תרגיל בית 1: שימוש באלגוריתמי חיפוש היוריסטיים לתכנון מסלולי חלוקה אופטימליים

מגישות:

לישר כהן 302376785

טניה דרובצ'נקו 320932544

מטרות התרגיל

- נתמודד עם בעיות פרקטיות ותיאורטיות של חיפוש במרחבי מצבים עצומים.
- נתרגל את הנלמד בהרצאות ובתרגולים.
- נתנסה בתכנות ב- python לפתרון בעיות פרקטיות.

הנחיות כלליות

- **תאריך הגשה:** יום שני, 18.05.2020, בשעה 23:59.
- את המטלה יש להגיש **בזוגות בלבד**.
- יש להגיש מטלות מוקלדות בלבד. פתרונות בכתב יד לא ייבדקו.
- ניתן לשלוח שאלות בנוגע לתרגיל לתיבת המייל הקורסית: ai.technion@gmail.com. אנו מבקשים לא לשלוח הודעות בנוגע לתרגיל לתיבות הדואר של הסגל. לפני שליחת שאלה, בדקו האם קיימת לה תשובה כבר ב-FAQ. נציין כי שאלות שנענו כבר ב- FAQ לא יענו שוב במייל.
- המתרגל האחראי על תרגיל זה: אלעד נחמיאס.

חלק א' – מבוא והנחיות (2.5 נק' יבש)

במטלה זו נעסוק בהפעלת אלגוריתמי חיפוש על מרחבי מצבים גדולים במיוחד לבעיות ניווט. מומלץ לחזור על שקפי ההרצאות והתרגולים הרלוונטיים לפני תחילת העבודה על התרגיל.

במהלך התרגיל תתבקשו להריץ מספר ניסויים ולדווח על תוצאותיהם. אתם נדרשים לבצע ניתוח של התוצאות, כפי שיוסבר בהמשך.

מוטיבציה

ברקע התפרצות נגיף הקורונה בישראל, מד"א עובדים סביב השעון בביצוע בדיקות לאבחון הוירוס. מד"א מגיעים לביתו של כל מי שמדווח על תסמינים ובודקים אותו ואת כל הדיירים המתגוררים ביחד איתו. במקביל ללימודיו בטכניון, מוטי מתנדב במד"א והינו בעל הכשרה לנהג אמבולנס. בתחילת המשמרת מוטי מקבל רשימה של כל הבדיקות שיש לבצע ומיד יוצא לדרך.

באמבולנס יש מקרר מיוחד שבו ניתן לשמור את כל הבדיקות שנלקחו עד כה. המקום במקרר מוגבל, וכאשר מתמלא מוטי צריך לעבור באחת מהמעבדות האזוריות כדי להעביר להם את הבדיקות ולפנות מקום במקרר. בנוסף, עקב המחסור במטושים, מספר המטושים הזמינים (והנדרשים לצורך הבדיקות) הינו מוגבל. כאשר מוטי עובר במעבדה, פרט לפריקת הבדיקות, הוא גם לוקח משם את כל המטושים הזמינים. כאשר נגמרים למוטי המטושים באמבולנס הוא חייב לעבור במעבדה, גם אם המקרר שלו ריק (כלומר אין לו בדיקות להעביר למעבדה). בכל מעבדה יש מספר אחר של מטושים זמינים.

מוטי עמוס בלימודים ולכן הוא רוצה לסיים את המשמרת כמה שיותר מהר ולהגיע הביתה כדי לעבוד על ההגשות שלו. למזלו, חברים של מוטי (זה אתם!) במקרה לוקחים הסמסטר את הקורס "מבוא לבינה מלאכותית". מוטי מבקש מכם לעזור לו לתכנן מראש את הדרך היעילה ביותר לבצע את כל הבדיקות.

פורמאליזם – הגדרת הבעיה

נתונה רשת כבישים בצורת גרף $StreetsMap = (V_{map}, E_{map})$ שבה כל צומת מייצג צומת דרכים (junction), והקשתות מייצגות דרך (כביש) המקשרת בין צמתי דרכים (links).

לאמבולנס יש קיבולת מרבית של $AmbulanceTestsCapacity > 0$ בדיקות.

נתונה נקודת מוצא על רשת הכבישים $v_0 \in V_{map}$, וכן נתונות $k \in \mathbb{N}$ דירות שאליהן יש להגיע ולבצע בדיקה: $Apartments = \{d_1, \dots, d_k\} \neq \emptyset$, כאשר דירה i כוללת: מיקום $d_i.loc \in V_{map}$, ומספר הדיירים שיש לבדוק $d_i.roommates \in \{1,2, \dots, AmbulanceTestsCapacity\}$.

נתונות $m \in \mathbb{N}$ מעבדות $Labs = \{l_1, \dots, l_m\}$. לכל מעבדה יש מיקום $l_i.loc \in V_{map}$ וכן מספר מטושים זמינים $l_i.matoshim \in \mathbb{N}^+$.

לצורך פשטות, במהלך כל התרגיל נניח כי הדירות, המעבדות ונק' המוצא הינן נקודות זרות במפה. כלומר $\left|\{v_0\} \cup \{d_i.loc\}_{i\in[k]} \cup \{l_i.loc\}_{i\in[m]}\right| \equiv k + m + 1$.

סידור ביקורים הינו פרמוטציה $\pi = w_1, \dots, w_k$ של הנקודות $\{d_i.loc\}_{i\in[k]}$.

את איכות סידור ביקורים π שיחושב ע"י התוכנית נמדוד לפי מספר מדדים שונים, כפי שיפורט בהמשך. הפתרון לבעיה לפי מדד איכות נתון הינו סידור ביקורים אצל לקוחות בעל מחיר מינימלי ע"פ מדד איכות זה.

הבנת קושי הבעיה

בשלב זה אנחנו רוצים לקבל קצת אינטואיציה לגבי הקושי של הבעיה. המטרה היא להשתכנע שאנחנו לא מסוגלים לפתור את הבעיה בעזרת חיפוש brute-force (בגלל מגבלת משאבים). לצורך זאת, ראשית ננסה להעריך את מספר הסידורים החוקיים השונים אותם יש לבחון במסגרת ריצת brute-force. לשם פשטות החישוב אנו מתעלמים כרגע משאר אילוצי הבעיה.

תרגיל

1. יבש (2.5 נק'): מלאו את הטבלה הבאה. הזינו את מספר הפרמוטציות האפשריות (וערכי $\log_2$ שלהן) עבור ערכי k (מספר הדירות שיש לבקר בהן) המופיעים בטבלה. ~~היעזרו בנוסחה שמצאתם בסעיף (1)~~. נניח שמחשב יחיד יכול לבחון 2^{30} סידורים בשנייה. מלאו בעמודה האחרונה כמה זמן ייקח למחשב זה לבדוק כל אחד מהסידורים (לפי היחידות המפורטות).

k	$\#possiblePaths$	$\log_2(\#possiblePaths)$	Calculation time
10	10! = 3628800	21.8	< 1 sec
13	13! = 6227020800	32.53	5.8 [sec]
15	15! = 1.307x10^12	40.25	20.29 [mins]
16	16!=2.0922x10^13	44.25	5.42 [hours]
17	17!=3.55x10^14	48.37	3.834 [days]
20	20!=2.43x10^18	61.1	71.85 [years]
21	21!=5.11x10^19	65.47	1508.83 [years]
24	24!=6.205x10^23	79.05	18.323 [million years]

חלק ב' – הגדרת מרחב החיפוש במפה

כאמור נתונה רשת כבישים בצורת גרף $StreetsMap = (V_{map}, E_{map})$. בעיית המפה עוסקת במציאת מסלול ברשת הכבישים $StreetsMap$ בעל עלות מינימלית (ביחס לפונק' עלות נתונה המוגדרת על כבישים במפה). בחלק זה נייצג את בעיית המפה כמרחב חיפוש. ניצמד להגדרה שלמדנו בכיתה עבור מרחבי חיפוש. אנו מתחילים בבעיית המפה משום שהיא בעיה יחסית פשוטה, הייצוג שלה כמרחב חיפוש הוא אינטואיטיבי ואנו אכן נעשה בה שימוש בחלקים הבאים.

בהינתן רשת הכבישים, נקודת מקור $v_{src} \in V_{map}$ ונקודת יעד $v_{dst} \in V_{map}$, נגדיר מרחב חיפוש עבור מציאת מסלול ביניהן:

$$\mathcal{S}_{map} \triangleq \langle S_{map}, O_{map}, I_{map}, G_{map} \rangle$$

- **קבוצת המצבים:**
נרצה לייצג מצב כך שיחזיק את כל המידע שנחוץ לנו עליו במהלך החיפוש במרחב. במקרה המדובר מספיק לשמור את הצומת ברשת הכבישים.
$$S_{map} \triangleq \{(v: u) | u \in V_{map}\}$$
- **קבוצת האופרטורים:**
ניתן לעבור ממצב אחד לעוקבו בתנאי שיש כביש מהצומת המיוצג ע"י המצב הראשון לצומת המיוצג ע"י המצב העוקב.
$$O_{map} \triangleq \{(s_1, s_2) | s_1, s_2 \in S_{map} \wedge (s_1.v, s_2.v) \in E_{map}\}$$
- **עלות אופרטור:**
נגדיר את פונק' העלות עבור מעבר מצומת דרכים אחד $s_1 \in S_{map}$ לצומת דרכים עוקב שלו $o \in O_{map}, s_2 =$ $o(s_1)$ באופן הבא:
$$cost_{map}^{dist}((s_1, s_2)) = roadLength((s_1.v, s_2.v))$$
- **המצב ההתחלתי:**
$$I_{map} \triangleq (v: v_{src})$$
- **מצבי המטרה:**
$$G_{map} \triangleq \{(v: v_{dst})\}$$

חלק ג' – הגדרת מרחב החיפוש של בעיית מד"א (9 נק' יבש)

בהינתן רשת הכבישים, נקודת המוצא ורשימת ההזמנות, נגדיר מרחב חיפוש עבור בעיית מד"א:

$$\mathcal{S}_{MDA} = \langle S_{MDA}, O_{MDA}, I_{MDA}, G_{MDA} \rangle$$

- **קבוצת המצבים:**

$$S_{MDA} \triangleq \{(v_0, \emptyset, \emptyset, InitialNrMatoshimAmb, \emptyset)\} \cup \left\{ \left(\begin{array}{c} \underbrace{curLoc}_{\text{המיקום הנוכחי}}; \underbrace{Taken}_{\text{קבוצת בדיקות על האמבולנס}}; \underbrace{Transferred}_{\text{קבוצת בדיקות שהועברו למעבדה}}; \\ \underbrace{Matoshim}_{\text{מספר מטושים זמינים באמבולנס}}; \underbrace{VisitedLabs}_{\text{המעבדות שבוקרו}} \end{array} \right) \middle| \begin{array}{c} curLoc \in \\ \{d.loc | d \in Taken\} \cup \{l.loc | l \in VisitedLabs\} \\ \cancel{\emptyset \neq} Taken \cup Transferred \subseteq Apartments \\ Taken \cap Transferred = \emptyset \\ Matoshim \in \mathbb{N} \\ VisitedLabs \subseteq Labs \end{array} \right\}$$

- **קבוצת האופרטורים:**

<u>אופרטורים עבור ביקור בדירה:</u>

ישנם k אופרטורים כאלו. לכל $i \in [k]$ נגדיר את האופרטור o_{d_i} להיות האופרטור שבהינתן מצב $s \in S_{MDA}$, המצב העוקב $o_{d_i}(s)$ (המתקבל מהפעלת האופרטור על המצב s) הינו מצב שבו המיקום הנוכחי הוא הנק' $d_i.loc$, מס' המטושים הזמינים באמבולנס קטן ב- $d_i.roommates$ וכן הבדיקות שנלקחו מדירה d_i נמצאות במקרר של האמבולנס ($d_i \in Taken$).

