נעסוק במאמר זה. הבעיה שלנו היא כזו – נתון לנו גרף כלשהו G עם קבוצת קודקודים V וקבוצת צלעות E, ואנחנו צריכים למצוא עבורו k-צביעה (במידה והגרף אכן k-צביע) כלשהי. כלומר מרחב המצבים שלנו הוא השמות של צבעים לכל אחד מהקדקודים – וקטור בגודל מספר הקדקודים בגרף, שהקואורדינטות שלו הן הצבע המתאים לכל קדקוד. במקרה זה קבוצת הפתרונות יהיו הווקטורים המתארים צביעה חוקית של הגרף, כלומר הווקטורים בהם לא קיימים זוג קדקודים הקשור לאותה הצלע ובעל אותם ערכים של צבעים (אותן ערך בקואורדינטה המתאימה). המטרה שלנו תהיה להתחיל ממספר מצבים התחלתיים אקראיים, שאינם דווקא שייכים לקבוצת הפתרונות, וע"י סדרת צעדים, להגיע למצב כלשהו ששייך לקבוצת הפתרונות של הבעיה (במידה וזו לא קבוצה ריקה כמובן, כלומר הגרף אכן k-צביע).

מהי אותה "סדרת צעדים" שתצליח להביא אותנו אל הפתרון המיוחל? אלגוריתם חיפוש מקומי מנסה לחפש מצבים **שכנים או יורשים** למצבים הנוכחיים שלנו, ולהתקדם עם הזמן לקבוצת מצבים **טובה** יותר. מהמשפט האחרון אפשר להבין שאנחנו צריכים להגדיר בכל אלגוריתם חיפוש מקומי את מושג ה**שכנות**, ואת מושג **טיב** המצב, כלומר עד כמה הוא קרוב לפתרון. בתיאור כל אחד מהאלגוריתמים שהשתמשנו נסביר כיצד קבענו אלו מצבים הם השכנים של מצב כלשהו. את **טיב** כל אחד מהמצבים, אנחנו מעריכים בעזרת **היוריסטיקה**, ונסביר על זאת בחלק הבא.

נעמוד כעת על היתרונות באלגוריתם חיפוש, וכן על נקודות התורפה שלו. היתרון המרכזי הראשון באלגוריתם חיפוש מקומי הוא צריכת הזיכרון שלו. באלגוריתם חיפוש נשמור תמיד כמות **קבועה** של מצבים, ולכן גודל הזיכרון שנזדקק לו לא תלוי בגודל הקלט, אלא קבוע. אלגוריתם חיפוש מקומי בדרך כלל מאפשר לנו למצוא פתרון "סביר" גם במרחב מצבים גדול מאוד או אינסופי, ובסיבוכיות זמן פולינומית. עם זאת, חשוב לעמוד גם על החסרונות שטבועים במהות של אלגוריתם חיפוש **מקומי**.



כפי שהסברנו קודם, יש לנו איזושהי פונקציית הערכה שמעידה על "טיב" המצב שלנו. בשביל למצוא פתרון אנחנו מחפשים איזשהו מקסימום או מינימום גלובלי של הפונקציה הזו, כלומר מקסימום של הפונקציה על כל מרחב המצבים. חיפוש מקומי פועל תמיד לפי המצבים שאנחנו מחזיקים כרגע בזיכרון, ולפי השכנים שלהם, ומאופן הפעולה שלו אנחנו יכולים במקרים רבים "להיתקע" במקסימום מקומי. כלומר במידה והגענו למצב מסוים, שאין לו שכנים בטווח הקרוב שהם "טובים" ממנו, מבחינת אלגוריתם החיפוש המקומי מצב זה ייחשב כמצב הכי קרוב לפתרון הבעיה שניתן להגיע אליו. במקרה זה לא נגיע לפתרון הטוב יותר – המקסימום הגלובלי. באיור למעלה ניתן לראות איזושהי המחשה לנושא זה, עבור מקסימום מקומי או "כתף" בגרף המצבים. יש לציין שהמגוון הגדול של אלגוריתמי החיפוש בא כדי לענות הרבה פעמים על הצורך הזה של להימנע מ"מקסימום מקומי". לאלגוריתמים שונים מגוון דרכים שעוזרות להם להימנע ממצבים כאלו. למשל, שלושת האלגוריתמים בהם אנחנו השתמשנו, מתחילים מ**קבוצה** של מצבים התחלתיים, ומשתמשים ב**שיתוף מידע** בין ענפי החיפוש השונים הקשורים לכל מצב, בכדי להימנע מתקיעה במקסימום מקומי או כתף בגרף המצבים.

היוריסטיקה

כפי שהסברנו קודם, אלגוריתם חיפוש מחזיק פונקציה שמעריכה את טיב המצב, ועל פי פונקציה זו האלגוריתם בוחר עם אלו מצבים להמשיך את החיפוש. במקרה שלנו, בחרנו בפונקציה פשוטה מאוד. ההיוריסטיקה שלנו סופרת את מספר הקונפליקטים, בצביעה שמתוארת ע"י וקטור המצב שלנו. קונפליקט מוגד כאן כמקרה בו קדקוד מסוים צבוע בצבע זהה לקדקוד אחר הקשור אליו בצלע. במקרה שלנו אנו מנסים להביא את הפונקציה הזו למינימום – עד ל-0. במידה והגענו לכך, מצאנו מצב שהוא בקבוצת הפתרונות של הבעיה שלנו, שכן אם אין בו קונפליקטים מדובר בצביעה חוקית לגרף. כמובן שניתן לבנות פונקציות הערכה מתוחכמות בהרבה מזו שתיארנו כאן, אך בשלבי הניסוי של האלגוריתמים שלנו ראינו כי פונקציה זו בהחלט מספיקה לצרכים שלנו.

Local Beam Search

בתחילת הריצה האלגוריתם בוחר באקראי קבוצת מצבים בגודל מסוים, זהו הדור הראשון. על מנת ליצור את הדור הבא, האלגוריתם יתבונן בכלל היורשים של הדור הנוכחי. קבוצת היורשים של מצב מסוים יהיו כלל המצבים אשר זהים למצב המקורי עד כדי צבעו של קדקוד יחיד. האלגוריתם יפסיק את ריצתו כאשר נמצא פתרון חוקי או כאשר הוא יתקבע על אוכלוסייה מסוימת.

