דו"ח פרויקט סיום לקורס בינה מלאכותית

בעיית מסלולי האוטובוסים

מגישים : ניר משה (300307824), עקיבא שיף (311617963), רחל הכהן (200679207)

תוכן עניינים

מבוא

כסטודנטים המתגוררים בירושלים, אנו נתקלים באופן יום-יומי בבעיית יעילות קווי האוטובוס בעיר. תלונות על מערך התחבורה, משכי ההמתנה ואי-יעילות מסלולי הנסיעות נשמעות לעתים קרובות ולכן החלטנו לנצל את הידע שצברנו בקורס בינה מלאכותית על-מנת למצוא אלגוריתם שינסה לפתור את הבעיה. לפיכך, הבעיה אותה אנו מעוניינים לפתור בפרויקט זה היא **בעיית מציאת מסלולי אוטובוסים אופטימליים**. בחיים האמיתיים, זו בעיה מורכבת מאוד המושפעת מפרמטרים רבים שמטעמי סיבוכיות, לא לכולם נתייחס במסגרת פרויקט זה אלא נתמודד עם כמה גרסאות פשוטות יותר של הבעיה.

עולם הבעיה UTNDP - Urban Transit Network Design Problem מורכב ביסודו ממספר תתי-בעיות שונות שכל אחת מהם מסובכת לכשעצמה. צריך לקבוע את תדירות האוטובוסים, את לוח הזמנים של כל אחד, לתאם מעברים ואינטראקציות, וכו׳. הבעיה הראשונה מתוך תתי הבעיות הללו נקראת במחקר UTRP - Urban Transit Routing Problem והיא עוסקת בקביעת המסלולים של האוטובוסים עצמם.

בעיה זו היא וריאציה על בעיית VRP שהיא בעצמה הכללה של בעית הסוכן הנוסע, TSP. הוריאציה מתבטאת בכך שישנם אילוצים שונים ל הפתרון וכן המטרה היא שונה. אך כמו VRP גם הבעיה שלנו היא NP קשה.

על אף העבודה הרבה שנעשתה בשנים האחרונות בתחום, מסתבר שרוב המאמרים מתעסקים בסט נתונים ספציפי ויש מעט מאוד ניסיון לשתף נתונים ולשפר אלגוריתמים בהתבסס על עבודות קודמות. ישנם שני יוצאים מן הכלל. הראשון הינו מאמרו של Mandl משנת 1979, שניסח את הבעיה והציג פתרון על גרף בעל 15 קודקודים. סט הנתונים שלו מצוטט בהרבה מאוד מאמרים וכן מחקרים שונים מציעים גישות שונות לפתור את סט הנתונים שלו. המאמר השני הינו של Mumford בשנת 2013 אשר הנגישה סט חדש של ארבעה גרפים בעלי 30+ קודקודים ואשר נמצאים היום גם כן בשימוש נרחב במחקרים אשר יכולים עכשיו להשוות ביצועים תודות לגרפים המשותפים.

את חלק מהעבודה שלנו גם כן נבצע על הגרף של Mandl וכן על שני גרפים של Mumford.

נציג כעת באופן פורמלי את הבעיה איתה אנו מתמודדים:

הצגת הבעיה

נציג את פרטי הבעיה באופן פורמלי:

קלט:

- $G = (V, E)$ - מפת תחנות אוטובוס בעיר והכבישים המחברים ביניהן
- $D: E \to R^+$ - פונקציית המרחק בין כל שתי תחנות המחוברות ביניהן בגרף
- $P = \{(i,j) | i,j \in V\}$ - רשימת נוסעים הנתונים ע״י קודקוד מוצא וקודקוד יעד של כל אחד מהם. בהתייחס לנוסע ספציפי, נכנה את i,j בשמות p.src, p.dst בהתאמה
- N - מספר מסלולי האוטובוס שאנו מעוניינים להגדיר

פלט:

נרצה להחזיר חלוקה אופטימלית (לא זרה) של קודקודי המפה ל-N מסלולים שונים (כל מסלול באורך בין Min ל-Max) כך שלכל נוסע יש דרך (ע״י מסלול אחד או יותר) להגיע ליעדו. אנו מתייחסים למסלול כמסלול דו-כיווני כך שאין חשיבות לסדר הופעת תחנת המוצא והיעד של נוסע. מספיק ששתי התחנות מופיעות באותו המסלול (או במסלולים נחתכים. נרחיב על כך מיד) כדי לקבוע שנוסע הצליח להגיע ליעדו. כמו כן, אנו מאפשרים לנוסע להגיע ליעדו ע״י החלפת מסלול באמצע הדרך. לכן, אם שני מסלולים נפגשים בקודקוד כלשהו בגרף, נאמר שהמסלולים מקושרים ולכן נוסע שעלה בתחנה באחד המסלולים יוכל להגיע גם לכל תחנה בה עובר המסלול השני.

הגדרת אופטימליות:

ישנם שיקולים רבים שיכולים לבוא בחשבון בעת תכנון מערך אוטובוסים. ברור שפתרון אופטימלי עבור החברה המפעילה את קווי האוטובוס לא יהיה פתרון אופטימלי עבור הנוסעים כיון שכל אחד מהם מנסה למזער ערך אחר. לפיכך, נקטנו בגישה הנהוגה במחקר בתחום והיא להפריד את הבעיה לשתי פונקציות מחיר שונות, אחת המחשבת את שיקולי מפעיל האוטובוסים ואחת המחשבת את שיקולי הנוסעים.

- פונקציית המחיר עבור מפעיל האוטובוסים - Operator Cost:

הפעלת מסלולי אוטובוסים ארוכים כרוכה הן בעלויות דלק גבוהות והן בזמן עבודה יקר של הנהגים ולכן פונקציית המחיר אותה חברת האוטובוסים מעוניינת למזער היא סך אורכי המסלולים של הפתרון:

$$C_{operator} = \sum_{a=1}^{N} \sum_{(i,j)\in a} D(i,j)$$

- <u>פונקציית המחיר עבור הנוסעים - Passenger Cost</u>

הנוסעים מעוניינים לצמצם את זמן ההגעה שלהם ליעדם. זמן ההגעה מורכב הן מאורך המסלול אותו עליהם לקחת על-מנת להגיע אליו והן מזמני המתנה במקרה של מעבר בין מסלולים. על-מנת להתחשב בשיקולים אלה, אנו מניחים שכל האוטובסים נוסעים במהירות שווה ולכן צמצום זמן הנסיעה שקול לצמצום אורך המסלול. כמו כן, נטען שזמן ההמתנה לקו אחר אורך זמן קבוע (של 5 דקות, הנחה קצת נאיבית לטעמנו...) ולכן מזעור זמן ההמתנה במעבר לקו אחר שקול למזעור מספר החלפות הקווים באופן כללי:

$$C_{passenger} = \frac{1}{|P|} \sum_{p\in P} \left(\frac{T_{p.src\to p.dst}}{G_{p.src\to p.dst}} \right)$$

כאשר G מייצג את הגרף המקורי ו- $G_{p.src\to p.dst}$ הינו אורך המסלול הכי קצר בגרף עבור נוסע p ואילו T מייצג את גרף התעבורה שבנינו המכיל רק צלעות שנמצאות על מסלולי אוטובוסים וכן מעברים על קודקודים משותפים (באופן טיפוסי לגרף כזה יש יותר קודקודים מהגרף המקורי) ולכן הגודל $T_{p.src\to p.dst}$ הינו אורך המסלול שלוקח לנוסע להגיע על ידי מפת האוטובוסים. הסכום מנורמל על ידי $|P|$ שזהו מספר הנוסעים הכולל ולכן הגודל שנקבל מייצג פי כמה זמן בממוצע לקח לכל הנוסעים להגיע ליעדם ביחס לזמן המינימלי בו הם יכלו לעבור את המסלול.

ההפרדה לשתי פונקציות מחיר שונות מאפשרת לנו בכל ריצה ועבור כל אלגוריתם להגדיר כרצוננו את המשקל שאנו רוצים לתת לכל שיקול. בחלק מהמקרים נרצה להתעלם לגמרי מפונקציה אחת ולהתחשב רק בפונקציה השניה ובמקרים אחרים נרצה להתחשב בשתי פונקציות המחיר ולסכום אותן יחד לפונקציה אחת לפי משקלות שונים שנקבע. בכל חלק מחלקי הפרויקט נפרט מהו המשקל שבחרנו לתת לכל אחת מהפונקציות.