הפעלת האופרטור o_{d_i} על מצב $s \in S_{MDA}$ אפשרית אמ"מ הבדיקות של הדירה d_i לא נלקחו כבר ($d_i \notin s.Taken \cup s.Transferred$), יש באמבולנס מספיק מטושים זמינים בשביל לקחת בדיקות לכל הדיירים בדירה d_i, וכן יש די מקום פנוי במקרר באמבולנס עבור אחסון כל הבדיקות מדירה זו, כלומר מתקיים התנאי:

$$CanVisit(s, d_i) \triangleq \left[\begin{array}{c} d_i \notin s.Taken \cup s.Transferred \\ \wedge \\ d_i.roommates \leq s.Matoshim \\ \wedge \\ d_i.roommates \leq AmbulaceTestsCapacity - \sum_{d \in s.Taken} d.roommates \end{array} \right]$$

הגדרה פורמלית: לכל $i \in [k]$ נגדיר את האופרטור o_{d_i} באופן הבא:

$$\forall_{s \in S_{MDA}}: o_{d_i}(s) \triangleq \begin{cases} \left(\begin{array}{c} d_i.loc; s.Taken \cup \{d_i\}; s.Transferred, \\ s.Matoshim - d_i.roommates; s.VisitedLabs \end{array} \right) & ; \quad CanVisit(s, d_i) \\ \emptyset & ; \quad otherwise \end{cases}$$

וכן תחום הפעולה של האופרטור o_{d_i} מוגדר בהתאם:

$$Domain(o_{d_i}) = \{s \in S_{MDA} | o_{d_i}(s) \neq \emptyset\} = \{s \in S_{MDA} | CanVisit(s, d_i)\}$$

<u>אופרטורים עבור מעבר במעבדה:</u>

ישנם m אופרטורים כאלו. לכל $i \in [m]$ נגדיר את האופרטור o_{l_i} להיות האופרטור שבהינתן מצב $s \in S_{MDA}$, המצב העוקב $o_{l_i}(s)$ (המתקבל מהפעלת האופרטור על המצב s) הינו מצב שבו המיקום הנוכחי הוא הנק' $l_i.loc$, הבדיקות שבמקרר באמבולנס מועברות למעבדה (המקרר נותר ריק), וכן המטושים הזמינים במעבדה מאוחסנים באמבולנס.

הפעלת האופרטור o_{l_i} על מצב $s \in S_{MDA}$ אפשרית רק אם המקרר באמבולנס אינו ריק **או** שהמעבר במעבדה יוסיף מטושים נוספים לאמבולנס (לא עברנו במעבדה זו בעבר). כלומר מתקיים התנאי:

$$CanVisit(s, l_i) = s.Taken \neq \emptyset \ \vee \ l_i \notin s.VisitedLabs \cancel{> 0}$$

הגדרה פורמלית: לכל $i \in [m]$ נגדיר את האופרטור o_{l_i} באופן הבא: (אינדיקטור כמו בהסתברות)

$$\forall_{s \in S_{MDA}}: o_{l_i}(s) \triangleq \begin{cases} \left(\begin{array}{c} l_i.loc; \emptyset; s.Transferred \cup s.Taken; \\ s.Matoshim + l_i.matoshim \cdot \mathbb{I}_{l_i \notin s.VisitedLabs}; \\ s.VisitedLabs \cup \{l_i\} \end{array} \right) & ; \quad CanVisit(s, l_i) \\ \emptyset & ; \quad otherwise \end{cases}$$

וכן תחום הפעולה של האופרטור o_{l_i} מוגדר בהתאם:

$$Domain(o_{l_i}) = \{s \in S_{MDA} | o_{l_i}(s) \neq \emptyset\} = \{s \in S_{MDA} | CanVisit(s, l_i)\}$$

<u>לבסוף, קבוצת כל האופרטורים הינה:</u>

$$O_{MDA} \triangleq \{o_{d_i}\}_{i\in[k]} \cup \{o_{l_i}\}_{i\in[m]}$$

- **עלות אופרטור:**

במטלה נגדיר 2 פונקציות עלות עבור הפעלת אופרטור $o \in O_{MDA}$ על מצב $s \in Domain(o)$.

1. אורך המסלול הקצר ביותר על גבי המפה מהנק׳ בה נמצא האמבולנס במצב s לנק׳ בה מצוי האמבולנס במצב $o(s)$:

$$cost_{MDA}^{dist}(s,o) \triangleq optimalDistanceOnStreetsMap(s.curLoc, o(s).curLoc)$$

2. מרחקי הנסיעה שעברו כל הבדיקות במקרה:

$$cost_{MDA}^{test\ travel}(s,o) \triangleq \left[\sum_{d\in s.Taken} d.roommates\right] \cdot cost_{MDA}^{dist}(s,o)$$

   - כל אחת משתי פונק׳ העלויות הללו למעשה מגדירה ווריאציה לבעיה. בסופו של דבר כשפותרים בעיה צריך להחליט באיזו פונק׳ עלות משתמשים.
   - בחלקים הראשונים של התרגיל נשתמש בפונק׳ העלות $cost_{MDA}^{dist}$ ובחלקים מתקדמים נעשה שימוש ב- $cost_{MDA}^{test\ travel}$.
   - שימו לב: בהינתן אופרטור $o \in O_{MDA}$ ומצב $s \in Domain(o)$, על מנת לחשב את $cost_{MDA}^{dist}(s,o)$ או את $cost_{MDA}^{test\ travel}(s,o)$, **יש צורך בפתרון של בעיית המפה**.

- **המצב ההתחלתי:**

$$I_{MDA} \triangleq (v_0, \emptyset, \emptyset, InitialNrMatoshimAmb, \emptyset)$$

- **מצבי המטרה:**

$$G_{MDA} \triangleq \{(l_i.loc, \emptyset, Apartments, M, L) \in S | i \in [m], M \in \mathbb{N}, L \subseteq Labs\}$$

תרגילים

2. יבש (1.5 נק׳): מהם ערכי הקיצון (המקסימלי והמינימלי) האפשריים של דרגת היציאה במרחב החיפוש? נמקו בקצרה.
<u>תשובה:</u> דרגת היציאה המינימלית היא אפס עבור מצב בור או מצב מטרה שלא ניתן להפעיל עליהם אף אופרטור.
דרגת היציאה המקסימלית תתקבל עבור המצב ההתחלתי $m+k$ במידה וניתן להפעיל עליו כל אחד מהאופרטורים (o_{l_i} לכל $i \in [m]$ ו-o_{d_i} לכל $i \in [k]$).

3. יבש (1.5 נק׳): האם ייתכנו מעגלים במרחב המצבים שלנו? אם כן תנו דוגמה למעגל כזה, אחרת נמקו.
<u>תשובה:</u> כן, יתכן מצב שבוא נגמרו המטושים באמבולנס ולכן במצב הבא הוא מגיע למעבדה x (שעוד לא ביקרו בה) לקחת מטושים ולמסור בדיקות. במצב שלאחר מכן הוא מגיע לדירה d לקחת בדיקות ושוב נגמרו המטושים כי הם הספיקו בדיוק לכמות הדיירים בדירה. במצב הבא הוא יכול לנסוע למעבדה x (שכבר ביקרו בה אבל המקרר לא ריק) כדי למסור בדיקות.
כלומר נסגר מעגל וכל האופרטורים והמצבים בדרך היו תקינים לפי הגדרות מרחב החיפוש.

4. יבש (1.5 נק׳): כמה מצבים יש במרחב זה (כפי שהוגדר)? האם כולם ישיגים? נמקו.
<u>תשובה:</u> מספר המצבים שיש במרחב ללא התייחסות למספר המטושים הוא
$\underbrace{1}_{\text{מצב התחלתי}} + \underbrace{k}_{\text{המיקום הוא דירה}} \cdot \underbrace{3^{k-1}}_{\text{אפשרויות ליתר הדירות}} \cdot \underbrace{2^m}_{\text{אפשרויות למעבדות}} + \underbrace{m}_{\text{המיקום הוא מעבד}} \cdot \underbrace{2^k}_{\text{אפשרויות לדירות}} \cdot \underbrace{2^{m-1}}_{\text{אפשרויות ליתר המעבדות}}$ נסביר:
- יש מצב התחלתי יחיד שלא ניתן לחזור אליו.
- יש מצבים בהם המיקום הוא דירה: k דירות בהן אפשר להימצא, ליתר k-1 הדירות יש שלוש אפשרויות {לא ביקרנו, אספנו, מסרנו}, ולכל m המעבדות יש שתי אפשרויות {ביקרנו, לא ביקרנו}.
- יש מצבים בהם המיקום הוא מעבדה: m מעבדות בהן אפשר להימצא, ליתר m-1 המעבדות יש שני אפשרויות {ביקרנו, לא ביקרנו}, לכל k הדירות יש שתי אפשרויות {לא ביקרנו, מסרנו} בגלל שכשמגיעים למעבדה מוסרים את הבדיקות.
בכל אחד מהמצבים האפשריים – מספר המטושים על האמבולנס נקבע באופן יחיד לפי מספר המעבדות ומספר הדירות שבהן ביקרנו.