ניתן להבחין כי אלגוריתם זה יפעל בצורה קיצונית יחסית. כיוון שהוא מחפש את היורשים הטובים ביותר הוא תמיד מתקדם בכיוון בו השיפוע הוא החד ביותר. מכאן כי האלגוריתם יתכנס אל מקסימום, מקומי או גלובלי, באופן מהיר יחסית, כלומר, אחרי מספר קטן של איטרציות. נשים לב שלעומת יתרון זה קיים החיסרון של התכנסות אל מקסימום מקומי: מכיוון שהאלגוריתם תמיד בוחר בשיפור המשמעותי ביותר אז אם תנאי ההתחלה של האלגוריתם שמים אותו בקרבת מקסימום מקומי סביר שהוא יטפס אליו. מעבר לכך, ברגע שהאלגוריתם הגיע למקסימום מקומי כלשהו הוא לעולם לא יצליח לחמוק ממנו.

תכונה מעניינת נוספת של אלגוריתם זה הוא היותו דטרמיניסטי. עבור תנאי התחלה מסוימים, כלומר, עבור אוכלוסייה התחלתית מסוימת, האלגוריתם יניב תמיד את אותו אוכלוסייה סופית - יתכנס תמיד לאותו פתרון. היותו של האלגוריתם חסר אלמנט הסתברותי מקשה עליו לחמוק ממקסימום מקומי: אם הוא נמצא בקרבתו, אין סיכוי שיבחר להתקדם שלא בכיוון הפסגה הקרובה אליו ולהימנע מלהגיע אליה.

פרמטר חשוב באלגוריתם זה הינו גודל האוכלוסייה. ככל שזו גדולה יותר אז האלגוריתם יהיה מסוגל לסרוק חלק גדול יותר ממרחב המצבים. כלומר, אוכלוסייה גדולה תבוא לידי ביטוי בכך שהאוכלוסייה ההתחלתית תהיה פרוסה באזורים רבים במרחב ובכך תאפשר לאלגוריתם לטפס על מספר רב של מקסימות. כמובן שזה יגדיל את הסיכוי להגיע אל המקסימום הגלובלי. נשים לב כי גם כאן העובדה שהאלגוריתם מחפש את היורשים הטובים ביותר יכול להחליש מאד את היתרון שבאוכלוסייה גדולה: אם מצב אחד ויורשיו חולשים על מצבים אחרים אז במהרה הם ישתלטו על אותם מצבים ובדורות הבאים כבר לא יהיו מצבים מהאזור הנמוך.

נוכל לשים לב כבר בשלב זה כי בקבוצת היורשים של מצב מסוים איברים רבים, ולכן זמן הריצה של תהליך יצירת דור חדש יהיה יחסית לא מבוטל.

Stochastic Local Beam Search

אלגוריתם זה הינו גרסה שונה של ה- local beam search. השוני מתבטא בהוספת היבט של אקראיות. במקום שכל דור ייווצר מתוך היורשים המוצלחים ביותר של הדור שקדם לו, נבחר ב**אקראי** יורשים מתוך קבוצת היורשים המוצלחים יותר מהמצב ממנו נוצרו. מבחינה טכנית יותר, על מנת ליצור דור חדש נבחר רנדומלית מצבים מהדור הקודם. עבור כל מצב שבחרנו, נגריל לו יורשים עד שנקבל מצב שמהווה שיפור שלו, ואותו נכניס לדור החדש. נדגיש כי אנו מחפשים שיפור **ממש** לעומת המצב המקורי. כלומר, אם מצאנו מצב שזהה בטיבו למצב המקורי נמשיך בחיפוש. בגלל שאנחנו בוחרים יורשים באקראי, נרצה להגביל את מספר הניסיונות למצוא שיפור של האב לערך כלשהו (זהו פרמטר שבדקנו את השפעתו על ריצת האלגוריתם). ייתכן כי לא נמצא יורש המהווה שיפור לעומת אביו במספר הניסיונות שהגדרנו, הן בשל האופי ההסתברותי של חיפוש היורש, והן כי ייתכן והוא לא קיים. מעבר לכך, ייתכן כי היורש הנבחר מהווה הרעה ממש. זהו האלגוריתם היחיד מבין האלה שבדקנו שבו ייתכן כי דור מסוים מהווה הרעה לעומת הדור שקדם לו. במקרה ולא נמצא שיפור נבחר להכניס לדור החדש את היורש המוצלח ביותר מבין כל היורשים שבדקנו. האלגוריתם יפסיק את ריצתו באותם תנאים כמו ה- local beam search.

העובדה שלאלגוריתם זה אין נטייה ברורה להתקדם בכיוון בו השיפוע הוא החד ביותר, אלא מקיים כלל נוקשה הרבה פחות, וזה התקדמות בכיוון בו יש שיפוע חיובי, תורמת מאד במציאת המקסימום הגלובלי. פעמים רבות, על מנת להגיע לאותו מקסימום מהמצב בו אנו נמצאים עכשיו לא נצטרך ללכת בכיוון בו השיפוע הוא החד ביותר, אלא להתקדם בכיוון אחר לחלוטין.

כפי שציינו מוקדם יותר, ייתכן כי פונקציית ההערכה תמצא כי דור מסוים עדיף פחות על הדור שקדם לו. כלומר, הוא רחוק יותר מפתרון חוקי לבעיה, לפי פונקציית ההערכה. זהו האלגוריתם היחיד בו מצב כזה הוא אפשרי. אמנם גם באלגוריתם הגנטי ייתכן מצב בו מופקים ממצבים מסוימים מצב מוצלח פחות, אך לא ייתכן כי דור שלם מהווה הדרדרות לעומת כל מצב בדור שקדם לו. לעובדה זו השלכות מעניינות, אליהן נתייחס בהמשך.

בניגוד לאלגוריתם הלא הסתברותי שעסקנו בו, באלגוריתם זה הגדלת האוכלוסייה יכולה לבוא לידי ביטוי ביתר עוצמה. שכן, לכל מצב הסתברות שווה להיבחר להעביר יורשים לדור בא, ולכן, האלגוריתם יסייר ויבדוק את רוב המצבים המצויים באוכלוסייה שלו בכל רגע. מכאן כי בהסתברות גבוהה יותר יגיע אל המקסימום הגלובלי.

אלגוריתם גנטי

נעבור כעת לתיאור המרכיבים של אלגוריתם גנטי כללי, ולתיאור הבחירות שאנחנו עשינו בדרכי המימוש של האלגוריתם הגנטי המותאם לפתרון הבעיה שלנו.