במסגרת הפרויקט, בחרנו לנסות לגשת לבעיה בשתי אסטרטגיות שונות: הראשונה, בניית פתרון באופן הדרגתי ע"י informed search והתקדמות "חכמה" במרחב החיפוש ע"י יוריסטיקות שונות והאלגוריתמים שפיתחנו. בגישה זו אנו מתחילים ממפת עיר ריקה ובכל שלב עושים חישוב למציאת הצעד הבא שיקרב אותנו לעבר הפתרון האופטימלי. האסטרטגיה השניה היתה לפעול למציאת פתרון באמצעות חיפוש לוקאלי ובאופן ספציפי, ע"י שימוש באלגוריתם גנטי. האלגוריתם שמימשנו בחלק זה הוא האלגוריתם המתואר במאמרה של Mumford מ-2013. בגישה זו אנו מתחילים מסט ראשוני של פתרונות אקראיים שלמים ומנסים בכל "דור" להרכיב מהם פתרונות חדשים ולברור מתוך מערך הפתרונות הקיים את הפתרונות הטובים ביותר.
יתרון הגישה הראשונה הוא שהיא מנצלת את כל הידע הקיים בבעיה בעת בניית פתרון. במהלך הריצה אנו כאמור, מתקדמים באופן מחושב יותר במרחב החיפוש ולכן כשאנו מוצאים פתרון, סביר להניח שפתרון זה יהיה טוב במיוחד. עם זאת, חסרון הגישה הזו הוא בזמן הריצה הגבוה שהיא דורשת. במקרה הגרוע, זמן הריצה הוא אקספוננציאלי ולכן אנו לא יכולים להרשות לעצמנו לסמוך על גישה זו בעת התמודדות עם מפת עיר גדולה.
הגישה השניה מציגה את המצב ההפוך מבחינת יתרונות וחסרונות - בגישה זו, אופן ההתקדמות במרחב החיפוש אינו כל כך חכם אך זמן הריצה הוא פולינומיאלי. כאמור, אנו יוצרים באקראי סט של פתרונות מוכנים לבעיה. בנייתם אינה מתבססת על אף מידע הנתון בבעיה ולכן אין לנו שום הבטחה לגבי איכותם. בהמשך אנו יוצרים מתוך פתרונות אלה פתרונות חדשים תוך שימוש בעקרונות האלגוריתמים הגנטיים. גם אופן יצירת הפתרונות החדשים אינו "חכם" במיוחד ואינו מבטיח התקדמות לעבר פתרון אופטימלי. יתרונה

של השיטה הזו נעוץ בזמן הריצה הנמוך יחסית שלה המאפשר לנו לסרוק מספר גדול של פתרונות אפשריים ובכך מגדיל את הסיכוי שאחד מהם יהיה פתרון מספיק טוב.
ניתן להסיק מדברים אלה שדרכי הפעולה בהן נקטנו עבור כל אחת מהגישות שונות מאוד זו מזו. כל גישה דרשה הגדרה מעט שונה של הבעיה והגדרת הנחות מקלות אחרות. בפתרון באמצעות informed search, כיון שהסיבוכיות שלו גבוהה, נאלצנו להניח יותר הנחות מקלות ולפשט יותר את הבעיה על-מנת להצליח להתמודד איתה בכלים החישוביים שגישה זו מאפשרת. בפתרון באמצעות האלגוריתם הגנטי, כיון שהוא אינו מתבסס על המידע הנתון בעת בניית פתרון, נאלצנו להחמיר מעט את דרישות הבעיה על-מנת להבטיח שפתרון שיוגרל יהיה חוקי לכל קלט שיתקבל.

<u>הנחות והפשטות</u>
במהלך הניסיונות השונים למצוא פתרונות לבעיה, הגדרנו עליה כמה הנחות והפשטות על-מנת להקל את ההתמודדות איתה. נפרט להלן כמה הנחות כלליות שהנחנו במסגרת העבודה:

1. איננו מתייחסים כלל לשיקולים של מספר רכבי אוטובוס בכל מסלול, קיבולת מספר נוסעים, תזמון הקווים וכו'. כל השיקולים האלה לא נלקחים בחשבון במסגרת פרויקט הזה.
2. דרישות הנוסעים אינן משתנות עם הזמן. אנו מנסים לבנות מערך מסלולים אשר תומך בדרישות הנוסעים הקיימות לפנינו כרגע ולא צריכים לדאוג לתמיכה בדרישה אחרת בהמשך.
3. ייתכן שיהיו מפות בהן לא ניתן יהיה למצוא פתרון.
4. מפת העיר היא תמיד גרף קשיר.

רק בחלק ב':

- אין קודקודים מנוונים
- בחישוב פונקציית המחיר של הנוסעים, נוסע בוחר את המסלול שלו לפי בחירת המסלול הקצר ביותר
- לא חוזרים על אותו קודקוד פעמיים במסלול

<u>חלק ראשון - בניית פתרון באמצעות **informed search**</u>

כצעד ראשון, החלטנו כאמור לנסות לבנות פתרון באופן מחושב ומתוכנן. אנו מתחילים ממפת עיר ריקה ובונים על גביה מסלולי אוטובוסים תוך התחשבות בנתוני הבעיה הקיימים. כשאנו בונים פתרון באופן כזה, אנו למעשה מתחשבים בשיקולים ובמחיר המוגדרים על הבעיה ומנסים בכל שלב לתכנן מהו המהלך הבא שיספק באופן הטוב ביותר את השיקולים האלה. כיון שבבעיה שלנו יש, כאמור, שתי פונקציות מחיר שונות בהן אנו מעוניינים להתחשב, הרי שמכל אחת מהפונקציות האלה נגזרת דרך בניה שונה. לפיכך, בשלב זה ראינו לנכון להפריד לגמרי את שתי פונקציות המחיר ולמצוא אסטרטגיות בניה בנפרד עבור כל אחת מהפוקנציות. נתאר להלן את הפתרונות שמצאנו עבור כל אחת מהבעיות:

כמו כן, בחלק זה הנחנו כמה הנחות נוספות על הבעיה:

1. ישנם קודקודים "מנוונים" בגרף שלא מהווים תחנת מוצא או יעד עבור אף נוסע.
2. פונקציית המשקל של הצלעות היא המרחק האוקלידי ביניהם ולכן מקיימת אי-שוויון המשולש.

בעיית מפעיל האוטובוסים
בבעיה זו, אם כן, אנו מנסים לבנות פתרון אשר ימזער את פונקציית המחיר של מפעיל האוטובוסים. כזכור, פונקציית מחיר זו היא למעשה סך המרחקים שכל האוטובוסים עוברים בפתרון. לבעיה זו אנו מציעים מספר פתרונות אשר לכל אחד מהם יתרונות אחרים

פתרון ראשון - חיפוש A^*

בשלב ראשון בחרנו להיעזר באלגוריתמי חיפוש מודע (Informed search) באמצעות חיפוש A*. A* הוא אלגוריתם המתאים לפתרון הבעיה שלנו כיון שבאופן שבו היא מוגדרת בשלב זה, שאנחנו בונים את הפתרון מאפס, אנחנו יודעים להבחין בקלות בין פתרון חלקי לבין פתרון שלם ולכן יש לנו את היכולת לזהות שמצב מסוים הוא מצב מטרה (בחלק השני נגדיר באופן שונה מצבים בעולם החיפוש ושם הנחה זו כבר לא תהיה נכונה). כמו כן, קל להגדיר מהם מצבים עוקבים למצב מסויים ולכן יש אפשרות לעבור על פני מרחב החיפוש באופן סדור יחסית. עם זאת, A* מחזיר בדרך כלל את סדרת הפעולות שיש לנקוט על-מנת להגיע למצב המטרה. אנחנו פחות מתעניינים בדרך להשגת הפתרון אלא רק בפתרון עצמו אך זה לא גורע מהתרומה של אלגוריתם זה לפתרון הבעיה שלנו ולכן על אף אי-התאמה קטנה זו, בחרנו להשתמש באלגוריתם זה.