נשים לב כי לא כל המצבים ישיגים מאחר והאופרטורים צריכים לעמוד בתנאים מסויימים. למשל, המצב $\{(l_i.loc, \emptyset, \emptyset, 0, \{l_i\})\}$ מתאר כי האמבולנס הגיע למעבדה כלשהי בה אין מטושים כאשר המקרר שלו ריק (הוא לא רוקן אותו כשהגיע למעבדה מאחר והקבוצה $Transferred$ ריקה).

עם זאת מכיוון שמספר המטושים האפשרי אינו חסום, נקבל שישנם אינסוף מצבים אפשריים. קיימים גם מצבים לא ישיגים: לדוגמא, כל המצבים בהם מספר המטושים באמבולנס גדול ממספר המטושים ההתחלתי ועוד סך המטושים בכל המעבדות. לא ניתן לקבל יותר מטושים מסכום זה, ולכן זה מצב לא ישיג אך מהגדרת המרחב הוא קיים.

5. יבש (1.5 נק'): האם ייתכנו בורות ישיגים מהמצב ההתחלתי שאינם מצבי מטרה במרחב המצבים? אם כן – איך זה ייתכן? אם לא – למה?
   תשובה: כן, יתכנו בורות ישיגים מהמצב ההתחלתי שאינם מצבי מטרה. למשל, המצב $\{(l_i.loc, \emptyset, Apartments\backslash\{d_j\}, 0, Labs)\}$ מתאר מצב בוא כבר ביקרנו בכל המעבדות וגם נגמרו כל המטושים, לכן לא ניתן להגיע למעבדה אחרת. כמו כן, לא סיימנו לבקר בכל הדירות מכיוון שלא ביקרנו בדירה d_j אבל אי אפשר להגיע אליה כי אין מטושים, ולכן מצב זה הוא מצב בור.

6. יבש (1.5 נק'): מהו טווח האורכים האפשריים של מסלולים במרחב ממצב התחלתי אל מצב סופי? (אורך מסלול = מס' הקשתות)
   תשובה: כל מסלול מחייב אותנו לעבור בכל דירה פעם אחת בדיוק.
   האורך המינימלי, בהנחה ומספר המטושים ההתחלתי על האמבולנס מספיק לכל האנשים בדירות, מתקבל על ידי ביקור בדירות ולבסוף ביקור במעבדה אחת (כדי למסור את הבדיקות). סה"כ $k+1$.
   האורך המקסימלי יתקבל עבור סדרת הצעדים הבאה למשל:
   - מספר המטושים ההתחלתי הוא אפס ולכן נצטרך להתחיל מביקור במעבדה.
   - לכל מעבדה יש מספר x של מטושים ובכל דירה יש מספר x של דיירים.
   - על כל מעבדה נבצע פעם אחת מעגל כפי שתיארנו בשאלה 3, כך שבכל מעבדה נבקר פעמיים ובכל דירה פעם אחת. סה"כ $2m+k$.

7. יבש (1.5 נק'): הגדירו פורמלית ובצורה ישירה את פונקציית העוקב $Succ: S \rightarrow \mathcal{P}(S)$ המתאימה לבעיה זו (ללא שימוש בקבוצת האופרטורים O).
   שימו לב, אנו מצפים לביטוי מהצורה: $Succ(s) = \{(?,?,?,?,?)|?\} \cup \{(?,?,?,?,?)|?\}$
   תשובה: נגדיר את פונקציית העוקב $\boldsymbol{Succ}: \boldsymbol{S} \rightarrow \boldsymbol{P}(\boldsymbol{S})$ באופן הבא:

$$Succ(s) \stackrel{\text{def}}{=} \left\{ s' \in S \,\middle|\, s' = \begin{pmatrix} d_i.loc;\ s.Taken \cup \{d_i\};\ s.Transferred; \\ s.Matoshim - d_i.roommates; s.VisitedLabs \end{pmatrix} s.t. CanVisit(s, d_i) \vee s' \right.$$

$$\left. = \begin{pmatrix} l_i.loc; \emptyset;\ s.Transferred \cup s.Taken; \\ s.Matoshim + l_i.matoshim \cdot \mathbb{I}_{l_i \notin s.VisitedLabs}; \\ s.VisitedLabs \cup \{l_i\} \end{pmatrix} s.t. CanVisit(s, l_i) \right\}$$

חלק ד' – מתחילים לתכנת (1 נק' יבש)

הורידו את *ai_hw1.zip* מהאתר וטענו את התיקייה שבתוכו לסביבת העבודה המועדפת עליכם.

מבנה מפת הדרכים

בתרגיל נעשה שימוש במפת רשת הכבישים של העיר תל אביב. את המפה אנו טוענים פעם אחת בקובץ main.py למשתנה גלובלי בשם streets_map. המפות מיוצגות ע"י אובייקט מטיפוס ***StreetsMap***. הטיפוס ***StreetsMap*** יורש מ- *dict*; כלומר ***StreetsMap*** הינו בבסיסו מיפוי ממזהה ייחודי של צומת במפה (מספר שלם) לאובייקט מטיפוס ***Junction*** שמייצג את אותו הצומת.

כל צומת הוא כאמור מטיפוס ***Junction***. לצומת יש את השדות הבאים: (1) מספר *index* ייחודי; (2+3) קואורדינטות lat, lon (קווי אורך ורוחב) של המיקום הגיאוגרפי של הצומת במפה; ו- (4) רשימה outgoing_*links* המכילה את כל הקשתות לשכניו. כל קשת כזו מייצגת כביש במפה. קשת היא אובייקט מטיפוס ***Link*** עם מאפיינים source ו- target – המזהים של צמתי המקור והיעד של הקשת, distance – אורך הכביש (במטרים).

שימו לב: אין לבצע באף שלב טעינה של מפות. טענו בשבילכם את המפות פעם אחת בתחילת קובץ ה- main.py שסיפקנו לכם. יש לכם גישה למפות בכל מקום בו תזדקקו להן. באופן כללי, טעינות מיותרות בקוד יגרמו להגדלת זמן הפתרון ואולי יובילו לחריגה מהזמן המקסימלי.

הכרת תשתית הקוד הכללית (שסופקה לכם בתרגיל זה) לייצוג ופתרון בעיות גרפים

המחלקות **GraphProblem**, **GraphProblemState** (בקובץ graph_search/graph_problem_interface.py) מגדירות את הממשק (interface) בו נשתמש על מנת לייצג מרחב מצבים. אלו הן מחלקות אבסטרקטיות – כלומר מוגדרות בהן מתודות שאינן ממומשות. לכן, בפרט, לא ניתן ליצור ישירות אובייקט מטיפוסים אלו (ואין לכך שום משמעות). כדי להגדיר מרחב מצבים חדש יש לרשת (inherit) משתי המחלקות הנ"ל. בהמשך התרגיל תראו דוגמא למימוש של מרחב מצבים באופן הנ"ל (שסיפקנו עבורכם) ותממשו מרחב נוסף כזה בעצמכם.

המחלקה **GraphProblemSolver** (באותו הקובץ) מגדירה את הממשק בו נשתמש בכדי לחפש בגרפים. למחלקה יש מתודה אבסטרקטית אחת בשם solve_problem() שמקבלת כפרמטר בעיה (אובייקט מטיפוס שיורש מ- GraphProblem) ומחזירה את תוצאות החיפוש (אובייקט מטיפוס SearchResult). כל אלג' חיפוש שנממש ישתמש בממשק הנ"ל (ירש ממחלקה זו או ממחלקה שיורשת ממנה).

שימו לב: אלגוריתמי החיפוש אותם נממש לאורך התרגיל יהיו כלליים בכך שלא יניחו כלום על הבעיות אותן יפתרו, פרט לכך שהן תואמות לממשק המוגדר ע"י GraphProblemState, GraphProblem. כלומר, בעתיד תוכלו לקחת את המימוש שלכם מקורס זה כפי שהוא בכדי לפתור בעיות חדשות.

המחלקה **BestFirstSearch** (בקובץ graph_search/best_first_search.py) יורשת מהמחלקה GraphProblemSolver (שתוארה לעיל) ומייצגת אלגוריתמי חיפוש מהמחלקה Best First Search. כפי שנלמד בכיתה, אלו הם אלגוריתמים שמתחזקים תור עדיפויות בשם **open** של צמתים (פתוחים) הממתינים לפיתוח. כל עוד תור זה אינו ריק, האלג' בוחר את הצומת הבא בתור העדיפויות ומפתח אותו. המחלקה ממשת את המתודה solve_problem() בהתאם. דוגמאות לאלגוריתמים ממשפחה זו: Uniform Cost, Greedy Best Search, A*. כאמור, Best First Search הינה משפחה של אלגוריתמי חיפוש (מכונה גם "אלגוריתם גנרי"), כלומר היא מגדירה שלד כללי של מבנה האלגוריתם, ומשאירה מספר פרטי מימוש חסרים. לכן, המחלקה BestFirstSearch אף היא אבסטרקטית. גם בה מוגדרות מספר מתודות אבסטרקטיות שעל היורש (אלגוריתם החיפוש הקונקרטי) לממש. המתודה האבסטרקטית _calc_node_expanding_priority() מאפשרת ליורש להגדיר את אופן חישוב ערך ה- f-score של צומת. כזכור, ערך זה משמש כעדיפות של צומת בתור העדיפויות open (בתרגיל זה אנו מכנים ערך זה בשם expanding priority). המתודה האבסטרקטית _open_successor_node() מאפשרת ליורש להגדיר את אופן הטיפול בצומת חדש שזה עתה נוצר ומייצג מצב עוקב של המצב המיוצג ע"י הצומת שנבחר אחרון לפיתוח (הכנסה ל- open, בדיקה ב- close במידת הצורך). בנוסף, האלגוריתם מאפשר מצב של חיפוש-גרף כפי שנלמד בכיתה, ע"י תחזוק אוסף **סגור / close** של צמתים שכבר פיתחנו במהלך החיפוש (ה- constructor של BestFirstSearch מקבל פרמטר בולאני בשם use_close שקובע האם להשתמש ב- close).

מבנה הקלטים לבעיית מד"א ואופן טעינתם

המחלקה **MDAProblemInput** (בקובץ problems/mda_problem_input.py) מייצגת קלט לבעיית מד"א. מחלקה זו אחראית לטעינה של קלטים שסיפקנו לכם כקבצי טקסט. המחלקה שמייצגת את בעיית מד"א (נראה בהמשך) תקבל אובייקט מסוג זה. בקובץ הראשי main.py כבר כתובות שורות הקוד שאחראיות להשתמש במחלקה זו ע"מ לטעון את הקלטים הנדרשים במקומות הנדרשים. הבהרה: אין לבצע טעינות נוספות של הקלטים. אנו עשינו זאת בשבילכם בכל המקומות הנדרשים.