סקירה כללית:

אלגוריתם גנטי מתבסס על רעיונות מתורת האבולוציה בטבע. לפי האבולוציה עקרון "הברירה הטבעית" שולט בהתפתחות המינים בטבע, כאשר לאורך מספיק זמן (מספיק דורות) נצפה לשינוי באוכלוסיות של מינים, לפי ההתאמה שלהם לסביבתם. נצפה לעליה בשכיחות אוכלוסיות של מינים המותאמים יותר לסביבתם, ובתמצות – "המתאים יותר שורד" (בניגוד לסיסמא הידועה "החזק שורד"). אנחנו נמדל את הבעיה שלנו כך שנשמור על כללים מסוימים המתקיימים בטבע, כמו למשל תורשה ומוטציות שיוצרות מגוון גנטי, וכן נגדיר "התאמה" לפי ההיוריסטיקה שלנו, עליה כבר דנו לפני כן. אז, ניצור מעין הפעלה מדומה של עיקרון הברירה הטבעית, ונצפה להישאר עם אוכלוסייה יותר ויותר מותאמת, כלומר מצבים יותר ויותר קרובים לפתרון הבעיה.

אלגוריתם גנטי פועל באופן הבא. בהתחלה מוגרלת אוכלוסיה אקראית של מצבים. הדור הבא יהיה מורכב בחלקו מהפרטים הטובים ביותר (כלומר אלו בעלי פונקציית ההתאמה הגבוהה ביותר) מהדור הקודם, ובחלקו מ"זיווג" של פרטים אלו. "זיווגים" אלו יתבססו על שני ה"הורים" שלהם, ויתרחשו בהם מוטציות בהסתברות מסוימת, כלומר שינויים אקראיים כלשהם. בתהליך זה נוצר דור חדש. נמשיך כך הלאה לאורך מספר רב של דורות, ולאורך זמן נצפה ל"השבחת" הפרטים שלנו, כלומר לעליה ממוצעת בפונקציית ההתאמה שלהם.

נשים לב כי במקרה שלנו השכנות במצבים, מוגדרת לפי ה"תורשה", כלומר לפי זיווג של שני פרטים, ואילו טיב כל מצב מוגדר לפי פונקציית ההתאמה שלו.

פסודו - קוד של אלגוריתם גנטי סטנדרטי:

1. **אתחל** אוכלוסייה התחלתית של מצבים באופן אקראי.
2. הערך את **ההתאמה** ($\boldsymbol{fitness}$) של הפרטים באוכלוסייה.
3. כל עוד תנאי הסיום לא מולאו, צור אוכלוסייה חדשה של מצבים באופן הבא:
   a. **ברור פרטים** ($\boldsymbol{selection}$) מתוך האוכלוסייה ע"פ ההתאמה שלהם.
   b. הפעל עליהם אופרטורים גנטיים – **זיווג** ($\boldsymbol{crossover}$) **ומוטציות** ($\boldsymbol{mutation}$), ליצירת פריטים חדשים באוכלוסייה, עד ליצירת אוכלוסייה חדשה בגודל המקורי.
   c. הערך את **ההתאמה** ($\boldsymbol{fitness}$) של הפרטים באוכלוסייה.
4. **סיים**.

מימוש האלגוריתם לפתרון הבעיה שלנו:

- ✓ **ייצוג** – בדומה לאלגוריתמים הקודמים, כל מצב מיוצג ע"י object מסוג Chromosome, שמחזיק ווקטור עם קואורדינטה לכל קדקוד, כך שהערך בה מצביע על הצבע שניתן לו.

עבור n קדקודים בגרף

- ✓ **התאמה** ($\boldsymbol{fitness}$) – שוב בדומה לאלגוריתמים הקודמים, פונקציית ההתאמה סופרת את מספר הקונפליקטים, מספר המקרים בהם קדקוד מחובר בצלע לקדקוד אחר, שצבוע בצבע זהה לו. נשים לב שככל שפונקציית התאמה נמוכה יותר, כך ההתאמה טובה יותר.

$\boldsymbol{chromosom}$

- ✓ **יצירת הדור החדש**– בכל דור השארנו חלק כלשהו קבוע מהאוכלוסייה (בקוד: REMAINING_PART) – הפריטים (chromosomes) שהיו בעלי ההתאמה הגבוהה ביותר.

f

לאחר מכן ייצרנו פריטים חדשים שישלימו את האוכלוסייה לגודל המקורי, ע"י זיווג של פריטים אלו. לפני כן היינו צריכים לבחור מאילו פריטים לייצר את הזיווג.

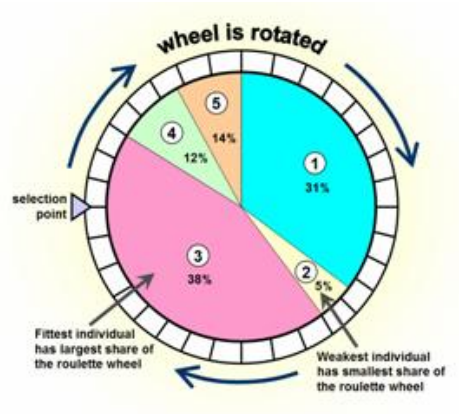
- ✓ **בחירה** ($\boldsymbol{selection}$) – השתמשנו בשתי שיטות לבחירת המיועדים לזיווג:
  - **שיטת הטורניר**: בשיטה זו בשביל לבחור הורה, בחרנו באקראי שני פריטים מהאוכלוסייה (אלה שכבר עברו לשלב הבא) והשארנו את המתאים מביניהם.

Selection1: Tournament Selection

C1 = select random chromosome from the remaining par

C2 = select random chromosome from the remaining part

return the chromosome with the better fitness



- **רולטה** $(\boldsymbol{Roulette\ Wheel\ Selection})$: בשיטה זו, ככל שפונקציית ההתאמה שלך טובה יותר, כך גדלים סיכוייך להיבחר. בגלל שככל שה-*fitness* של *chromosome* יותר קטן, כך אתה הוא מתאים, ההסתברות לבחור ב-*chromosome* תהיה ביחס הפוך ל-*fitness* שלו. כלומר:

$$\mathrm{P}(X=c)=\frac{fitness^{-1}(c)}{\sum_{i=0}^{pop\ size} fitness^{-1}(i)}$$

<u>Selection2: Roulette Wheel Selection</u>

*random = (random number in (0,1))**

(sum of c. $fitness^{-1}$ *for every chromosome c in population)*

iterate through population and for every chromosome c subtract c.$fitness^{-1}$

until random is bigger than 0.

return the chromosome which was last added

✓ **זיווג** $(\boldsymbol{crossover})$ – לאחר שהשארנו את החלק באוכלוסייה המתאים ביותר, רצינו ליצור מצבים חדשים ע"י זיווג בין המצבים הנותרים (המתאימים ביותר). לשם כך בכל פעם נבחר זוג של כרומוזומים באמצעות שיטות ה-*selection* שהסברנו קודם, ועליהם נפעיל פונקציית זיווג ליצירת פרט חדש באוכלוסייה.