היתרון הברור של אלגוריתם A* הוא שעם יוריסטיקת חיפוש מוצלחת, הוא ימצא לנו את הפתרון האופטימלי לבעיה. כיון שהוא למעשה עושה חיפוש חכם בכל מרחב הפתרונות האפשריים ובכך מהווה מעין brute-force מקוצר, הרי שאם אנו עורכים חיפוש ע"י שימוש ביורסטיקה אדמיסבילית, אזי מובטח לנו שהפתרון שיימצא הוא הפתרון הטוב ביותר. עם זאת, כמובן שאנו משלמים על כך בזמן ריצה. כפי שנראה בהמשך, זמן הריצה של אלגוריתם זה הוא אקספוננציאלי ולכן הוא ישמש אותנו רק עבור מציאת פתרונות בגרפים קטנים.

תיאור האלגוריתם:

כיון ש-A* הוא אלגוריתם המבצע חיפוש מודע, נתחיל מתיאור מרחב החיפוש עליו האלגוריתם רץ:

- State - מצב (קודקוד בעץ החיפוש) הוא N רשימות של מסלולים (מסלול מיוצג ע"י סדרת הקודקודים שהוא עובר).
- Initial state - המצב ההתחלתי של מרחב הבעיה הוא N מסלולים ריקים.
- Successor states - בהינתן מצב s, נגדיר צאצא של s להיות מצב שבו כל המסלולים זהים ל-s פרט למסלול אחד שבו יש קודקוד נוסף במסלול. נדרוש כמובן שבגרף הבעיה קיימת צלע בין שני הקודקודים האחרונים במסלול זה.
- Goal state - נגדיר שנוסע "מסופק" במצב s, אם יש לנוסע איזושהי דרך להגיע מתחנת היעד לתחנת המוצא שלו (כפי שהגדרנו לעיל). נגדיר שמצב s הוא מצב מטרה אם כל הנוסעים סופקו במצב זה.

לפי האלגוריתם, אנו מתחילים ממצב ההתחלה ומשם מתקדמים למצבים העוקבים על-ידי כך שבכל פעם אנו חושפים את הקודקודים העוקבים ומכניסים אותם לתור קדימויות הקובע את סדר הרחבת הקודקודים לפי הפונקציה $f(v) = g(v) + h(v)$. כאשר $g(v)$ היא פונקציית $C_{operator}$ ו-$h(v)$ זו פונקציה יוריסטית שמספקת לנו הערכה לגבי המחיר הצפוי בהמשך הדרך.

כיון שהבעיה מורכבת, אחת ההנחות שהנחנו בחלק זה היא שהגרף עליו אנו עובדים הוא ***גרף מלא***. בהמשך נביא הסבר מדוע הנחה זו לא באמת מגבילה אותנו ותאפשר לנו בהמשך למצוא פתרונות גם עבור גרפים שאינם מלאים. תחת ההנחות האלה, נציע שתי יוריסטיקות:

- יוריסטיקת Min-unvisited-heuristic:

  יוריסטיקה זו מחזירה לכל מצב את הערך הבא:

$$h(s) = \min\{w(v,u) | u \in unvisited(s)\}$$

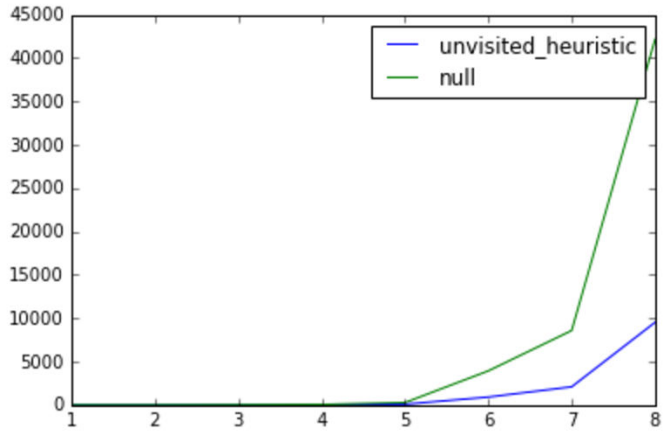
  כאשר $unvisited(s)$ זו קבוצה של קודקודים במפה שעדיין לא הוספנו אותם למסלול המוגדר במצב s.

  קל לראות שיוריסטיקה זו היא אדמיסבילית כיון שהיא מחזירה את הערך המינימלי מבין הערכים האפשריים למצב הבא. כמו כן, יוריסטיקה זו היא גם קונסיסטנטית כיון שאנו מניחים שפונקציית המשקל על הצלעות של המפה תמיד מקיימת את אי-שוויון המשולש. לפיכך, פונקציית ה-cost תמיד תהיה קונסיסטנטית וכיון שהיורסטיקה שלנו תמיד קטנה מה-cost אז גם היא תהיה קונסיסטנטית.

- <u>יוריסטיקת האפס – Null-heuristic</u>:

יוריסטיקה זו מחזירה עבור כל מצב את הערך 0. ראינו בשיעור שפונקציה זו היא קונסיסטנטית. היא תשמש אותנו בעיקר להשוואה מול היוריסטיקה הראשונה.

<u>ביצועים</u>: בתרשים הבא ציר ה-X מציין את מספר הקודקודים בגרף מלא, וציר ה-Y מציין את מספר הקודקודים שהורחבו במהלך החיפוש:



כפי שצפינו מראש, וכפי שעולה מהתוצאות המוצגות בגרף, אלגוריתם A* עם היוריסטיקות השונות, אמנם יעיל במציאת פתרון בגרפים קטנים אך מעבר ל-7-8 קודקודים, זמני הריצה כבר גבוהים מאוד.
לפיכך ניסינו למצוא פתרונות נוספים.

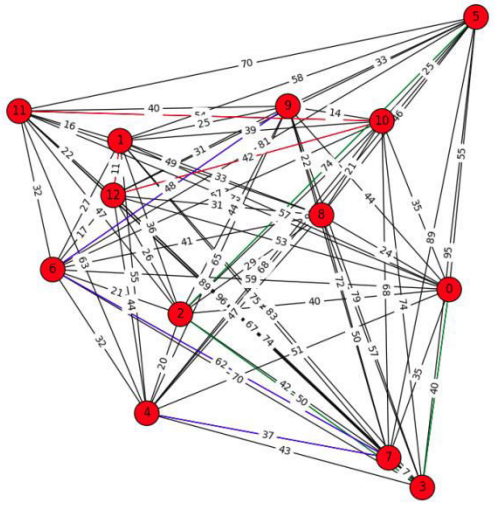
פתרון שני - אלגוריתם חמדני מקורב

בשלב הבא, פיתחנו אלגוריתם חמדני אשר ימצא פתרון מקרב לבעיה.
<u>תיאור האלגוריתם</u>:

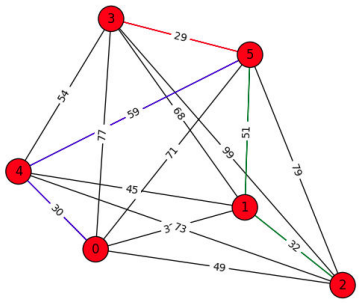
1. $relevant - nodes \leftarrow \emptyset$
2. לכל נוסע p בצע:
   a. $relevant - nodes \leftarrow relevant - nodes \cup \{p.src, p.dst\}$
3. $G' = (relevant - nodes, E)$
4. בחר קודקוד באופן אקראי מהקבוצה $v \leftarrow random(relevant - nodes)$
5. $route \leftarrow \{v\}$
6. לכל קודקוד v ב-$relevant - nodes$:
   a. נגדיר את קבוצת הקודקודים הבאים בתור, שאינם מופיעים במסלול:
   $left = relevnt - nodes \setminus route$
   b. $route \leftarrow route \cup \{(v,u) \mid w(v,u) = min_{(v,u)\in left}\{w(v,u)\}\}$
7. חלק את המסלול ל-N קטעים בעלי מספר תחנות זהה – החזר תוצאה זו.