תרגילים

8. רטוב + יבש: סעיף זה נועד על מנת להתחיל להכיר את מבנה הקוד.
   a. חלצו את תוכן התיקייה ai_hw1.zip.
   b. אם אתם משתמשים ב- IDE לכתיבת והרצת קוד פייתון (אנחנו ממליצים מאוד על PyCharm), פתחו פרויקט חדש שתיקיית האם שלו היא התיקייה הראשית של קובץ ה- zip שחולץ (אמור להיות שם קובץ בשם main.py).
   c. פתחו את הקובץ main.py, קראו את החלק בקוד שמעליו מופיעה הערה המתאימה למספר סעיף זה. שורות קוד אלו מבצעות: יצירת בעיית מפה חדשה, יצירת אובייקט מסוג אלג׳ חיפוש uniform cost, הרצת אלג׳ החיפוש על הבעיה ולבסוף הדפסת התשובה שהתקבלה מההרצה. הריצו את הקובץ. וודאו שמודפסת לכם שורה למסך שמתארת את פתרון בעיית החיפוש במפה. זאת גם הזדמנות טובה לוודא שהחבילות numpy, scipy, networkx, matplotlib מותקנות אצלכם כראוי.
   d. פתחו את הקובץ problems/map_problem.py. בתוכו יש לכם שתי משימות (המסומנות ע"י הערות **TODO** כמו בעוד מקומות רבים לאורך המטלה). אחת במתודה בשם expand_state_with_costs() והשנייה במתודה בשם is_goal(). בשתי משימות אלו אתם מתבקשים לבצע שינוי בקוד של המחלקה MapProblem כדי לתקן ולהשלים את המימוש שסיפקנו לכם.
   e. זוהי בעיה פשוטה ולכן נוח להתחיל בה כדי להתמצא בקוד שסופק לכם. עיינו במימוש של המחלקות בקובץ זה זו וודאו שאתם מבינים את החלקים השונים. שימו לב שמחלקה זו יורשת מהמחלקה GraphProblem (שתוארה מקודם) וממששת את המתודות האבסטרקטיות הנדרשות.
   f. עתה, לאחר תיקון קוד המחלקה MapProblem, הריצו בשנית את main.py.
   g. יבש (1 נק׳): הוסיפו לדו"ח את פלט הריצה המתוקנת.

תשובה:

```
/usr/local/bin/python3.7 /Users/Tsuf/PycharmProjects/intro-to-ai/HW1/ai_hw1/main.py

Running all experiments



Solve the map problem.

StreetsMap(src: 54 dst: 549)      UniformCost           time:  1.60   #dev: 17354   |space|: 17514   total_g_cost: 7465.52560   |path|: 136   path: [  54 ==>   55 ==>   56 ==>   57 ==>   58 ==>   59 ==>   60 ==> 28893 ==> 14580 ==> 14590 ==> 14591 ==> 14592 ==> 14593 ==> 81892 ==> 25814 ==>   81 ==> 26236 ==> 26234 ==>  1188 ==> 33068 ==> 33069 ==> 33070 ==> 15474 ==> 33071 ==>  5020 ==> 21699 ==> 33072 ==> 33073 ==> 33074 ==> 16203 ==>  9847 ==>  9848 ==>  9849 ==>  9850 ==>  9851 ==>   335 ==>  9852 ==> 82906 ==> 82907 ==> 82908 ==> 82909 ==> 95454 ==> 96539 ==> 72369 ==> 94627 ==> 38553 ==> 72367 ==> 29007 ==> 94632 ==> 96540 ==>  9269 ==> 82890 ==> 29049 ==> 29026 ==> 82682 ==> 71897 ==> 83380 ==> 96541 ==> 82904 ==> 96542 ==> 96543 ==> 96544 ==> 96545 ==> 96546 ==> 96547 ==> 82911 ==> 82928 ==> 24841 ==> 24842 ==> 24843 ==>  5215 ==> 24844 ==>  9274 ==> 24845 ==> 24846 ==> 24847 ==> 24848 ==> 24849 ==> 24850 ==> 24851 ==> 24852 ==> 24853 ==> 24854 ==> 24855 ==> 24856 ==> 24857 ==> 24858 ==> 24859 ==> 24860 ==> 24861 ==> 24862 ==> 24863 ==> 24864 ==> 24865 ==> 24866 ==> 82208 ==> 82209 ==> 82210 ==> 21518 ==> 21431 ==> 21432 ==> 21433 ==> 21434 ==> 21435 ==> 21436 ==> 21437 ==> 21438 ==> 21439 ==> 21440 ==> 21441 ==> 21442 ==> 21443 ==> 21444 ==> 21445 ==> 21446 ==> 21447 ==> 21448 ==> 21449 ==> 21450 ==> 21451 ==>   621 ==> 21452 ==> 21453 ==> 21454 ==> 21495 ==> 21496 ==>   539 ==>   540 ==>   541 ==>   542 ==>   543 ==>   544 ==>   545 ==>   546 ==>   547 ==>   548 ==>   549]
```

חלק ה' – אלגוריתם A* (3 נק' יבש)

עתה נתחיל במימוש Weighted A*.

עיינו בקובץ framework/graph_search/astar.py. שם מופיע מימוש חלקי למחלקה AStar. שימו לב: המחלקה AStar יורשת מהמחלקה האבסטרקטית BestFirstSearch (הסברנו עליה בחלק ד'). זהו את החלק בהצהרת המחלקה AStar בו הירושה מוגדרת. המחלקה AStar צריכה לממש את המתודות האבסטרקטיות שמוגדרות ע"י BestFirstSearch. הכותרות של מתודות אלו מופיעות כבר במימוש החלקי של המחלקה AStar, אך ללא מימושן. בסעיף זה נרצה להשלים את המימוש של המחלקה AStar ולבחון אותה.

שימו לב: לאורך התרגיל כולו אין לשנות את החתימות של המתודות שסיפקנו לכם. בנוסף, אין לשנות קבצים שלא התבקשתם באופן מפורש.

תרגילים

9. רטוב: השלימו את המשימות הדרושות תחת הערות ה- **TODO** בקובץ framework/graph_search/astar.py כך שנקבל מימוש תקין לאלגוריתם Weighted A*, כפי שראיתם בהרצאות. בכדי להבין את מטרת המתודות השונות שעליכם לממש, הביטו במימוש המחלקה BestFirstSearch שעושה בהן שימוש. בנוסף, היעזרו במימוש שסיפקנו לכם ל- UniformCost (בקובץ framework/graph_search/uniform_cost.py). שימו לב בשקפים מההרצאה להבדלים בין אלג' UniformCost לבין אלג' A*.
10. רטוב: בכדי לבחון את האלג' שזה עתה מימשתם, השלימו את המשימות הדרושות תחת הערות ה- **TODO** הרלוונטיות לסעיף זה בקובץ main.py. כידוע, לצורך הרצת A* יש צורך בהיוריסטיקה. ה- constructor של המחלקה AStar מקבל את טיפוס ההיוריסטיקה שמעוניינים להשתמש בה. לצורך בדיקת שפיות, הפעילו את ה- A* על בעיית המפה שפתרתם בסעיף הקודם עם NullHeuristic (מסופקת בקובץ framework/graph_search/graph_problem_interface.py. מחלקה זו כבר מוכרת מ- main.py ללא צורך בביצוע import נוסף. באופן כללי אין לעשות imports בתרגיל זה כלל). וודאו שהתוצאה המודפסת זהה לזו שקיבלתם בעזרת Uniform Cost.
11. רטוב + יבש (1 נק'): כפי שראינו בהרצאות ובתרגולים, היוריסטיקה פשוטה לבעיית המפה היא מרחק אווירי לפתרון. היכנסו לקובץ problems/map_heuristics.py וממשו את ההיוריסטיקה הזו במחלקה AirDistHeuristic (מלאו את המקומות החסרים תחת ההערות שהשארנו לכם שם). כעת הריצו שוב את הבעיה שפתרתם בסעיף הקודם, אך כעת בעזרת ההיוריסטיקה (מלאו ב- main.py את המשימות שקשורות לסעיף זה). העתיקו לדו"ח את פלט הריצה. כתוב בדו"ח את מס' פיתוחי המצבים היחסי שחסכנו לעומת הריצה העיוורת (ההפרש חלקי מס' הפיתוחים בריצה **בלי** ~~עם~~ ההיוריסטיקה).

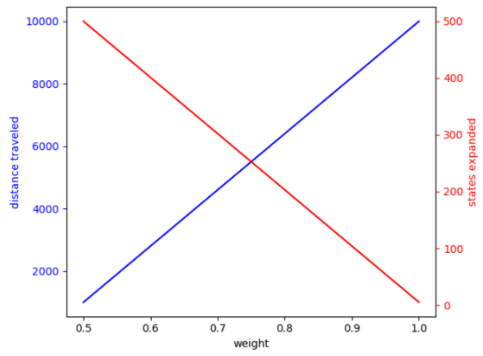
שימו לב: בכדי לחשב מרחק בין זוג Junctions, **אין לחשב את המרחק האווירי ישירות על ידי קווי רוחב ואורך**, אלא יש להשתמש במתודה calc_air_distance_from() של המחלקה Junction.