השתמשנו בשתי שיטות שונות לזיווג:

- **זיווג נקודה אחת (*one-point crossover*)** – במקרה הזה, נבחר באקראי קואורדינטה מסוימת בווקטור ה-*chromosome*. עד לקואורדינטה הזו, ה"ילד" יהיה זהה ל"הורה" ה-1. החל מהקואורדינטה הזו והלאה, ה"ילד" יהיה זהה ל"הורה" ה-2.

<u>Crossover1: one-point crossover</u>

Choose random coordinate r

for every coordinate i of the "child" chromosome:

if (i<r)

child(i) = parent1(i)

otherwise

child(i)=parent2(i)

- **אחיד (*uniform crossover*)** – כל קואורדינטה בווקטור "הילד", מתקבלת מאחד "ההורים", בהסתברות פורפורציונית להתאמה של ההורה. ככל שההתאמה של ההורה טובה יותר, כלומר נמוכה יותר, כך ההסתברות לקבל ממנו את הקואורדינטה גדולה יותר.

<u>Crossover2: uniform crossover</u>

for every coordinate i of the "child" chromosome:

with probability (fitness(parent2)/(fitness(parent1)+fitness(parent2))):

child(i) = parent1(i)

otherwise

child(i)=parent2(i)

- ✓ **מוטציה *(mutation)*** – בשלב זה נערוך שינויים כלשהם ב"ילדים" שייצרנו – מוטציות. המוטציות הן אלו שיוצרות בעצם פרטים שמכילים מידע חדש (שלא כולו הגיע מאחד מ"ההורים"), ולכן יש להם תפקיד מרכזי בשיפור האוכלוסייה עם יצירת דורות חדשים. לגבי כל "ילד" שייצרנו ע"י זיווג, הפעלנו עליו מוטציה בהסתברות שקבענו מראש *(MUTATION_RATE)*. בסה"כ עבדנו עם שתי שיטות למוטציות:
  - **השיטה האקראית** – בשיטה זו פשוט החלפנו באקראיות בין שתי קואורדינטות של ה"ילד"

<u>Mutation1:</u>

Choose two random coordinates i,j and then

switch between child(i) and child(j)

  - **השיטה המוכוונת** – בשיטה זו עוברים על כל אחד מהקדקודים בגרף, המיוצגים ע"י הקואורדינטה ב-*choromosome*, ובודקים האם הוא נמצא בקונפליקט עם קדקוד אחר – כלומר אם יש לו קדקוד שכן הצבוע באותו הצבע.

במידה וכן, מוצאים את כל הצבעים שבהם מותר לצבוע את הקדקוד, כלומר כל הצבעים שאין **כרגע** אף שכן של הקדקוד שצבוע בהם. בוחרים צבע באקראי מהצבעים הללו, וצובעים בו את הקדקוד.

<u>Mutation2:</u>

For every vertex i in the graph (coordinate i in the chromosome)

if(i has conflict)

allowed colors = all colors minus the colors of i's neighbors

choose random color c from allowed colors

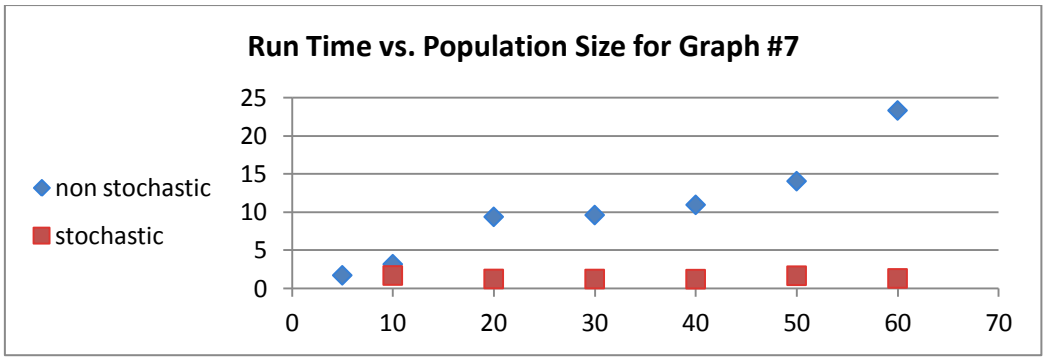
child(i)=c

תוצאות

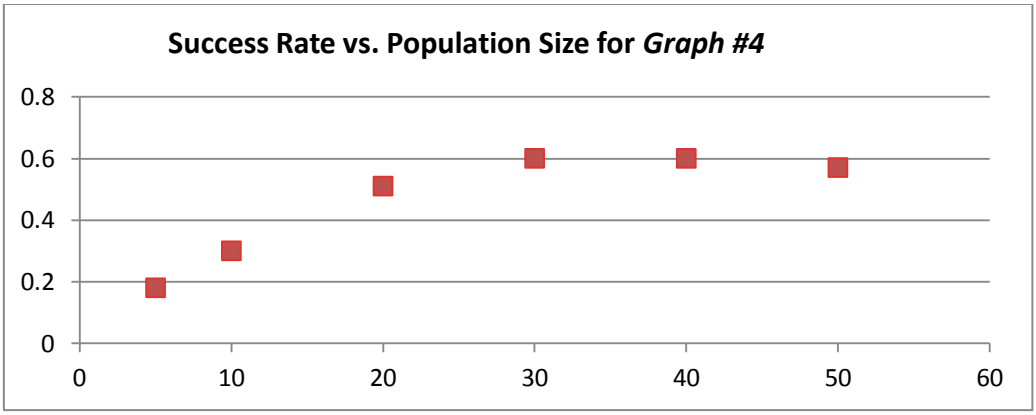
לאחר שסקרנו את עקרון פעולתם של כל אחד מאלגוריתמי החיפוש המקומי שבדקנו, ולאחר שפירטנו על המימוש אותו יישמנו עבורם, נעבור לחלק של תיאור התוצאות. בחלק זה נפרט כיצד נפתרה בעיית צביעת הגרפים, עבור גרפים שונים ובתנאים שונים, כאשר נמצא כיצד הפרמטרים השונים של כל אלגוריתם משפיעים על הפתרון ומהירותו, וננסה למצוא אילו פרמטרים הם פחות או יותר אופטימליים לכל אחד מהאלגוריתמים. לאחר מכן נשווה בין ביצועיהם של אלגוריתמי החיפוש השונים, מבחינת פרמטרים שונים הקשורים בפתרון – ובעיקר הזמן שנדרש לאלגוריתם כדי להגיע אליו.