הסבר: בהתאם להנחות שהנחנו על הבעיה בשלב זה, יתכן שישנם בגרף קודקודים "מנוונים". לכן, האלגוריתם מתחיל ביצירת תת-גרף חדש שמכיל רק את הקודקודים הרלוונטיים, שהם הקודקודים שאינם מנוונים, כלומר שמהווים קודקוד מוצא או יעד עבור לפחות נוסע אחד. לכל קודקוד "רלוונטי", נבחר את הקודקוד הרלוונטי הקרוב אליו ביותר שעוד לא נבחר קודם ונחבר אותם יחד למסלול. כיון שהגרף מלא, כמובן שצעד זה הינו אפשרי. נמשיך באותו אופן עד שניצור מסלול אחד ארוך המכסה את כל הקודקודים הרלוונטיים ואז נחלק אותו ל-N חלקים כמספר המסלולים הנדרש.

להלן שני פתרונות שאלגוריתם זה מצא עבור גרפים מלאים בגדלים שונים. בשניהם מספר המסלולים מוגדר להיות 3 :



3 מסלולים בגרף מלא של 13 קודקודים



3 מסלולים בגרף מלא של 6 קודקודים.

נחשב את סיבוכיות האלגוריתם: קיימת סריקה של כל הנוסעים (הקבוצה P) ע"מ למצוא את קבוצת הקודקודים הרלוונטים וחיפוש מינימום לכל קודקוד. לכן בסה"כ האלגוריתם פועל בסיבוכיות:

$$O\left(\underbrace{|P|}_{Passengers} + \underbrace{|V|^2}_{minimum\ search\ for\ each\ node}\right) \in O(|P| + |V|^2)$$

בהמשך הפרק נראה את התוצאות והביצועים שאלגוריתם זה מראה.

הרחבת הפתרון עבור גרפים לא מלאים:

עבור אלגוריתם A* וגם עבור האלגוריתם החמדן הנחנו הנחה מקלה שהגרף הוא גרף מלא. נראה כעת מדוע הנחה זו היא הגיונית ומאפשרת לנו בכל זאת למצוא פתרונות גם עבור גרפים אחרים. לשם כך, נציג אלגוריתם אשר בהינתן גרף לא מלא G, יחזיר לנו גרף מלא G' שניתן יהיה לעבור בקלות בין פתרונות ב-G' ל-G. כלומר, מתוך פתרון ב-G' ניתן יהיה לחלץ פתרון מ-G כך שהמחיר שלהם יהיה שווה. לכל גרף, נוכל להריץ את שני האלגוריתמים שהצגנו על הגרף השקול לו G' ולחלץ מכך את הפתרון עבור הגרף שלנו ולכן צמצום הבעיה עבור גרפים מלאים לא באמת מגבילה אותנו.

הרעיון של האלגוריתם הוא למצוא את כל המסלולים הקצרים ביותר בגרף באמצעות Floyd-Warshall ואז לצמצם את הגרף רק לקודקודים הרלוונטיים כאשר בין כל שני קודקודים רלוונטיים נמתח צלע שמשקלה הוא כמשקל המסלול הקצר ביותר ביניהם וכן נשמור מסלול זה לטובת פעולת חילוץ הפתרון.

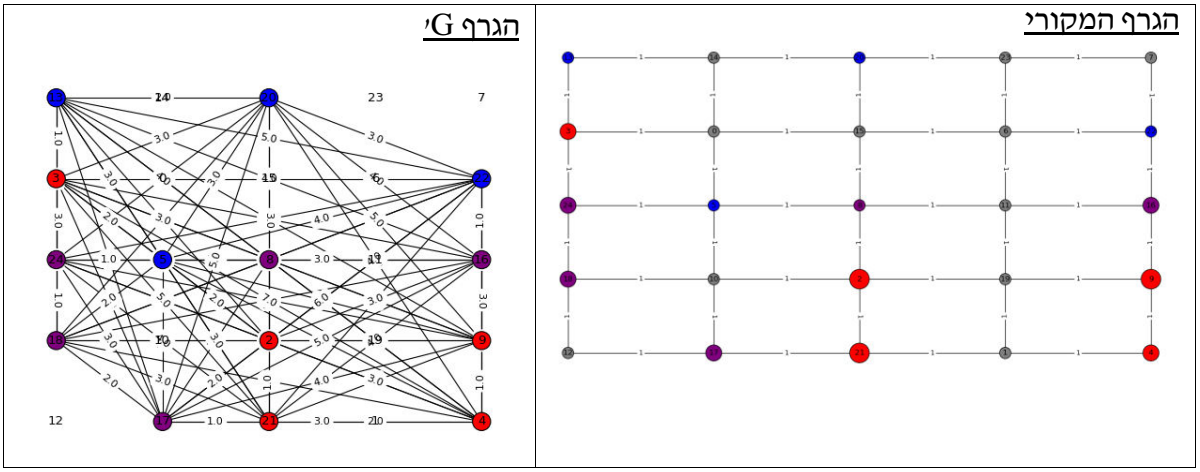
<u>תיאור האלגוריתם:</u>

1. $relevant - nodes \leftarrow \emptyset$
2. חשב את כל המסלולים הקצרים במפה: $shorts \leftarrow Floyd\_Warshall(G)$
3. לכל נוסע p בצע:
   a. $relevant - nodes \leftarrow relevant - nodes \cup \{p.src, p.dst\}$
4. $E' = \{(u,v)\ |u,v \in relevant - nodes\}$

5. $G' = (relevant - nodes, E')$
6. הגדר פונקציית משקל באופן הבא:
   a. $w: V \times V \to \mathbb{R}$
   b. $w(u,v) = shorts(u,v)$
7. בנוסף, כל צלע תשמור שדה נוסף שהוא המסלול עצמו: $(u,v).path = shorts(u,v)$

אלגוריתם זה, רץ בזמן פולינומיאלי $O(|V|^3)$ והוא, כאמור מאפשר לנו להריץ את שני האלגוריתמים שהצגנו – A* והאלגוריתם החמדן – על כל צורה של גרף ולא רק על גרפים מלאים. על-מנת להריץ את האלגוריתמים על גרף לא מלא כלשהו G, פשוט נפעיל את האלגוריתם שתיארנו לעיל על G ונמצא גרף שקול לו G'. נריץ את האלגוריתם הרצוי על G'. הרצת האלגוריתם על G' תחזיר לנו פתרון כלשהו עבור G'. נחלץ מפתרון זה את הפתרון עבור G באופן הבא: לכל מסלול A בפתרון של G', נחליף כל צלע $(u,v) \in A$ ברשימת הצלעות $(u,v).path$.

מצורפת דוגמא להפעלת האלגוריתם על גרף בצורת Grid.



פתרון שלישי - מציאת פתרון מקורב באמצעות אלגוריתם HUB

לבסוף, נציע אלגוריתם נוסף שפיתחנו למציאת פתרון מקרב לבעיה. אלגוריתם זה מתבסס על חיפוש A* שתיארנו בתחילת חלק זה אשר הראינו שהוא פועל טוב עבור גרפים קטנים. כמו כן, הוא מנצל את התכונות הייחודיות של הבעיה ובונה מראש פתרון בצורה שבסבירות גבוהה תתאים לה. הרעיון לאלגוריתם זה נולד מתוך חשיבה על האופן בו קוויי האוטובוס פועלים במציאות. במציאות, וכן בעולם הגרפים, אנו מכירים את הרעיון של Hub שהוא מוקד מרכזי בגרף/מפה שמהווה מעין נקודה מרכזת לשאר המסלולים. נתאר להלן את עיקרי האלגוריתם:

ראשית, נציין שאלגוריתם זה מתאים לבעיה בה אנו נדרשים למצוא יותר ממסלול אוטובוס אחד. לכן נגדיר N>1 (כאשר N הוא מספר המסלולים המבוקש). הפתרון הזה יתבסס על ההנחה שנוסעים יכולים להחליף מסלולים בדרך. הפתרון שנציע יבטיח שכל נוסע יחליף לכל היותר 3 אוטובוסים.

נחלק את המפה ל-$N-1$ מקבצים (clusters) של קודקודים לפי מידת הקרבה ביניהם באמצעות אלגוריתם k-means. לכל מקבץ, נמצא את הקודקוד הקרוב ביותר ל"מרכז העיר" (המוגדר כנקודת מרכז הטווח עליו מתפרשת מפת הגרף) ונגדיר אותו כקודקוד יציאה. כעת, ניצור מכל מקבץ תת-גרף המכיל רק את הקודקודים הרלוונטיים ונריץ על גרף זה אלגוריתם A* שתיארנו לעיל. A* יחזיר לנו מסלול אחד לכל מקבץ.