תשובה: מספר פיתוחי המצבים היחסי שחסכנו לעומת הריצה העיוורת הוא $\frac{17354-2015}{17354}=0.8838$.

```
/usr/local/bin/python3.7 /Users/Tsuf/PycharmProjects/intro-to-ai/HW1/ai_hw1/main.py

Running all experiments


Solve the map problem.

StreetsMap(src: 54 dst: 549)       UniformCost             time:  0.99  #dev: 17354  |space|: 17514   total_g_cost:
7465.52560   |path|: 136   path: [  54  ==>    55  ==>    56  ==>    57  ==>    58  ==>    59  ==>    60  ==> 28893  ==> 14580
==> 14590  ==> 14591  ==> 14592  ==> 14593  ==> 81892  ==> 25814  ==>    81  ==> 26236  ==> 26234  ==>  1188  ==>
33068  ==> 33069  ==> 33070  ==> 15474  ==> 33071  ==>  5020  ==> 21699  ==> 33072  ==> 33073  ==> 33074  ==>
16203  ==>  9847  ==>  9848  ==>  9849  ==>  9850  ==>  9851  ==>   335  ==>  9852  ==> 82906  ==> 82907  ==> 82908
==> 82909  ==> 95454  ==> 96539  ==> 72369  ==> 94627  ==> 38553  ==> 72367  ==> 29007  ==> 94632  ==> 96540
==>  9269  ==> 82890  ==> 29049  ==> 29026  ==> 82682  ==> 71897  ==> 83380  ==> 96541  ==> 82904  ==> 96542
==> 96543  ==> 96544  ==> 96545  ==> 96546  ==> 96547  ==> 82911  ==> 82928  ==> 24841  ==> 24842  ==> 24843
==>  5215  ==> 24844  ==>  9274  ==> 24845  ==> 24846  ==> 24847  ==> 24848  ==> 24849  ==> 24850  ==> 24851
==> 24852  ==> 24853  ==> 24854  ==> 24855  ==> 24856  ==> 24857  ==> 24858  ==> 24859  ==> 24860  ==> 24861
==> 24862  ==> 24863  ==> 24864  ==> 24865  ==> 24866  ==> 82208  ==> 82209  ==> 82210  ==> 21518  ==> 21431
==> 21432  ==> 21433  ==> 21434  ==> 21435  ==> 21436  ==> 21437  ==> 21438  ==> 21439  ==> 21440  ==> 21441
```

```
==> 21442 ==> 21443 ==> 21444 ==> 21445 ==> 21446 ==> 21447 ==> 21448 ==> 21449 ==> 21450 ==> 21451
==>   621 ==> 21452 ==> 21453 ==> 21454 ==> 21495 ==> 21496 ==>   539 ==>   540 ==>   541 ==>   542 ==>
543 ==>   544 ==>   545 ==>   546 ==>   547 ==>   548 ==>   549]

StreetsMap(src: 54 dst: 549)       A* (h=0, w=0.500)         time:   0.90   #dev: 17354   |space|: 17514    total_g_cost:
7465.52560   |path|: 136   path: [   54 ==>    55 ==>    56 ==>    57 ==>    58 ==>    59 ==>    60 ==> 28893 ==> 14580
==> 14590 ==> 14591 ==> 14592 ==> 14593 ==> 81892 ==> 25814 ==>    81 ==> 26236 ==> 26234 ==>  1188 ==>
33068 ==> 33069 ==> 33070 ==> 15474 ==> 33071 ==>  5020 ==> 21699 ==> 33072 ==> 33073 ==> 33074 ==>
16203 ==>  9847 ==>  9848 ==>  9849 ==>  9850 ==>  9851 ==>   335 ==>  9852 ==> 82906 ==> 82907 ==> 82908
==> 82909 ==> 95454 ==> 96539 ==> 72369 ==> 94627 ==> 38553 ==> 72367 ==> 29007 ==> 94632 ==> 96540
==>  9269 ==> 82890 ==> 29049 ==> 29026 ==> 82682 ==> 71897 ==> 83380 ==> 96541 ==> 82904 ==> 96542
==> 96543 ==> 96544 ==> 96545 ==> 96546 ==> 96547 ==> 82911 ==> 82928 ==> 24841 ==> 24842 ==> 24843
==>  5215 ==> 24844 ==>  9274 ==> 24845 ==> 24846 ==> 24847 ==> 24848 ==> 24849 ==> 24850 ==> 24851
==> 24852 ==> 24853 ==> 24854 ==> 24855 ==> 24856 ==> 24857 ==> 24858 ==> 24859 ==> 24860 ==> 24861
==> 24862 ==> 24863 ==> 24864 ==> 24865 ==> 24866 ==> 82208 ==> 82209 ==> 82210 ==> 21518 ==> 21431
==> 21432 ==> 21433 ==> 21434 ==> 21435 ==> 21436 ==> 21437 ==> 21438 ==> 21439 ==> 21440 ==> 21441
==> 21442 ==> 21443 ==> 21444 ==> 21445 ==> 21446 ==> 21447 ==> 21448 ==> 21449 ==> 21450 ==> 21451
==>   621 ==> 21452 ==> 21453 ==> 21454 ==> 21495 ==> 21496 ==>   539 ==>   540 ==>   541 ==>   542 ==>
543 ==>   544 ==>   545 ==>   546 ==>   547 ==>   548 ==>   549]

StreetsMap(src: 54 dst: 549)       A* (h=AirDist, w=0.500)     time:   0.16   #dev: 2015    |space|: 2229     total_g_cost:
7465.52560   |path|: 136   path: [   54 ==>    55 ==>    56 ==>    57 ==>    58 ==>    59 ==>    60 ==> 28893 ==> 14580
==> 14590 ==> 14591 ==> 14592 ==> 14593 ==> 81892 ==> 25814 ==>    81 ==> 26236 ==> 26234 ==>  1188 ==>
33068 ==> 33069 ==> 33070 ==> 15474 ==> 33071 ==>  5020 ==> 21699 ==> 33072 ==> 33073 ==> 33074 ==>
16203 ==>  9847 ==>  9848 ==>  9849 ==>  9850 ==>  9851 ==>   335 ==>  9852 ==> 82906 ==> 82907 ==> 82908
==> 82909 ==> 95454 ==> 96539 ==> 72369 ==> 94627 ==> 38553 ==> 72367 ==> 29007 ==> 94632 ==> 96540
==>  9269 ==> 82890 ==> 29049 ==> 29026 ==> 82682 ==> 71897 ==> 83380 ==> 96541 ==> 82904 ==> 96542
==> 96543 ==> 96544 ==> 96545 ==> 96546 ==> 96547 ==> 82911 ==> 82928 ==> 24841 ==> 24842 ==> 24843
==>  5215 ==> 24844 ==>  9274 ==> 24845 ==> 24846 ==> 24847 ==> 24848 ==> 24849 ==> 24850 ==> 24851
==> 24852 ==> 24853 ==> 24854 ==> 24855 ==> 24856 ==> 24857 ==> 24858 ==> 24859 ==> 24860 ==> 24861
==> 24862 ==> 24863 ==> 24864 ==> 24865 ==> 24866 ==> 82208 ==> 82209 ==> 82210 ==> 21518 ==> 21431
==> 21432 ==> 21433 ==> 21434 ==> 21435 ==> 21436 ==> 21437 ==> 21438 ==> 21439 ==> 21440 ==> 21441
==> 21442 ==> 21443 ==> 21444 ==> 21445 ==> 21446 ==> 21447 ==> 21448 ==> 21449 ==> 21450 ==> 21451
==>   621 ==> 21452 ==> 21453 ==> 21454 ==> 21495 ==> 21496 ==>   539 ==>   540 ==>   541 ==>   542 ==>
543 ==>   544 ==>   545 ==>   546 ==>   547 ==>   548 ==>   549]



Process finished with exit code 0
```

12. רטוב + יבש (2 נק'): כעת נרצה לבחון את השפעת המשקל w על ריצת wA*. מלאו בקובץ main.py את המשימות הרלוונטיות לסעיף זה. בנוסף, ממשו את הפונק' run_astar_for_weights_in_range() שחתימתה מופיעה בקובץ main.py. פונק' זו מקבלת היוריסטיקה ובעיה לפתרון ומשתמשת באלג' wA* בכדי לפתור את בעיה זו תוך שימוש בהיוריסטיקה הנתונה ועם n משקולות שונות בתחום הסגור $[0.5,0.95]$. את התוצאות של ריצות אלו היא אמורה לשמור ברשימות ולאחר מכן היא אמורה לקרוא לפונק' בשם plot_distance_and_expanded_wrt_weight_figure() (שגם בה עליכם להשלים את המימוש באיזורים החסרים). פונק' זו אחראית ליצור גרף שבו מופיעות 2 עקומות: אחת מהעקומות (הכחולה) מתארת את טיב הפתרונות (בציר y) כפונק' של המשקל (אורך המסלול במקרה של בעיית המפה הבסיסית). העקומה השנייה (האדומה) מתארת את מספר המצבים שפותחו כפונק' של המשקל. עתה השתמשו בפונק' run_astar_for_weights_in_range() מהמקום הרלוונטי ב- main.py (מספר סעיף זה מצוין במקום זה) ע"מ ליצור את הגרף המתאים עבור פתרון בעיית המפה תוך שימוש בהיוריסטיקה AirDistHeuristic. צרפו את הגרף שנוצר לדו"ח. הסבירו את הגרף שהתקבל. ציינו באיזה ערך w הייתם בוחרים ולמה. בכיתה למדתם כלל אצבע לפיו "ככל ש- w קטן יותר כך הפתרון איכותי יותר ומס' הפיתוחים גדול יותר". הכלל הנ"ל מצביע על מגמה כללית, אך איננו נכון באופן גורף (כלומר ייתכנו זוג ערכים $w_1 < w_2$ עבורם הפתרון המתקבל עם w_1 פחות טוב מאשר הפתרון המתקבל עם w_2 ו/או מס' הפיתוחים עם w_2 גדול יותר ממס' הפיתוחים עם w_1). כיצד הכלל שהוזכר והדגש הנ"ל באים לידי ביטוי בתרשים שקיבלתם? על התרשים להראות כמו בדוגמה הזו (צורת העקומות עצמן עשויה להשתנות כמובן):

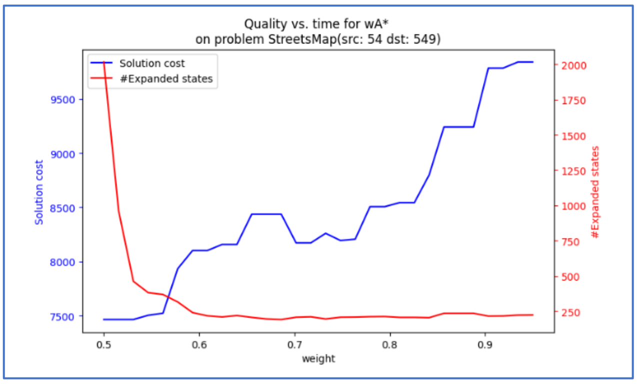


תשובה: הגרף שהתקבל מציג את ה-tradeoff הקיים באלגוריתם Weighted A*, בין איכות הפתרון לבין זמן החיפוש של הפתרון, בהתאם למשקל באמצעותו התבצעה ריצת האלגוריתם.

אני הייתי בוחר במשקל $w = 0.56$ עבורו מתקבל פתרון מעולה (בעלות נמוכה מאוד) תוך מספר נמוך מאוד של פיתוחים.

ככלל האצבע הנ"ל בא לידי ביטוי בגרף מאחר וניתן לראות שעבור המשקל הנמוך ביותר מתקבל הפתרון האיכותי ביותר (עלות פתרון נמוכה ביותר) ומספר הפיתוחים הגבוהה ביותר. זאת בנוסף למגמת ירידה בעקומה של מספר הפיתוחים (באדום) ומגמת עליה בעקומה של מחיר הפתרון (בכחול).