עבור אלגוריתמי ה – *local beam search* (ההסתברותי והלא הסתברותי):

נציג תחילה את השפעת גודל האוכלוסייה על זמן הריצה של האלגוריתמים. נבהיר את הכוונה במונח "זמן הריצה": ממוצע של משך זמן הריצה, כאשר לא מחשיבים ריצות שבהן האלגוריתם כשל במציאת צביעה חוקית. מצאנו כי זמן הריצה של האלגוריתם הלא הסתברותי מונוטוני עולה בגודל האוכלוסייה. אין זה מפתיע, בהתחשב בכך שכל התקדמות בדור כרוכה בבניית כלל היורשים של כל מצב באוכלוסייה הקיימת. לעומת זאת, עבור האלגוריתם ההסתברותי, בבדיקה על גרף זה, מצאנו כי זמן הריצה לא הושפע מגודל האוכלוסייה. יש לציין שעבור גרפים אחרים מצאנו קשר זהה לזה שקיבלנו עבור האלגוריתם הלא הסתברותי. ניתן לראות מגמות אלו בגרף הבא:

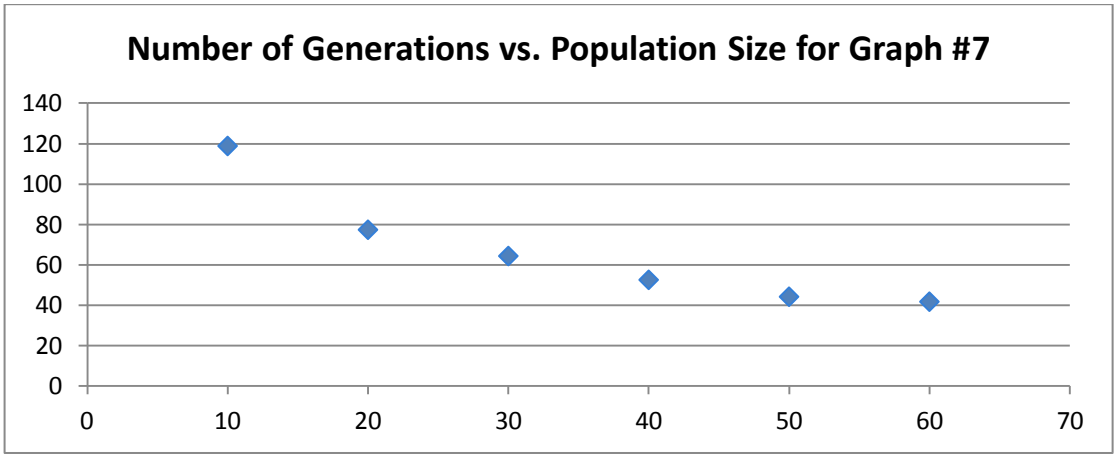


ראינו כי ככל שנגדיל את האוכלוסייה האלגוריתם הלא אקראי ישפר את אחוזי ההצלחה שלו, כלומר את אחוז הריצות בהן הוא מוצא צביעה חוקית לגרף. נסתייג ונאמר כי בשלב כלשהו, מגמה זו נפסקת, ואחוזי ההצלחה הופכים קבועים בגודל האוכלוסייה. מגמה זו היא הגיונית, שכן ככל שהאוכלוסייה גדלה האלגוריתם סורק חלק נרחב יותר ממרחב הפתרונות. מעניין לראות כי קיימת נקודה שממנה והלאה, גם אם נגדיל את האוכלוסייה לא נשפר את הסיכוי למצוא את המקסימום הגלובלי.



תופעה מפתיעה מאד שמצאנו היא כי מספר הדורות שנדרש לאלגוריתם הלא הסתברותי עד למציאת הפתרון כמעט ואינו תלוי בגודל האוכלוסייה. הסבר אפשרי לתופעה זו היא שעבור כל מצב התחלתי תוך מספר מסוים של דורות הוא יגיע לאיזשהו מקסימום שהיה בקרבתו. כלומר, תוך מספר קבוע של דורות כלל המצבים באוכלוסייה יהיו מקסימות.

לעומת זאת, עבור האלגוריתם ההסתברותי מספר הדורות הדרוש לפתרון היה מונוטוני יורד בגודל האוכלוסייה. מכאן כי אלגוריתם זה מהווה יותר שוטטות אקראית במרחב, ולכן הגדלת האוכלוסייה תקטין את מספר הצעדים במרחב שעלינו לבצע. מגמה זו מוצגת בגרף הבא:



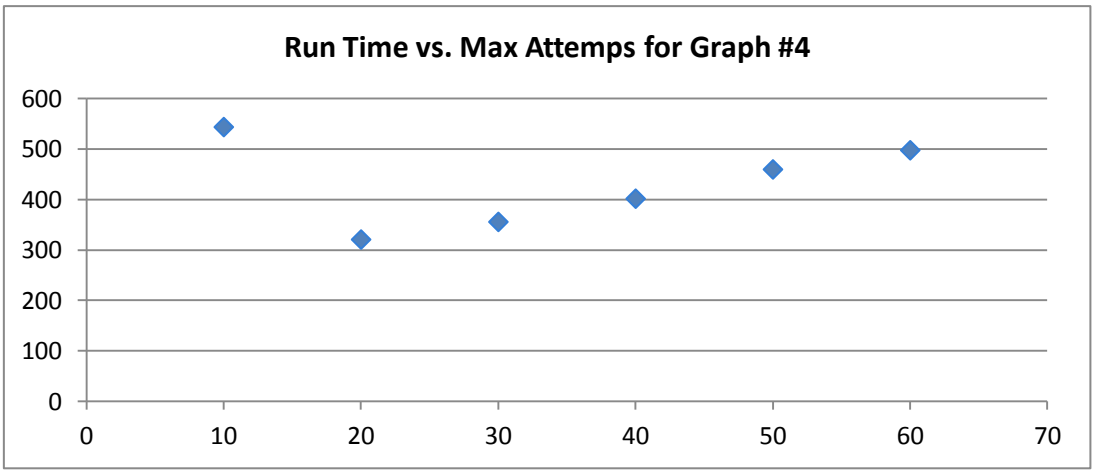
דבר נוסף שהבחנו בו הוא כי בכל ריצה מוצלחת של שני האלגוריתמים מספר הדורות שנדרש למציאת הפתרון היה קבוע יחסית. הכוונה כאן היא בין שתי ריצות עבור גודל אוכלוסייה קבוע. למשל, הרצנו את האלגוריתם הלא אקראי על גרף #5 כמה פעמים, ומצאנו כי מספר הדורות הממוצע שנדרש להגיע לפתרון היה 16.2, וסטיית התקן היתה 2.3. עבור האלגוריתם ההסתברותי הנתונים היו דומים.