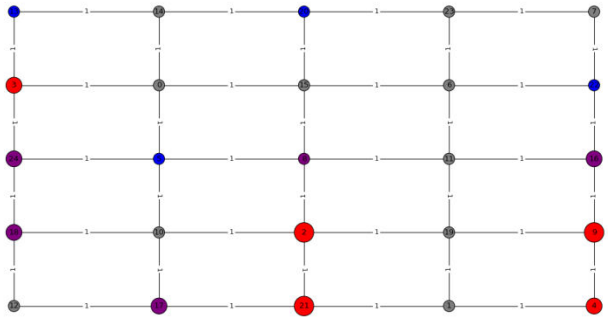
לבסוף, נגדיר את מסלול "מרכז העיר" להיות מסלול העובר בין קודקודי היציאה של כל המקבצים.

<u>תיאור האלגוריתם $(HUB\ ALGO)$</u>:

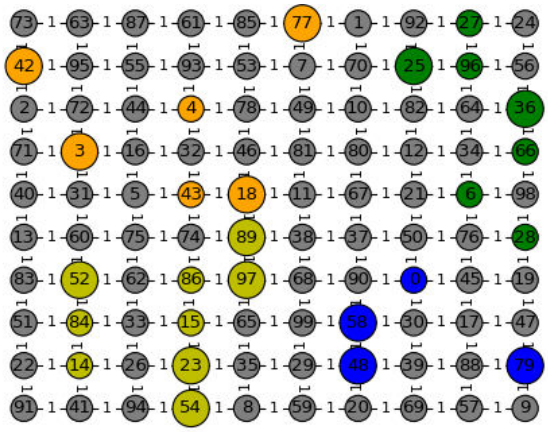
1. $relevant - nodes \leftarrow calculate\ relevant\ nodes(G)$
2. $all_{short} \leftarrow calculate\ all\ shortest\ path\ using\ dijekstra(G)$
3. $create\ N-1\ clusters\ using\ K-MEANS(relevant-nodes)$
4. לכל cluster בצע:
   a. צור גרף מלא מכל cluster:
      i. הגדר פונקציית משקל באופן הבא:
      ii. $w: V \times V \rightarrow \mathbb{R}$
      iii. $w(u,v) = all_{shorts}(u,v)$
   b. חשב מסלול המילטוני מינימאלי על הגרף החדש באמצעות אלגוריתם A*.
   c. קבע את המסלול עבור cluster זה.
5. חשב לכל cluster את הנקודה הקרובה ביותר למרכז הגרף.
6. הקצה קו אוטובוס נוסף המקשר בין כל הנקודות המרכזיות במסלול מינימלי.

נדגים את אופן הריצה של האלגוריתם על גרף Grid:

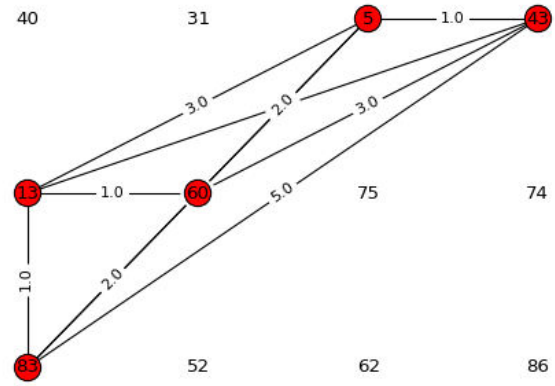
<u>שלב ראשון</u>: זהה קודקודים מעניינים בגרף וחשב מסלולים קצרים בין כל הקודקודים.



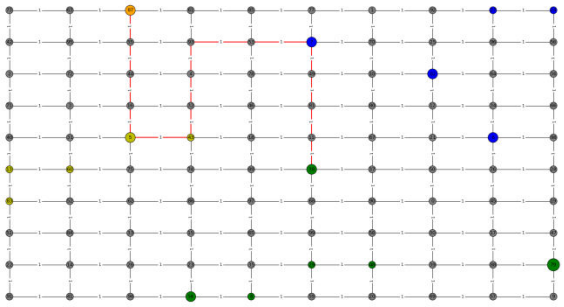
<u>שלב שני</u>: הפעל את $k-means$ עבור הקודקודים הרלוונטים. מצא $N-1$ clusters.



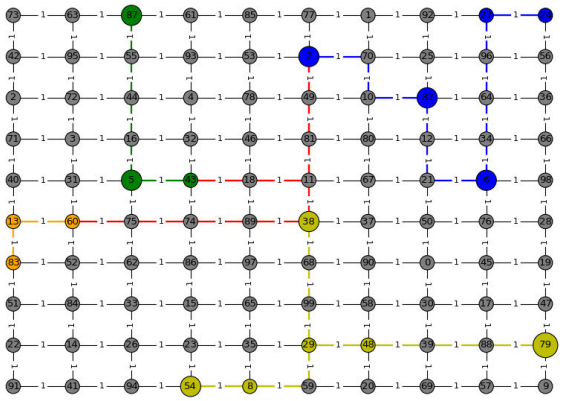
<u>שלב שלישי</u>: לכל cluster חשב גרף מלא. כך שמחיר כל קשת הוא המסלול הקצר ביותר בין אותם קודקודים. מצא בגרף זה מסלול המילטוני:



<u>שלב רביעי</u>: לכל cluster חשב "נקודה מרכזית" – הנקודה הקרובה ביותר למרכז. הקצה קו אוטובוס העובר במסלול המינימאלי בין כל אותן "נקודות מרכזיות":

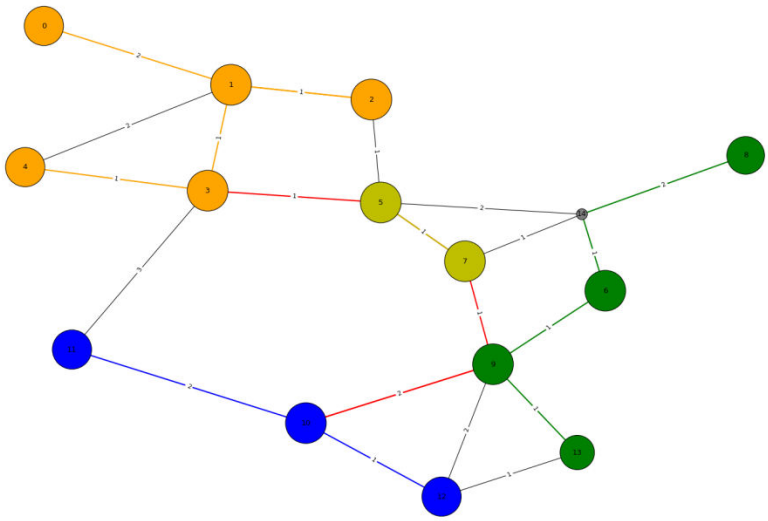


<u>שלב חמישי</u>: חשב מסלול אופטימאלי בכל cluster
באמצעות A^*:



נשים לב שאלגוריתם זה תמיד יחזיר גרף קשיר (בין הקודקודים הרלוונטיים) ובכך יבטיח שתמיד הפתרון המוחזר הוא חוקי.