ניתן לראות את הדגש הנ"ל בא לידי ביטוי במספר מקומות על העקומה הכחולה, למשל $\mathrm{w}_1 = 0.65 < 0.75 = \mathrm{w}_2$ אבל עלות הפתרון המתקבל עם w_1 גבוהה יותר מאשר עלות הפתרון המתקבל עם w_2.



חלק ו' – מימוש בעיית מד"א (16 נק' יבש)

כעת נרצה לממש את המחלקה שמייצגת את מרחב המצבים של בעיית מד"א. בבעיה זו נרצה למצוא סדר אופטימאלי למעבר של האמבולנס בדירות המדווחות (לצורך לקיחת בדיקות) והעברת הבדיקות למעבדות תוך התחשבות באילוצי הבעיה כפי שתוארה בחלק ג'.

בשאלות הוכח / הפרך קבילות של היוריסטיקה: אם אתם סבורים שהיוריסטיקה קבילה יש לספק הוכחה לכך. אם אתם סבורים שהיא איננה קבילה יש לספק דוגמא של מרחב חיפוש קטן ככל שתוכלו (ציירו גרף בו הצמתים הם נקודות במפה) עבורו הערך ההיוריסטי על אחד המצבים לפחות גדול ממש מעלות הפתרון האופטימלי למטרה.

13. רטוב: התבוננו בקובץ problems/mda_problem.py והשלימו את המימושים החסרים במתודות הבאות:
    a. MDAState.__eq__()
    b. MDAState.get_total_nr_tests_taken_and_stored_on_ambulance()
    c. MDAProblem.get_reported_apartments_waiting_to_visit()
    d. MDAProblem.get_operator_cost()
    e. MDAProblem.expand_state_with_costs()
    f. MDAProblem.is_goal()

הערה: המתודה MDAProblem.get_operator_cost() אמורה לחשב את עלות האופרטור שהופעל. כזכור, בחלק ג' ציינו כי בכדי לחשב את עלות האופרטור יש לפתור בעיה על רשת הכבישים. במימוש אנחנו אכן עושים זאת. בהערות בקוד (במתודה *get_operator_cost*()) הורנו לכם להשתמש בשדה (של הבעיה) בשם *map_distance_finder* בו שמור אובייקט מטיפוס ***CachedMapDistanceFinder***, שלו יש מתודה בשם *get_map_cost_between*() המחשבת ומחזירה את עלות פתרון אופטימלי על בעיית מפות הכבישים. מאחורי הקלעים המתודה הזו למעשה אמורה ליצור בעיית MapProblem חדשה ולקרוא ל- AStar.solve_problem() בכדי לפתור אותה. אך לפני זה, לטובת היעילות, היא בודקת האם כבר פתרנו בעיה זו בעבר ואם כן מאחרת את הפתרון שדאגנו לשמור כשפתרנו בעיה זאת לראשונה ומחזירה אותו מיד וללא חישובים נוספים. במובן זה המחלקה CachedMapDistanceFinder שומרת ב- cache הפנימי שלה תוצאות של חישובים קודמים. השלימו את הקוד של המתודה *get_map_cost_between*() של המחלקה CachedMapDistanceFinder בקובץ problems/cached_map_distance_finder.py.

14. רטוב: השלימו את הקוד ב- main.py תחת ההערה הרלוונטית לסעיף זה. הריצו את הקוד הנ"ל (הרצת UniformCost על בעיית מד"א עם הקלט הקטן). המטרה היא לוודא שהקוד שרשמתם בסעיף הקודם באמת רץ בהצלחה.

15. שאלה יבש (2 נק'): בתכנות לפעמים אנחנו רוצים לכפות על מבני נתונים / טיפוסים מסוימים להיות immutable/frozen. הכוונה היא שאחרי יצירת אובייקט מטיפוס שכזה לא יהיה ניתן לשנותו. הצהרה על טיפוס כ"קפוא" מגבילה אותנו, אך יחד עם זאת היא גם מגינה עלינו. (i) העתק לדו"ח את שורת הקוד הרלוונטית שקובעת שאובייקטים מהטיפוס MDAState יהיו בלתי ניתנים לשינוי. (ii) האם שורה זו מספיקה? מה עוד מבטיח שלא יהיה ניתן לשנות בטעות את האובייקט ו/או את המבנים שהוא מחזיק? (iii) הסבר למה אנחנו רוצים לעשות זאת ספציפית עבור הטיפוס MDAState – תן דוגמא למימוש שגוי של המתודה expand_state_with_costs במחלקה MDAProblem שממחיש את הצורך בטיפוסים "קפואים". טיפ: על הבאג להיגרם מכך שבפיתון משתנה מחזיק בפועל מצביע לאובייקט ולא העתק שלו.

i.

@dataclass(frozen=True)

class MDAState(GraphProblemState)

ii. בנוסף גם המאפיינים שלו (data members) שהם מטיפוס מצביע יהיו מוגדרים להיות:

FrozenSet:tests_on_ambulance: FrozenSet[ApartmentWithSymptomsReport]

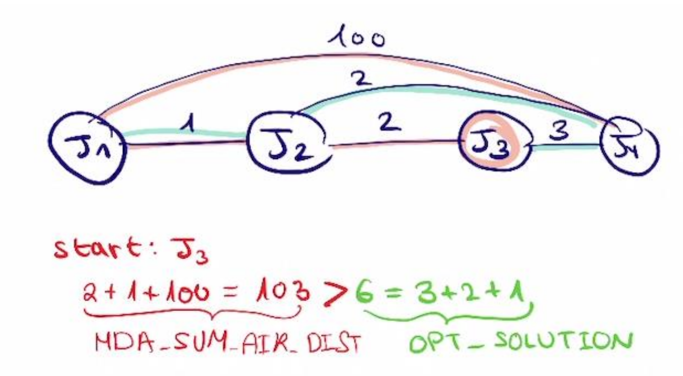
tests_transferred_to_lab: FrozenSet[ApartmentWithSymptomsReport]

visited_labs: FrozenSet[Laboratory]

iii. לא היינו רוצים לשנות את המצב הנוכחי אלא רק להוסיף מצבים לעץ החיפוש בהתאם למצב הנוכחי. אם היינו משנים את המצב הנוכחי זה היה משנה את עץ החיפוש והיה מחזיר תוצאות שגויות של חיפוש שמבוססות על מידע לא נכון, נוגד את אופן פעולת האלגוריתם.

עתה, כדי להריץ את A* על הבעיה, יש ראשית להגדיר (ולממש) היוריסטיקות עבור הבעיה.

16. רטוב: השלימו את המימוש עבור המתודה MDAProblem.get_all_certain_junctions_in_remaining _ambulance_path() (בקובץ problems/mda_problem.py). לאחר מכן, השלימו את המימוש עבור ההיוריסטיקה MDAMaxAirDistHeuristic (בקובץ problems/mda_heuristics.py). היוריסטיקה זו מתבוננת בכל הצמתים (במפת הכבישים) שיש לאמבולנס עוד לעבור בהם (כולל המיקום הנוכחי), ולוקחת את המרחק האווירי הגדול ביותר בין כל זוג מתוך קב' צמתים זו.
17. רטוב: השלימו את הקוד ב- main.py תחת ההערה הרלוונטית לסעיף זה. הריצו את הקוד הנ"ל (הרצת AStar על בעיית מד"א עם ההיוריסטיקה שמומשה בסעיף הקודם). המטרה היא לוודא שהקוד שרשמתם בסעיף הקודם באמת רץ בהצלחה ולבחון את התוצאות המתקבלות.
18. יבש (4 נק'): הוכח/הפרך: ההיוריסטיקה MDAMaxAirDistHeuristic הינה קבילה (עבור פונק' המחיר $cost_{MDA}^{dist}$). לטובת סעיף זה, הנח שכל הנקודות במפת הכבישים הן נקודות ב- $\mathbb{R}^2$ והמרחק בין זוג נק' הוא המרחק האוקלידי.
היוריסטיקה הינה קבילה.
MDAMaxAirDistHeuristic מחשבת את המרחק האווירי בין כל זוג צמתים.
לכן, כפי שלמדנו בכיתה פונקציית יוריסטיקה שמחשבת מרחק אווירי היא קבילה, מרחק המינימלי בין כל זוג נקודות - הוא כגודל המרחק האוקלידי בין 2 נקודות (קו ישר) ב- $\mathbb{R}^2$.
מכיוון שלוקחים את המרחק המקסימאלי המחושב לפי היוריסטיקה בין כל זוג נקודות שעוד לא ביקרנו בהן, לוקחים למעשה חסם עליון על המרחקים, לכן לא מאבדים פתרונות.
בנוסף יוריסטיקה זו היא מושלמת כיוון שהמרחק בפועל הוא המרחק האוקלידי, ולכן הערכת המרחק לפי היוריסטיקה שזה המרחק האווירי תהא שווה למרחק המינימאלי.
19. רטוב: השלימו את המימוש עבור ההיוריסטיקה MDASumAirDistHeuristic (בקובץ problems/mda_heuristics.py). היוריסטיקה זו מתבוננת בכל הצמתים (במפת הכבישים) שיש לאמבולנס עוד לעבור בהן (כולל המיקום הנוכחי), ומחשבת את עלות המסלול הבא: מסלול זה מתחיל בנק' הנוכחית בה נמצא האמבולנס. הנקודה ה- $i+1$ במסלול היא הקרובה ביותר לנק' i במסלול (מבחינת מרחק אווירי) מתוך כל הנק' שנותרו לביקור וטרם נבחרו למסלול זה.
20. רטוב: השלימו את הקוד ב- main.py תחת ההערה הרלוונטית לסעיף זה. הריצו את הקוד הנ"ל (הרצת Astar על בעיית מד"א עם ההיוריסטיקה שמומשה בסעיף הקודם). המטרה היא לוודא שהקוד שרשמתם בסעיף הקודם באמת רץ בהצלחה ולבחון את התוצאות המתקבלות.
21. יבש (4 נק'): הוכח/הפרך: ההיוריסטיקה MDASumAirDistHeuristic הינה קבילה (עבור פונק' המחיר $cost_{d}^{dist}$). לטובת סעיף זה, הנח שכל הנקודות במפת הכבישים הן נקודות ב- $\mathbb{R}^2$ והמרחק בין זוג נק' הוא המרחק האוקלידי.
היוריסטיקה אינה קבילה.
MDASumAirDistHeuristic מחשבת את המרחק האווירי המינימלי מכל הצמתים לצומת הנוכחי ומתקדמת לצומת עם המרחק המינימלי. ניתן דוגמה נגדית מדוע אינה קבילה:
נניח שהמרחק האווירי שווה למרחק האמיתי, נתבונן על מפה עם 4 צמתים:



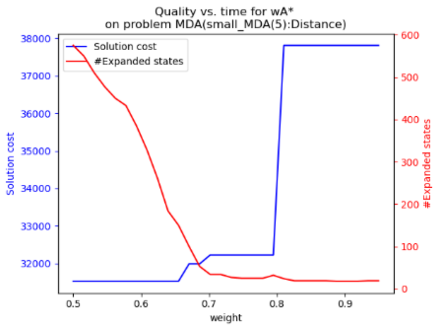
22. רטוב: השלימו את המימוש עבור ההיוריסטיקה MDAMSTAirDistHeuristic (בקובץ problems/mda_heuristics.py). היוריסטיקה זו מתבוננת בכל הצמתים (במפת הכבישים) הנותרים שעל האמבולנס לעבור בהם (מיקומי הדירות שלא עברנו בהן עדיין, כולל המיקום הנוכחי של האמבולנס וללא מיקומי מעבדות נוספות), ובונה גרף שכולל את כל צמתים אלו וקשת בין כל זוג צמתים שמשקלה מוגדר להיות המרחק האווירי בין זוג צמתים אלו. בשלב זה מחושב עץ פורס מינימלי על הגרף הנ"ל. משקל העץ שחושב הוא הערך ההיוריסטי.
23. רטוב: השלימו את הקוד ב- main.py תחת ההערה הרלוונטית לסעיף זה. הריצו את הקוד הנ"ל (הרצת AStar על בעיית מד"א עם ההיוריסטיקה שמומשה בסעיף הקודם). המטרה היא לוודא שהקוד שרשמתם בסעיף הקודם באמת רץ בהצלחה ולבחון את התוצאות המתקבלות.

24. יבש (4 נק'): הוכח/הפרך: ההיוריסטיקה MDAMSTAirDistHeuristic הינה קבילה (עבור פונק' המחיר $cost_d^{dist}$).
לטובת סעיף זה, הנח שכל הנקודות במפת הכבישים הן נקודות ב- $\mathbb{R}^2$ והמרחק בין זוג נק' הוא המרחק האוקלידי.
הוכחה:
ההיוריסטיקה MDAMSTAirDistHeuristic מחשבת עץ פורש מינימלי על הגרף המכיל את כל המשקלים והצמתים, נסמנו T1. נניח בשלילה כי היוריסטיקה אינה קבילה, אז קיים פתרון אופטימאלי השונה מהפתרון שמוחזר על ידי חישוב היוריסטיקה, עבורו קיים עפ"מ T2 :
- T2 הוא עץ פורש מכוון שעוברים בכל דירה פעם אחת, אין מעגלים, עוברים בכל הדירות בגרף.
- T2 הוא עץ פורש מינימאלי מכוון שהוא מייצג את הפתרון האופטימאלי

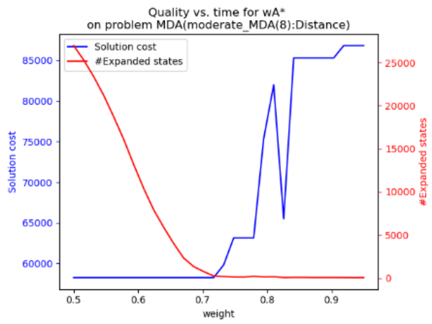
מתקיים w(T1) >w(T2), בסתירה לכך ש T1 עפ"מ.

25. רטוב + יבש (2 נק'): עתה נריץ את wA* עם ערכי w שונים כדי לצייר גרף שמציג את מגמת מחיר הפתרון מגמת מס' הפיתוחים כאשר w משתנה בתחום $[0.5,0.95]$. לצורך כך נשתמש בפונק' run_astar_for_weights_in_range() שכבר מימשנו בשלבים מוקדמים. השלימו בקובץ main.py את הקוד תחת ההערה הרלוונטית לסעיף זה. צרפו את הגרף שנוצר לדו"ח. הסבירו את הגרף שהתקבל. ציינו באיזה ערך w הייתם בוחרים ולמה.

MDAMSTAirDistHeuristic



:MDASumAirDistHeuristic



a) הסבירו את הגרף שהתקבל.
כצפוי ניתן לראות בגרפים שככל ש ה weight גדול יותר, כך מפתחים פחות צמתים אך משלמים על כך יותר ב solution cost. התוצאות תואמות את הנלמד בהרצאות ובתרגולים.
b) ציינו באיזה ערך w הייתם בוחרים ולמה?
לפי שני הגרפים ניתן לראות ש w=0.7 נותן את הפתרון האופטימאלי במבחינת מספר המצבים והן מבחינת פונ' המחיר, שזו בעצם נקודת החיתוך.

שימו לב: הסעיפים האחרונים יכולים לעזור לכם לוודא שהאלגוריתמים שלכם אכן עובדים כשורה. ודאו שהתוצאות שקיבלתם הגיוניות.

חלק ז' – מימוש והשוואת פונק' עלות שונות (25.5 נק' יבש)

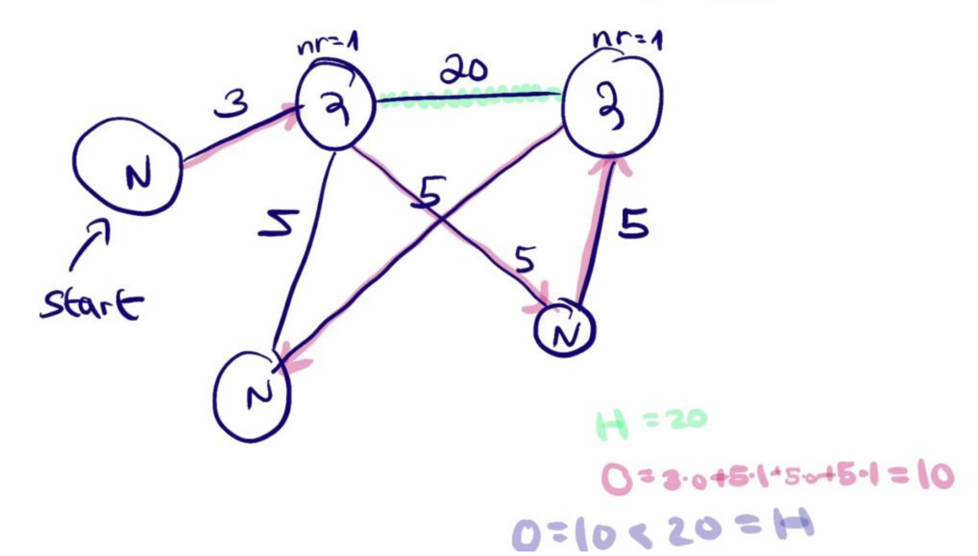
מסתבר שהמקרר באמבולנס אינו אידיאלי עבור אחסון ממושך של הדגימות. ככל שעובר יותר זמן שבו הבדיקות מאוחסנות באמבולנס (ולפני שהן עוברות לאחסון נאות במעבדה), כך יורדת אפקטיביות ואמינות הבדיקה. פונק' העלות $cost_d^{test\ travel}$ (שהוגדרה בחלק ג') מתארת את המדד הנ"ל. בחלק זה ננסה לשלב מדד זה בפתרון הבעיה.

26. יבש (2 נק'): הוכח/הפרך: ההיוריסטיקה MDAMaxAirDistHeuristic הינה קבילה (עבור פונק' המחיר $cost_d^{test\ travel}$). לטובת סעיף זה, הנח שכל הנקודות במפת הכבישים הן נקודות ב- $\mathbb{R}^2$ והמרחק בין זוג נק' הוא המרחק האוקלידי.

היוריסטיקה לא קבילה כלומר לא מעריכה אופטימאלית את מחיר המסלול מכל מצב למצב המטרה, דוגמה נגדית:

ייתכן מצב שבו המעבדה נמצאת במרחק קצר מדירה ואחריה נסיעה ארוכה עד לדירה הבאה לפי היוריסטיקה ופונ' המחיר MDACostTestsTravel תעריך את הנסיעה הארוכה כאפסית ותבחר מסלול רע יותר מאופטימאלי.

נניח שהמרחק האווירי שווה למרחק האמיתי וכי בכל דירה ישנו דייר יחיד, נתבונן על מפה עם 4 צמתים:

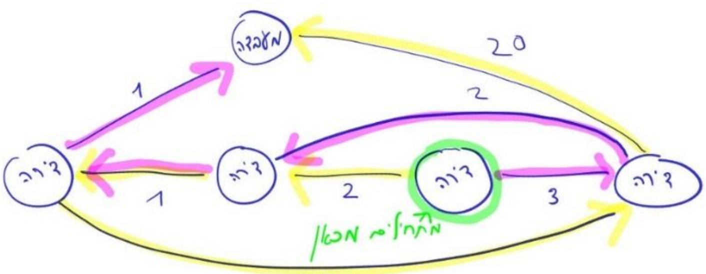


המסלול המסומן בירוק מתייחס לחישוב לפי MDAMaxAirDistHeuristic, והורוד לאופטימאלי.

הערה לבודק: ד' מסמל דירה ומ' מעבדה, בדוגמה זו ובדוגמאות הבאות, nr 1 את מספר השותפים.

27. יבש (2 נק'): הוכח/הפרך: ההיוריסטיקה MDASumAirDistHeuristic הינה קבילה (עבור פונק' המחיר $cost_d^{test\ travel}$). לטובת סעיף זה, הנח שכל הנקודות במפת הכבישים הן נקודות ב- $\mathbb{R}^2$ והמרחק בין זוג נק' הוא המרחק האוקלידי.

היוריסטיקה לא קבילה, נניח שהמרחק האווירי שווה למרחק האמיתי, נתבונן על מפה עם 4 צמתים כאשר בכל דירה דייר 1, החישוב מראה את המרחקים בפעול עם הפעלת היוריסטיקה הנ"ל:



חישוב המסלול לפי MDASumAirDistHeuristic:

2*1+1*2 +100*3+20*4 = 384

חישוב הפתרון האופטימאלי :

3*1+2*2+1*3+1*4 = 14

H = 384 > 14 = Oולכן היוריסטיקה לא קבילה.