נשים לב שנתון זה גוזר השלכות חשובות. לכל אלגוריתם, עלינו להגדיר את מספר הדורות המקסימלי שאנו נותנים לו לרוץ עד שמכריזים על כישלון. בגלל שמספר זה קבוע יחסית נקבל שהתלות של הצלחת האלגוריתם בפרמטר זה דומה לפונקציית מדרגה: מתחת לערך מסוים האלגוריתם ייכשל תמיד, ועבור כל מספר מעל אותו ערך לאלגוריתם יהיה אחוזי הצלחה זהים. אכן, בדיקה ישירה של שינוי מספר הדורות המקסימאלי איששה השערה זו. בכלל הריצות דאגנו לעבור ערך זה, ובכך לבטל השפעת פרמטר זה בתוצאות שקיבלנו.

האלגוריתם האקראי היה היחיד שהצליח, במידה מסוימת, לפתור בעיה קשה יותר מבעיית ה-*k* צביעה שהגדרנו: מציאת צביעה עם מספר צבעים נמוך מזה שהכנסנו. הגדרנו כי פונקציית ההערכה תשכלל גם את מספר הצבעים בפתרון המוצע, והכנסנו לאלגוריתם כערך התחלתי מספר צבעים גבוה. למשל, עבור גרף #4, עם כלל האלגוריתמים הצביעה הטובה ביותר שהצלחנו למצוא הייתה עם עשרה צבעים. אך עבור האלגוריתם האקראי, יכולנו לתת לו ערך התחלתי של עשרים צבעים, והוא בכל זאת יצליח ביותר ממחצית מהמקרים למצוא צביעה של עשרה צבעים. לדעתנו, דבר זה התאפשר בגלל העובדה שבריצת אלגוריתם זה ייתכן כי דור אחד גרוע מקודמו. תכונה זו, שהזכרנו בתחילת המאמר, אפשרה לאלגוריתם, לאחר שהוא כבר התכנס על פתרון חוקי עם מספר צבעים מסוימים, להתרחק מהפתרון, ובכך לתת לו הזדמנות לרדת במספר הצבעים.

פרמטר נוסף שמשפיע על האלגוריתם ההסתברותי הוא מספר הניסיונות שאנו מתירים על מנת למצוא יורש המהווה שיפור למצב שבחרנו (מתוך האוכלוסייה הקיימת). הזכרנו פרמטר זה בשלב סקירת

האלגוריתמים. לא מצאנו שיש לפרמטר זה השפעה על אחוזי ההצלחה של האלגוריתם. אך יש לו השפעה על זמן הריצה הממוצע. מצאנו כי קיים ערך אופטימלי לפרמטר זה, ובבדיקות האחרות קבענו אותו לערך אופטימלי זה.

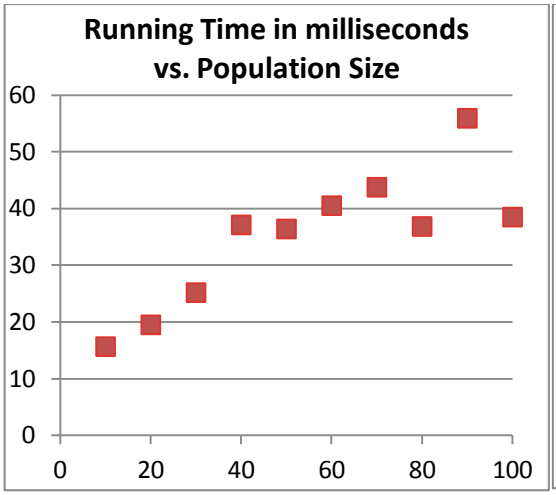


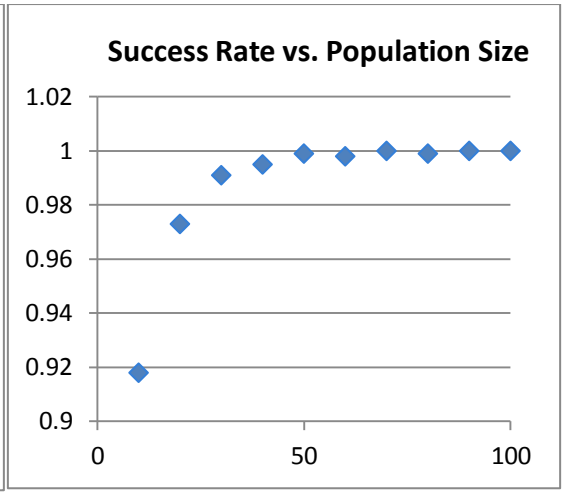
עבור האלגוריתם האקראי, ראינו כי בין ריצה לריצה זמן הריצה השתנה מאד. לראייה, עבור מספר רב של הרצות של האלגוריתם על אחד הגרפים מצאנו כי זמן הריצה הממוצע היה 344 מילי שניות וסטיית התקן הייתה 266 מילי שניות. נתון זה לכשעצמו אינו מטלטל במיוחד, אך בהתחשב בכך שמספר הדורות היה כמעט קבוע נסיק כי הזמן שנדרש לבנות דור השתנה מאד מריצה לריצה. חשבנו שהסיבה לכך היתה האופי ההסתברותי של האלגוריתם. משך הזמן הנדרש למצוא יורש מוצלח שיעבור לדור הבא, יכול להיות דינאמי מאד. זה נובע כמובן מכך שהאלגוריתם **מגריל** יורשים עד שהוא מוצא אחד מוצלח יותר מן המצב המקורי.