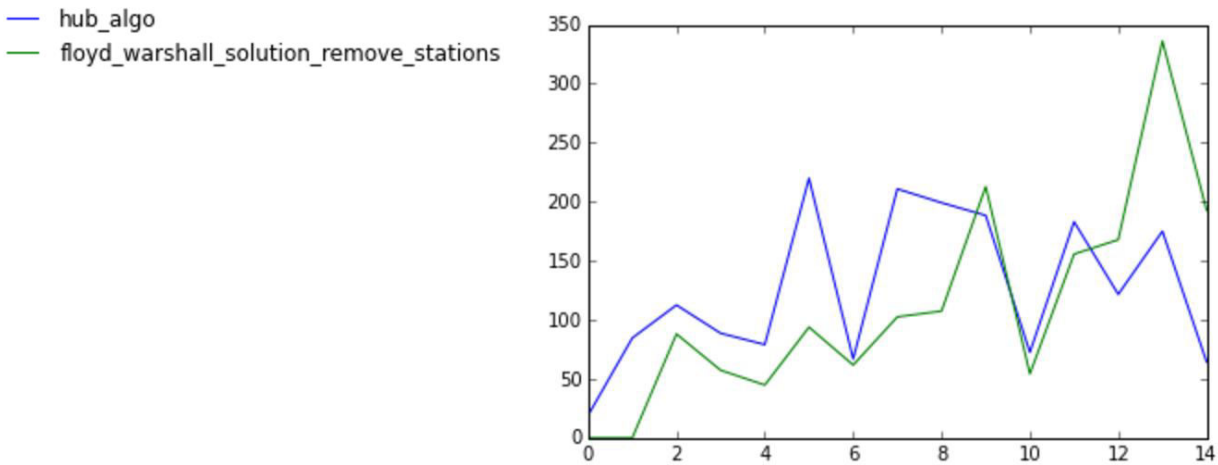
<u>דוגמא לפתרון על מפת Mandl (15 קודקודים)</u>:



השוואת ביצועים בין האלגוריתמים

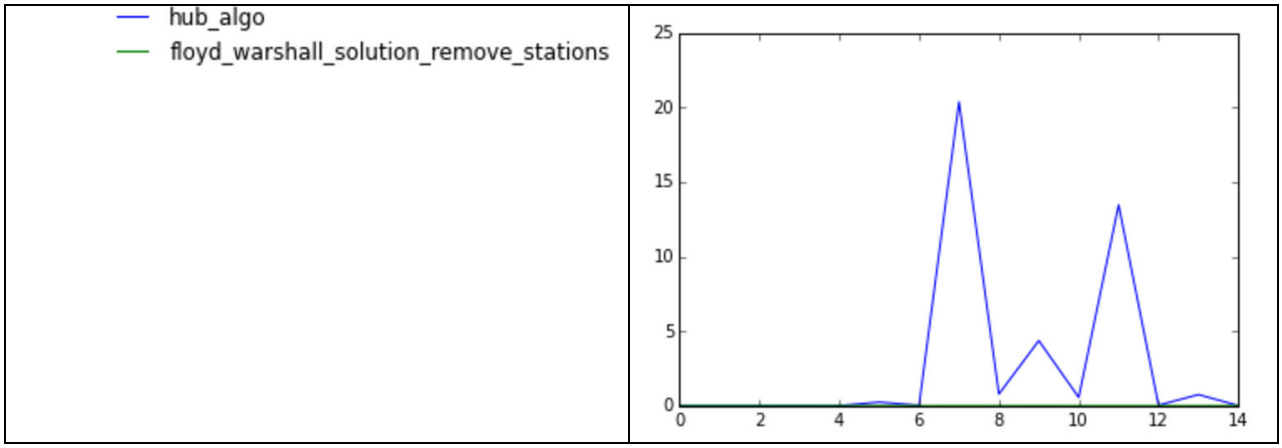
הצגנו שני אלגוריתמי קירוב, האחד חמדן הפועל בגרפים מלאים בלבד ואלגוריתם HUB הפועל בכל סוגי הגרפים. לכן לצורכי השוואה, מדדנו את תוצאות האלגוריתם על גרפים מלאים.

<u>השוואה ראשונה</u>: מדידת טיב הפתרון ($C_{operator}$) כפונקציה של מספר הקודקודים ה"מעניינים"



ניתן לראות כי אין אלגוריתם שהביצועים שלו תמיד טובים יותר מהאחר.

<u>השוואה שנייה</u>: מדידת זמני הריצה – גרף זה מציג את זמן הריצה של האלגוריתם כפונקציה של מספר הקודקודים המעניינים בגרף.



ניתן לראות כי האלגוריתם החמדן משמעותית הרבה יותר מהיר מאלגוריתם HUB. זאת כיון שאלגוריתם HUB מתבסס על חיפוש A*

בעיית נוסעי האוטובוסים

בחלק זה, אנו מנסים לבנות פתרון מאפס אשר ממזער את פונקצית המחיר של הנוסעים $C_{passenger}$. זוהי פונקציה הרבה יותר מורכבת שההתמודדות איתה היתה הרבה יותר קשה. נקדים כבר ונאמר שלא הצלחנו למצוא אלגוריתם טוב עבורה בחלק זה. הקושי העיקרי בבעיה טמון בכך שלצורך חישוב פונקציית המחיר של הנוסעים, יש צורך בראיה רחבה של מערך האוטובוסים כולו וכשאנו בונים את הפתרון מאפס, הרבה מהמידע הרלוונטי חסר לנו ואין לנו דרך טובה לבצע הערכה שתנבא באופן מדויק את המשך התפתחות הפתרון אותו אנו בונים. בכל זאת, נראה את הנסיונות שהיו לנו לפתור את הבעיה באמצעות חיפוש A*

מרחב החיפוש

מרחב החיפוש בבעיה זו דומה מאוד למרחב החיפוש אותו הגדרנו בבעיה הקודמת, למעט שינוי אחד בהגדרת המצבים.

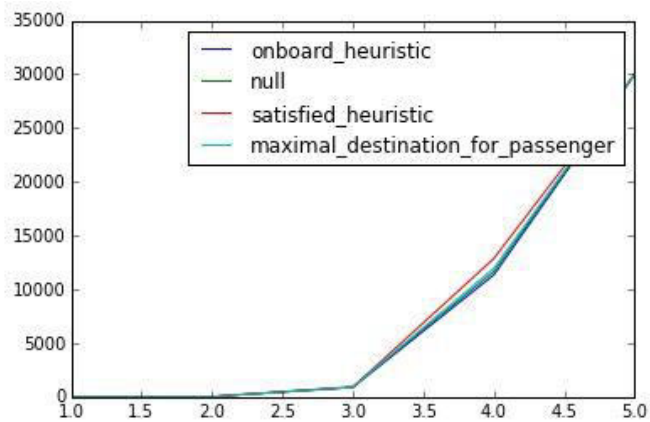
נשים לב כי בשונה מהבעיה הקודמת, בה נמדד המרחק בלבד, בבעיה זו, סדר המעבר על הקודקודים קריטי ומשפיע מאוד על טיב הפתרון. לפיכך, ראינו לנכון בשלב זה להגדיר מצבים במרחב החיפוש רק כמסלולים שלמים מקודקוד מעניין אחד לאחר. הסתכלות על מסלול חלקי מקודקוד מעניין לקודקוד לא מעניין לא מספקת לנו שום ערך אינפורמטיבי לגבי טיב הפתרון שנשיג אם נלך בכיוון זה. ערך פונקצית המחיר של הנוסעים הוא אינפורמטיבי רק בהינתן שני קודקודים רלוונטיים ולכן כך הגדרנו את המצבים במרחב החיפוש.

נציג להלן כמה פונקציות יוריסטיות שפיתחנו בניסיון למצוא אלגוריתם שיימצא את הפתרון האופטימלי. בכל היוריסטיקות אנו עוקבים אחרי מספר הנוסעים שסופקו (קיים מסלול מהמוצא ליעד שלהם) ואחרי מספר הנוסעים שעלו על אוטובוס (תחת ההנחה שברגע שבנינו מסלול שעובר בקודקוד המוצא שלהם, אז הם עולים עליו):

1. Satisfied heuristic – פונקציה זו נתוננה ע"י המשוואה הבאה:
$$h(v) = |P| - (|satisfied(V)| + |onboard(v))$$
2. $h(v) = |P| - |onboard(v)$ - Onboard heuristic
3. $h(v) = \max_{p \in satisfied(v)^C}\{opt(p)\}$ – Maximal destination for passenger heuristic

   כאשר opt(p) הוא אורך המסלול הקצר ביותר הקיים בגרף המפה המקורית, בלי התחשבות במערך המסלולים. כלומר, פונקציה זו מחזירה את המרחק האופטימלי המקסימלי של הנוסעים שעוד לא הגיעו ליעדם.
4. $h(v) = 0$ - Null heuristic

להלן השואת הביצועים בין 4 היוריסטיקות האלה. גרף זה מציג את זמן הריצה כפונקציה ל מספר הקודקודים בגרף:



ניתן לראות שכל היוריסטיקות הגיעו לתוצאות כמעט זהות. לא הצלחנו למצוא יוריסטיקה שתהיה משמעותית טובה יותר מהנתונים המוצגים כאן. ניתן לראות שזמני הריצה עולים בצורה אקספוננציאלית וכבר בגרף של 5 קודקודים הם מתקשים מאוד למצוא פתרון.