בדקנו האם העברת המצב המוצלח ביותר מתוך דור מסוים לדור הבא תשפר את ריצת האלגוריתם ההסתברותי. מבדיקותינו עלה כי פעמים רבות אנחנו מגיעים למצב כלשהו, ואז בדור שלאחר מכן כלל המצבים גרועים ממנו. כלומר המצב האיכותי ביותר בדור החדש נופל מזה שהיה בדור הקודם. עם מימוש צעד זה אנחנו לא מאפשרים הרעה במצב האיכותי ביותר בדור. בנוסף, אין בצעד זה כדי לשנות את האופי ההסתברותי של האלגוריתם ולהקשות עליו להתחמק ממקסימום מקומי, שכן מדובר רק במצב אחד מתוך אוכלוסייה גדולה. אך ראינו כי אפשרות כזו לא תועיל לריצת האלגוריתם, לא מבחינת אחוזי הצלחה ולא מבחינת זמן ריצה.

עבור האלגוריתם הגנטי:

בדומה להצגת התוצאות עבור האלגוריתמים הקודמים, נתחיל מלהראות את אחוזי ההצלחה (כפי שהגדרנו אותם קודם) ואת זמן הריצה, כתלות בגודל האוכלוסייה בכל דור. פרט לגודל האוכלוסייה שאר הפרמטרים הושארו קבועים, וכן התוצאות הינן תוצאה של מיצוע על 1000 ריצות. הגרף האלגוריתם הורץ היה *Graph4*.





ניתן לראות כי עליה בגודל האוכלוסייה מביאה מצד אחד לעליה בזמן הריצה במקרה של הצלחה, אך גם מביאה לעליה באחוזי ההצלחה. גודל אוכלוסיה של 30 קיבלנו מעין מצב אופטימלי, שבו אנחנו מעל 99% הצלחה, וזמן הריצה עוד סביר (כ-25 מילישניות). לכן בהמשך הבדיקות נקבע את גודל האוכלוסייה תמיד ל-30. בנוסף נשים לב כי בהשוואה לאלגוריתם *Local Beam Search*, קיבלנו כאן זמן ריצה נמוך ב-2 סדרי גודל (כמה עשרות מילישניות במקום כמה שניות).

כעת נעבור לניתוח של מרכיבי האלגוריתם השונים. נשווה קודם כל בין שתי שיטות הבחירה – שיטת הטורניר ושיטת ה"רולטה". את התוצאות הבאות קיבלנו עבור ממוצע של 6000 הרצות על *Graph4*:

שיטת בחירה	אחוז הצלחה	זמן ריצה ממוצע (מילישניות)
טורניר	98.47%	12.165
רולטה	98.75%	13.695

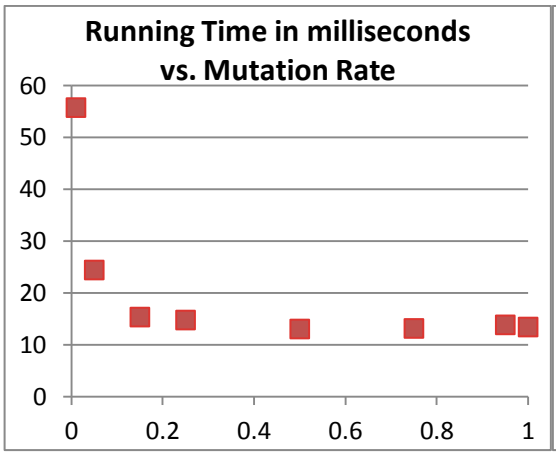
קיבלנו כי אחוזי ההצלחה של שתי שיטות הבחירה כמעט זהים, אך כי יש הפרש של כ-11% בין זמני הריצה (עקב המיצוע על מספר רב מאוד של חזרות, יש לנתון זה מובהקות סטטיסטית). לכן מעתה והלאה, נעדיף להשתמש בשיטת הבחירה של הטורניר.

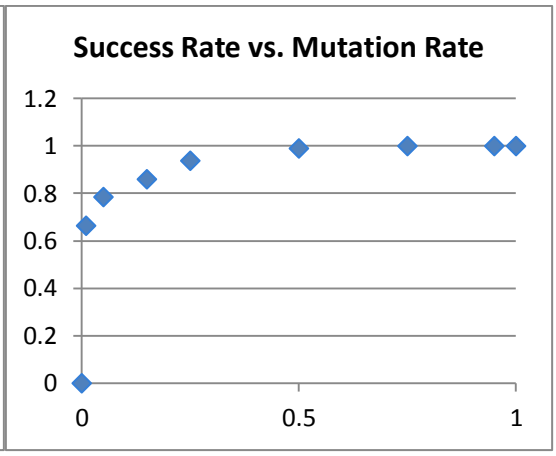
כעת נשווה בין שתי שיטות המוטציות שדיברנו עליהן. השיטה ראשונה הייתה אקראית לחלוטין, ואילו השיטה השנייה יותר "מוכוונת". פירוט של אופן פעולתן של שיטות אלו ניתן למצוא בחלק ההסבר על האלגוריתם הגנטי. נצפה כמובן שאחוזי ההצלחה יהיו גבוהים יותר בשיטה השנייה, ואילו שזמן הריצה יהיה נמוך יותר. את התוצאות הבאות קיבלנו עבור 100 הרצות על שני גרפים שונים:

שיטת מוטציה	גרף	אחוז הצלחה	זמן ריצה ממוצע (מילישניות)
1 - אקראית	*Graph4*	1%	70
	Graph5	47%	48.7
2 - מוכוונת	*Graph4*	99%	15.5
	Graph5	100%	4.1

ניתן לראות כי אכן התוצאות הן לפי הציפיות, וכי בהחלט יש פער משמעותי בשני התחומים בין שיטות המוטציות. נציין שעד כה בבדיקותינו השתמשנו תמיד בשיטה "הטובה יותר" – שיטה 2, וכן נמשיך להשתמש בשיטה זו בשאר ההרצות של האלגוריתם הגנטי שלנו.

לאחר שראינו איזו משיטות המוטציה עדיפה בעינינו, נבחן את ההשפעה של שיעור המוטציות. כלומר, נבחן כיצד ההסתברות להפעיל מוטציה על "ילד" חדש שנוצר מזיווג, משפיעה על אחוזי ההצלחה וזמן הריצה של האלגוריתם. הגרפים הבאים מציגים את אחוזי ההצלחה וזמן הריצה הממוצע כתלות בשיעור המוטציות, והם מתבססים על 1000 הרצות של האלגוריתם עבור כל שיעור מוטציות, על *Graph4*.