חלק שני - חיפוש לוקאלי באמצעות אלגוריתם גנטי

בחלק זה של הפרויקט, עברנו לגרפים גדולים יותר ובלי הנחות מוקדמות על הטופולוגיה של הגרף. לאור הקשיים (הצפויים) בהם נתקלנו בחלק הראשון, בחרנו לנקוט כעת באסטרטגיה של חיפוש לוקאלי. חיפוש כזה מתאים יותר לבעיה שלנו כיון שמרחב החיפוש הוא עצום ואין דרך פשוטה להכריע עד כמה פתרון קרוב לפתרון האופטימלי.
בנוסף, בעוד בחלק הראשון ניסינו לבנות מסלול מאפס בצורה חכמה, כאן מרחב החיפוש שלנו מכיל רק פתרונות מלאים לבעיה. כלומר, כל מצב במרחב החיפוש שלנו כעת הוא סט מסלולים שלם וחוקי. לפיכך, במקרה זה, אין לנו דרך להכריע שמצב מסויים הוא מצב מטרה כפי שעשינו בחלק הקודם אלא רק לתת לכל מצב ציון ולכן אנו עורכים חיפוש לוקאלי בין פתרונות אפשריים שונים לבעיה ובוחרים מביניהם את האופטימלי שמצאנו.

מימשנו בחלק זה של הפרויקט את האלגוריתם המוצג במאמרה של Mumford אותו נתאר להלן:
ראשית, כיון שיצירת הפתרונות באלגוריתם זה נעשית באופן כמעט אקראי, הגדרנו כמה תנאים שפתרון צריך לעמוד בו על-מנת להבטיח שהוא פתרון חוקי:

1. אורך כל המסלולים בפתרון הוא בין ערכים קבועים מראש - Min ו-Max
2. כל הקודקודים של גרף הבעיה מכוסים - הנחה זו מתבססת על ההנחה שאין קודקודים מנוונים בגרף הבעיה.
3. כל המסלולים מהווים גרף קשיר - כלומר, נקודות החיתוך בין מסלולים דואגות לכך שתמיד תהיה איזושהי דרך לעבור בין שני מסלולים שונים בגרף. הנחה זו מבטיחה שכל נוסע יכול להגיע ליעדו (אולי ע״י החלפה של קווים) ומאפשרת לנו מדד נח וזול יותר לאימות חוקיות הפתרון.

פונקציות המחיר בחלק זה של הבעיה זהות לפונקציות המחיר שהגדרנו בחלק הראשון. גם כאן אנו מנסים למצוא שני פתרונות אופטימליים, אחד לכל פונקציית מחיר. בשונה מבחלק הראשון, פונקציות המחיר השונות לא משפיעות כל כך על אופן ריצת האלגוריתם ולכן מציאת שני הפתרונות האלה מתרחשת במקביל.

האלגוריתם:

```
Generate initial population of feasible route sets
Calculate passenger and operator costs for each route set
Record the best-routeset-so-far for each objective
repeat
    for each individual in the population do
        This individual is Parent1
        Select a second individual at random (Parent2)
        Offspring ← Crossover(Parent1,Parent2)
        Repair(Offspring)
        Apply mutation(Offspring)
        if Offspring is a duplicate then
            Delete offspring
        else if Offspring dominates either parent then
            Offspring replaces the dominated parent, testing Parent 1 first,
            then Parent 2
        else if Offspring dominates either vector containing the best-so-far
        objective or Offspring improves on a best-so-far objective then
            Offspring replaces a parent, ensuring the other best-so-far objec-
            tive is not lost
        else if Offspring and parent(s) are mutually non-dominating then
            Find an individual in the population that is dominated by the
            Offspring and replace it with the Offspring
        else
            Delete Offspring
until the stopping condition is satisfied
print  All non-dominated solutions in the final population.
```

האלגוריתם מתחיל ביצירת מספר רב של פתרונות אקראיים המרכיבים יחד את "אוכלוסיית הדור הראשון". אנו שומרים עבור כל פתרון את וה-operator cost שלו ואת שני הפתרונות הטובים ביותר בדור זה (אחד לכל פונקציית מחיר). לאחר מכן, האלגוריתם מתקדם באופן איטרטיבי על פני מספר מוגדר מראש של "דורות" כאשר בתוך כל דור אנו יוצרים פתרונות חדשים באופן הבא:

בוחרים באקראי שני פתרונות "הורים" מהאוכלוסיה ומרכיבים מהם פתרון "בן" ע"י פונקציית cross-over שנפרט את פעולתה בהמשך. פתרון הבן מכיל כחצי מהמסלולים של הורה אחד וכחצי מהמסלולים של ההורה השני. בשלב הבא של יצירתו, פתרון הבן עובר מוטציה ע"י פונקציית mutation שגם עליה נרחיב מיד. במהלך כל תהליך יצירת הבן, נעשות פעולות תיקון ובקרה שמוודאות שהפתרון שנוצר הוא פתרון חוקי העומד בתנאים שהגדרנו מראש.

לאחר ייצורו של הבן, אנו מחשבים את ערך ה-passenger cost ו-operator cost שלו ולפי השוואת ערכים אלה לשאר חברי האוכלוסיה הנוכחית, אנו מחליטים האם הוא "ישרוד" ויצטרף לאוכלוסיה (על חשבון פרט אחר) או שייכחד. כך אנו ממשיכים למשך מספר קבוע מראש של דורות ובסוף הריצה אנו מחזירים את שני הפתרונות הטובים ביותר של הדור האחרון.

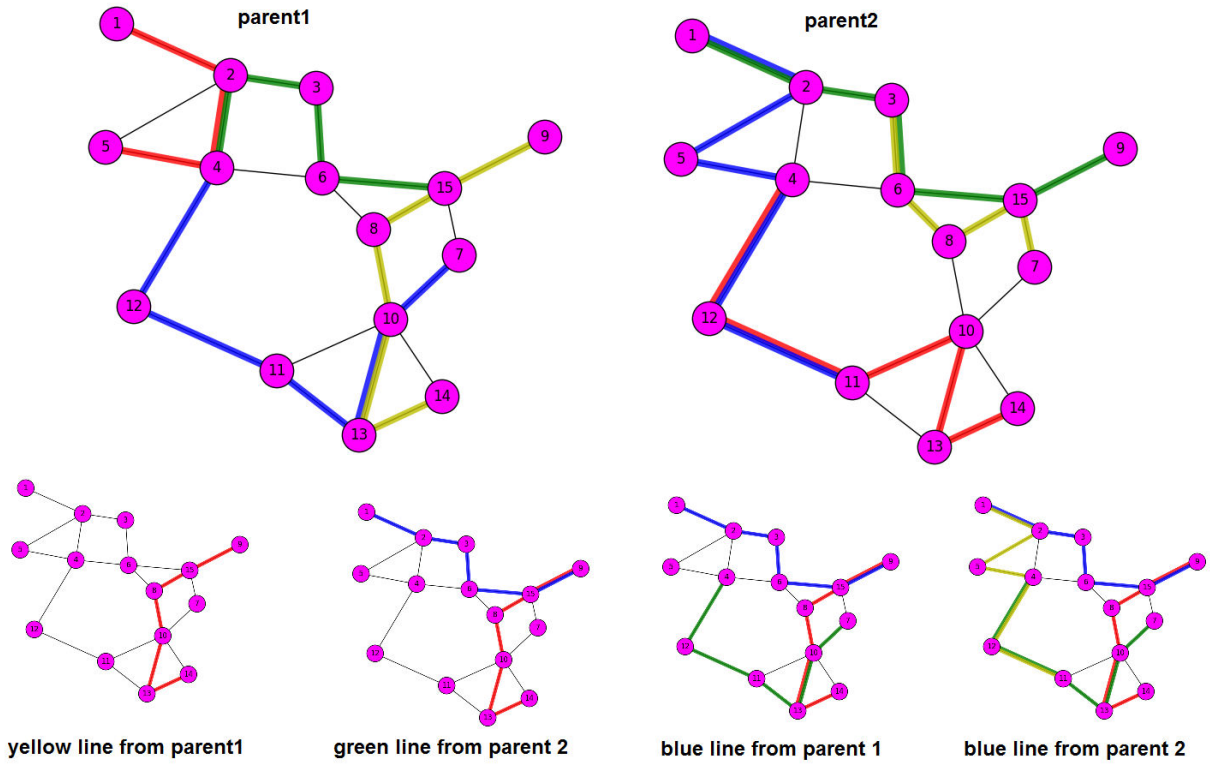
פונקציית cross-over

בניית פתרון הבן מתוך פתרונות ההורים נעשית באופן הבא:

בוחרים מסלול כלשהו מההורה הראשון ואז מההורה השני, שוב מהראשון וכן הלאה עד שמגיעים למספר המסלולים הדרוש. הבחירה של המסלול הראשון היא אקראית ולאחר מכן, כל מסלול נבחר לפי השיקולים הבאים: על-מנת לשמור על קישוריות הגרף, אנו מראש נבחר מההורה הנוכחי רק מסלולים שיש להם כבר איזשהו חיתוך עם אחד המסלולים הקיימים בבן. מבין מסלולים אלה, אנו נבחר את המסלול שמספר הקודקודים החדשים שהוא מוסיף לנו הוא הכי גדול ביחס לאורכו.

ההיגיון ביצירת בנים באופן הזה היא שאנו משתמשים במסלולים שכבר לכאורה "הוכיחו את עצמם" במבחן ההישרדות מהדורות הקודמים ובנוסף, בניית פתרון באופן הזה מבטיחה לנו התפרשות גדולה יחסית של

המסלולים על פני הגרף. התפרשות כזו טובה הן עבור הנוסעים והן עבור המפעיל ולכן יש סבירות גבוהה שהיא תמזער את שתי פונקציות המחיר. להלן הדגמה בו זוג "הורים" מרכיבים פתרון (משמאל לימין):



פונקציית mutation

כל פרט, לאחר ייצורו, עובר תהליך מוטטיבי של "עיוות" הפתרון המקורי כפי שהוא נוצר וזאת על-מנת להוסיף מימד של אקראיות למערכת ובכך להתפרש טוב יותר על פני מרחב החיפוש. המוטציה שהפעלנו כאן היא מחיקה או הוספה אקראית של קודקודים למסלולים. בחצי מהמקרים הפונקציה בוחרת לקצץ את המסלולים של הפתרון. היא מקצצת קודקודים מקצוות של מסלולים תוך כדי וידוא שקישוריות וכיסוי הגרף נשמרים. בחצי האחר של המקרים, היא מנסה להאריך מסלולים ע"י הוספת קודקודים בקצותיהם. שוב, תוך וידוא שהתנאים המבטיחים את חוקיות הפתרון נשמרים.

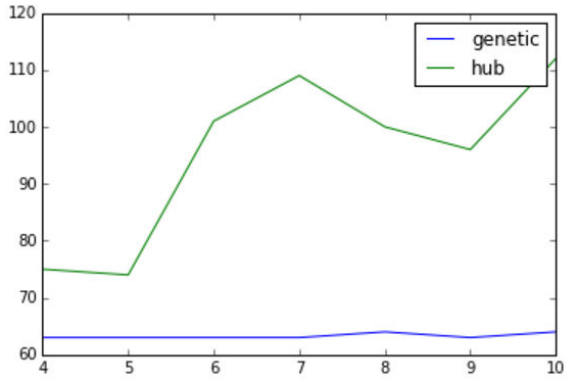
תוצאות:

<u>דיון</u> – החסם העיקרי שלנו היה זמן הריצה של האלגוריתם. על מנת לחשב ציון לכל מפת מסלולים נצטרכנו לחשב את משך זמן הנסיעה של כל נוסע לייעדו. בשביל זה, יצרנו גרף "תעבורה" המכיל רק את הצלעות בהם יש לפחות מסלול אחד וכן קודקודים משותפים למסלולים פוצלו לקודקודים שונים עם צלע של "מעבר" (המתנה 5 דקות) ביניהם. הגרף שיוצא הוא גדול בהרבה מהגרף המקורי שכן כל הקודקודים בהם יש יותר ממסלול אחד שוכפלו וכן נוספו צלעות ביניהם. בשלב הזה יכלנו להריץ אלגוריתם all pairs shortest path למציאת המסלול הכי קצר עבור כל הנוסעים. האלגוריתם הינו של Floyd Warshall אבל ראינו משיקולי יעילות שעל גרפים קטנים יותר מכ-50 קודקודים (ותלוי במספר הקווים) עדיף היה להפעיל Dijkstra על כל הקודקודים שזמן הריצה שלו הינו $O(\mathrm{EV} + \mathrm{V}^2 \log V)$ לעומת Floyd Warshall שהוא $O(V^3)$. היה ריאלי להריץ את התוכנה שלנו רק עד הגרף השני (כולל) של Mumford המכיל 70 קודקודים. מעבר לזה הקוד היה איטי מדי.

חלק שלישי - השוואה ומסקנות

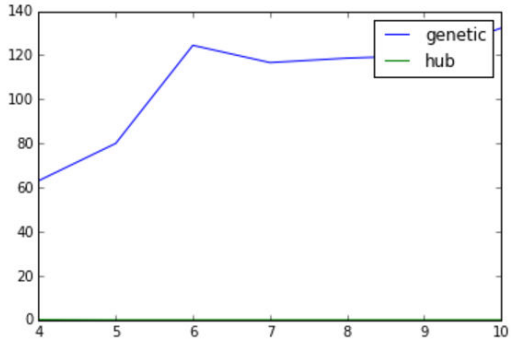
לסיום, השווינו בין ביצועי האלגוריתם הגנטי לבין ביצועי אגלוריתם HUB. כיון ש-HUB מתוכנן במטרה למזער את פונקצית המחיר של מפעיל האוטובוסים ואינו מתחשב כלל בפונקציית המחיר של הנוסעים, השווינו בין שני האלגוריתמים רק במדד של $C_{operator}$. הרצנו את שני האלגוריתמים על הגרף של Mandl המכיל 15 קודקודים.

גרף 1: גרף זה מציג את $C_{operator}$ כפונקציה של מספר קווי האוטובוס. ניתן לראות שהפתרון שאלגוריתם HUB מוצא הולך ומתרחק מהפתרון האופטימלי ככל שאנו עולים במספר הקווים בעוד האלגוריתם הגנטי נשאר עקבי ושואף לפתרון אופטימלי עבור כל מספר מסלולים. הסיבה לכך היא שככל שמספר המסלולים עולה, כך ה-clusters של אלגוריתם HUB יותר רחוקים זה מזה ולכן המסלול המרכזי המחבר בין כולם הינו ארוך יותר וזה מה שמייקר את הפתרון.



לעומת זאת, בהשוואת זמני ריצה ניתן לראות מגמה הפוכה:

גרף 2: גרף זה, כאמור מציג את זמן הריצה של כל אלגוריתם כפונקציה של מספר הקווים. כאן אנו רואים יתרון משמעותי לאלגוריתם HUB. למעשה, בעוד באלגוריתם הגנטי זמן הריצה עולה ככל שמגדילים את מספר המסלולים, באלגוריתם HUB, כיון שהחלק הכבד בו הוא מציאת מסלול בתוך cluster ע"י A*, הרי שככל שה-clusters יותר קטנים, כך זמן הריצה יורד.



לסיכום, ראינו שלכל אלגוריתם ישנם יתרונות וחסרונות שונים. עלות החישוב של מציאת מסלול אופטימלי היא יקרה ומורכבת ולכן הראינו כמה ניסיונות למצוא אלגוריתמים המוצאים פתרון מקרב בזמן קצר יותר.

ביבליוגרפיה:

- Mumford, Christine L. "New heuristic and evolutionary operators for the multi-objective urban transit routing problem." *Evolutionary Computation (CEC), 2013 IEEE Congress on*. IEEE, 2013. http://ieeexplore.ieee.org/xpls/icp.jsp?arnumber=6557668