ניתן לראות כי אחוז ההצלחה של האלגוריתם במציאת *k*-צביעה מתאימה, עולה בתלילות עם העלאת שיעור המוטציות בהתחלה, אך לאחר מכן הוא מגיע לרוויה. בצורה דומה ניתן לראות כי גם זמן הריצה הממוצע יורד בתלילות עם העלאת שיעור המוטציות בהתחלה, ואח"כ הוא מתייצב על ערך קבוע. משני הגרפי הללו ניתן להסיק שכדי להגיע לתוצאות אופטימליות, עלינו להיות מעל סף מסוים של שיעור מוטציות, במקרה הספציפי שבדקנו – בערך 0.5.

אלמנט שלישי שנותר לנו לבדוק באלגוריתם הגנטי, הוא הזיווג. כעת בחנו את שתי השיטות – שיטת זיווג נקודה-אחת, ושיטת הזיווג האחיד. בבדיקה של פתרון הבעיה עבור מספר גרפים שונים, מצאנו כי **אין הבדל משמעותי בין שתי שיטות הזיווג**.

כעת סיימנו לבדוק מספר אלמנטים מעניינים באלגוריתם הגנטי שלנו, כיצד הם משפיעים על אחוזי ההצלחה בפתרון הבעיה, וכיצד הם משפיעים על זמן הריצה של האלגוריתם. כעת נוכל לתת לאלגוריתם שלנו את הפרמטרים האופטימליים לפעולתו, ולהשוות את אחוזי ההצלחה וזמן הריצה, אל מול האלגוריתמים הקודמים שהצגנו – *Local Beam Search* ו- *Stochastic Beam Search*.

בטבלה הבאה מוצגים תוצאות פעולתם של שלושת האלגוריתמים שבדקנו על חמישה גרפים שונים. בטבלה מוצגים אחוזי ההצלחה של כל אלגוריתם וזמן הריצה הממוצע שלו, לגבי כל אחד מהגרפים. מבחינת שני המדדים, ניתן להבחין ביתרון ברור של האלגוריתם *Stochastic Beam Search* על האלגוריתם *Local Beam Search*, וכן יתרון ברור של האלגוריתם הגנטי על שניהם.

גרף	Local Beam Search		Stochastic Beam Search		אלגוריתם גנטי	
	אחוזי הצלחה	זמן ריצה ממוצע (מילישניות)	אחוזי הצלחה	זמן ריצה ממוצע (מילישניות)	אחוזי הצלחה	זמן ריצה ממוצע (מילישניות)
Graph3	70%	294.92	83.5%	114.28	100%	1.46
Graph4	61%	2445.12	100%	276.92	98.5%	13.47
Graph8	79%	2580.72	100%	1071.15	100%	7.68
Graph9	0%	-	93%	8743.47	99.7%	22.51
Graph10	0%	-	96%	5591.47	72%	1100.49

סיכום:

בפרויקט שלנו עסקנו בפתרון בעיית צביעת הגרפים בכלים של חיפוש מקומי. בעיית צביעת הגרפים היא בעיה שלא מוכר לה פתרון בסיבוכיות זמן פולינומיאלית, והשימוש בחיפוש מקומי מאפשר למצוא לבעיה פתרון במקרים מסוימים (ובמקרים אחרים פתרון "מקורב"), בסיבוכיות זמן טובה, ובסיבוכיות מקום קבועה. כדי להשתמש בחיפוש מקומי, הגדרנו את הייצוג של הבעיה ואת מרחב הפתרונות שלה. השתמשנו בשלושה אלגוריתמים – Local Beam Search, Stochastic Local Beam Search ואלגוריתם גנטי. בדקנו לגבי כל אחד מהם את ההשפעות של פרמטרים מסוים על שיעור ההצלחה של האלגוריתם, וכן זמן הריצה שלו. לגבי האלגוריתם הגנטי השווינו בין שיטות מימוש שונות לכל אחד מהמרכיבים שלו, ומצאנו האם יש הבדלים משמעותיים בין השיטות הללו. לבסוף ערכנו השוואה בין שלושת האלגוריתמים לגבי מספר גרפים. גם מבחינת שיעור ההצלחה וגם מבחינת זמן הריצה, קיבלנו כי האלגוריתם Local Beam Search היה בפער ניכר מהאלגוריתמים האחרים. בנוסף קיבלנו כי האלגוריתם הגנטי מביא לאחוזי הצלחה דומים לאלה של ה-Stochastic Local Beam Search , אך בזמן ריצה נמוך בצורה משמעותית (קיבלנו פער של בין פי 5 לפי 400 בין זמני הריצה של האלגוריתמים). מכאן אפשר להסיק, שבצורות המימוש שאנחנו בחרנו לאלגוריתמים האלו, לאלגוריתם הגנטי יתרון ברור בפתרון בעיית צביעת הגרפים.

נספח – הוראות הפעלה:

ראשית חלץ את קובץ ה - *tar*. כדי להפעיל את התוכנה הכנס את הפקודה:

java –jar GraphColorer.jar

מסך ראשון: עליכם לבחור בין יצירת גרף אקראי או בחירת גרף מתוך רשימה של גרפים מוכנים. **אנחנו ממליצים לבחור גרף מתוך הרשימה**.

יצירת גרף אקראי: עליכם להכניס את מספר הקדקודים בגרף, והצלעות ייבנו באופן אקראי. שימו לב שאם תכניסו מספר הקדקודים יהיה גדול (מעל 100) לתוכנה ייקח זמן רב לפעול.

בחירת גרף מתוך רשימה: רשומים הנתונים על כל גרף. בסוגריים מספר הצבעים המינימאלי שעבורו האלגוריתם מסוגל למצוא צביעה.

אל תסגרו את החלון בו מופיע הגרף שייצרתם!

בחירת האלגוריתם: **אנחנו ממליצים לעבוד את עם האלגוריתם הגנטי**. בכל מקרה, אם בחרתם מתוך הרשימה את אחד הגרפים הגדולים, האלגוריתם של ה- *local beam search* לא יצליח למצוא צביעה.

הגדרת פרמטרים לאלגוריתם: אם בחרתם גרף מתוך רשימה, כדאי להגדיר את מספר הצבעים לפי המספר שהומלץ. אם בניתם גרף משלכם, תצטרכו לשחק עם פרמטר זה עד שתמצאו את הערך המינימאלי עבורו אחד האלגוריתמים מסוגל למצוא צביעה.

אנחנו ממליצים לא לשנות את שני הפרמטרים האחרים. רק אם האלגוריתמים לא מצליחים למצוא צביעה כדאי לנסות ולהגדיל את מספר הדורות המקסימלי.

שימו לב ששלב מציאת צביעה יכול לקחת זמן רב!

הצגת הפתרון: המשחק יציג לכם את התוצאות אליהן הוא הגיע.