28. יבש (2 נק'): הוכח/הפרך: ההיוריסטיקה MDAMSTAirDistHeuristic הינה קבילה (עבור פונק' המחיר $cost_d^{test\ travel}$). לטובת סעיף זה, הנח שכל הנקודות במפת הכבישים הן נקודות ב- $\mathbb{R}^2$ והמרחק בין זוג נק' הוא המרחק האוקלידי.

הפרכה:

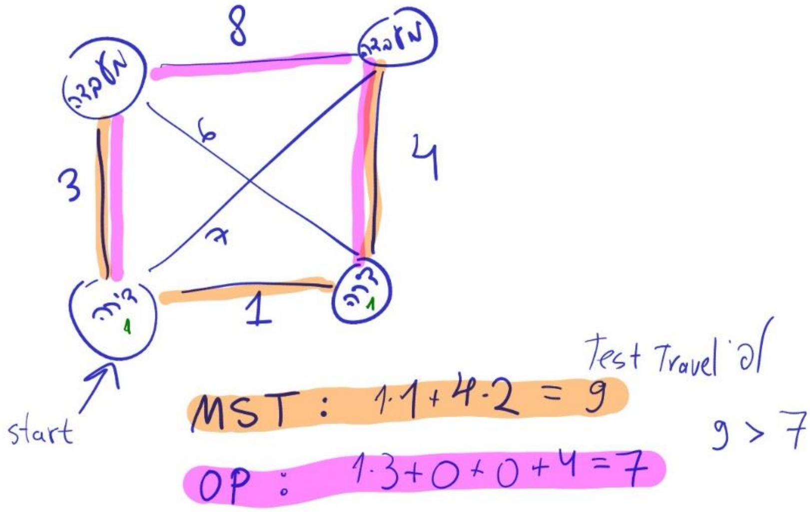
ההיוריסטיקה MDAMSTAirDistHeuristic מחשבת עבור הגרף, נסמנו G, עם המשקלים מינימליים בין כל זוג צמתים את העץ פורש המינימלי למציאת מסלול לפתרון.

נבחין כי G הוא גרף מלא מכיוון שכל קשת מייצגת את המרחק האווירי המינימאלי בין כל שני צמתים, ומכיוון שמהרחק האווירי הוא גודל פיזי – הקשת קיימת.

נתאר אתת המצב הבא:

נסמן H עבור חישוב היוריסטיקה ו O עבור הפתרון האפטימאלי

ניתן לראות בדוגמה הנגדית כי תוצאת היוריסטיקה גדולה ממש מתוצאת הפתרון האופטימאלי



הערה טכנית לגבי שימוש בפונקציות עלות שונות בקוד: כאשר פותרים את הבעיה יש לקבע פונק' עלות אחת שאיתה עובדים (היא תקבע את עלות האופרטורים והמסלולים). היינו רוצים דרך לקבוע בקוד באיזו פונק' עלות להשתמש עבור בעיית מד"א. איך זה נעשה? ה- constructor של המחלקה ***MDAProblem*** מקבל פרמטר בשם *optimization_objective* מטיפוס ***MDAOptimizationObjective*** (זהו *enum* שערכיו האפשריים הם *Distance*, *TestsTravelDistance*). העברת הערך בעת יצירת בעיה מגדיר ווריאנט של הבעיה (קובע את פונק' העלות להיות אחת מ- $cost_d^{dist}$, $cost_d^{test\ travel}$). בסעיפים הקודמים כאשר יצרנו בעיית מד"א העברנו לפרמטר *optimization_objective* את הערך *MDAOptimizationObjective.Distance* ובכך הורנו למחלקה *MDAProblem* להשתמש בפונק' העלות $cost_d^{dist}$. ערך זה נשמר באובייקט הבעיה תחת השדה *optimization_objective*. עתה, נוכל להשתמש בערך *MDAOptimizationObjective.TestsTravelDistance*.

29. רטוב: השלימו את המימוש עבור ההיוריסטיקה MDATestsTravelTimeToNearestLabHeuristic (בקובץ problems/mda_heuristics.py) בהתאם להערות המפורטות שם. היוריסטיקה זו מניחה מקרה קיצון שבו נוסעים למעבדה מיד אחרי כל ביצוע של בדיקה.

30. יבש (3 נק'):

(i) הוכח/הפרך: ההיוריסטיקה *MDATestsTravelTimeToNearestLabHeuristic* הינה קבילה (עבור פונק' המחיר $cost_d^{test\ travel}$). לטובת סעיף זה, הנח שכל הנקודות במפת הכבישים הן נקודות ב- $\mathbb{R}^2$ והמרחק בין זוג נק' הוא המרחק האוקלידי.

היוריסטיקה קבילה, עבור כל בדיקה שהאמבולנס לוקח מדירה כלשהי המרחק האופטימאלי שהבדיקה תעבור יהיה גדול או שווה מהמרחק למעבדה הקרובה ביותר לפי היוריסטיקה הנ"ל. המחיר הוא סכום מכפלת הבדיקות במרחק שהן עוברות.

(ii) מה אפשר לומר על היחס בין *MDATestsTravelTimeToNearestLabHeuristic* לבין $h^*_{test\ travel}$?

הם שווים, זוהי יוריסטיקה מושלמת כיוון שלא קיים מרחק קצר יותר שבדיקות יכולות לעבור מאשר מכל דירה שבה נאספו למעבדה הקרובה ביותר.

31. רטוב + יבש (1 נק'): השלימו בקובץ main.py את הקוד תחת ההערה הרלוונטית לסעיף זה. צרפו את התוצאות שקיבלתם לדו"ח (רק את העלויות של הפתרון ~~של שלושת הפתרונות השונים עם ציון של איזו פונק' עלות הייתה בשימוש בכל תוצאה~~). הדגישו איך רואים בתוצאות שהפתרון המתקבל אכן ממזער את המדד הרלוונטי בהתאם לפונק' העלות שהופעלה (השוו עם תוצאות מסעיפים קודמים של פתרון בעיה זו עם מדד המרחק).

הפתרון שמתקבל מהריצה של moderate_mda_problem_with_tests_travel_dist_cost עם MDATestsTravelDistToNearestLabHeuristic הוא:

total_g_cost: 104387.48471   total_cost: MDACost(dist= 137546.342m, tests-travel= 104387.485m) |path|: 18

בהרצות קודמות של בעיה זו ה tests-travel היה גדול בקירוב פי 2 לעומת הפלט של הריצה הנ"ל.

אבל ה total_g_cost של בעיות קודמות, total_g_cost: =~60000, קטן בקירוב פי 2 מאשר ה total_g_cost הנוכחי שהתקבל.

כלומר כצפוי הפתרון ממזער את מדד ה tests-travel אבל לא את עלות הפתרון הכללי, מכיוון שהיוריסטיקה נמדדת לפי המדד tests-travel.

שילוב בין 2 המדדים

נציג הצעה לשילוב בין 2 המדדים: נניח שעלות הפתרון שממזער את מדד המרחק הינו C^*_{dist}. נקבע ערך $\varepsilon > 0$. פתרון אופטימלי ע"פ <u>*הקריטריון המשולב*</u> הוא פתרון הממזער את המדד $TestsTravelDistance$ מבין כל הפתרונות האפשריים שעלות המרחק שלהם שווה/קטנה מ- $(1+\varepsilon)\cdot C^*_{dist}$.

<u>הצגה פורמלית</u>: נניח כי נתון ערך $\varepsilon > 0$ כלשהו ועבורו נגדיר את הבאים:

$$P^{I\to G}_{MDA} \triangleq \{\langle s_0,\dots,s_t\rangle | t\in\mathbb{N} \wedge s_0 = I_{MDA} \wedge \forall_{i<t}\, s_i \notin G \wedge s_t \in G_{MDA} \wedge \forall_{i\in\{1,\dots,t\}} \exists_{o\in O_{MDA}}\, o(s_{i-1}) = s_i\}$$

(זהו אוסף **כל המסלולים האפשריים** מהמצב ההתחלתי ועד מצב סופי במרחב $\mathcal{S}_{MDA}$)

$$C^*_{dist} \triangleq \min\{cost^{dist}_{MDA}(p) | p \in P^{I\to G}_{MDA}\}$$

$$DistEpsOptimal \triangleq \{p\in P^{I\to G}_{MDA} | cost^{dist}_{MDA}(p) \le (1+\varepsilon)\cdot C^*_{dist}\}$$

$$\widetilde{C^*} \triangleq \min\{cost^{test\ travel}_{MDA}(p) | p \in DistEpsOptimal\}$$

$$OptimalPaths \triangleq \{p\in DistEpsOptimal | cost^{test\ travel}_{MDA}(p) = \widetilde{C^*}\}$$

הקבוצה $OptimalPaths$ מכילה בדיוק את כל המסלולים שעונים על "הקריטריון המשולב" שהוצג מעלה.

לצורך הסעיף היבש הבא, נגדיר את הפעולה הכללית $\mathcal{P}(\mathcal{S})$ שמקבלת מרחב $\mathcal{S} = \langle S, O, I, G\rangle$ ומגדירה מרחב חדש $\mathcal{P}(\mathcal{S}) \triangleq \langle S^P, O^P, I^P, G^P\rangle$ באופן הבא:

- $S^P \triangleq \{\langle s_0,\dots,s_t\rangle | t\in\mathbb{N} \wedge s_0 = I \wedge \forall_{i<t} s_i \notin G \wedge \forall_{i\in\{1,\dots,t\}} \exists_{o\in O}\, o(s_{i-1}) = s_i\}$
- $O^P \triangleq \{o_i^P | o_i \in O\}$ כאשר $\forall_{p=\langle s_0,\dots,s_t\rangle\in S^P, o_i\in O}\, o_i^P(p) \triangleq \begin{cases} \langle s_0,\dots,s_t,o_i(s_t)\rangle & ; \quad o_i(s_t)\ne\emptyset \\ \emptyset & ; \quad otherwise\end{cases}$
- $I^P \triangleq \langle I\rangle$
- $G^P \triangleq \{\langle s_0,\dots,s_t\rangle \in S^P | s_t \in G\}$

אלג' $\mathcal{A}_1$ מריץ UCS על המרחב $\mathcal{P}(\mathcal{S}_{MDA})$ עם פונק' העלות הבאה:

$$cost\left(\underbrace{\langle s_0,\dots,s_t\rangle}_{p_1}, \underbrace{\langle s_0,\dots,s_t,s_{t+1}\rangle}_{p_2}\right) \triangleq \begin{cases} cost^{test\ travel}_{MDA}(p_2) & ; \quad cost^{dist}_{MDA}(p_2) \le (1+\varepsilon)\cdot C^*_{dist} \\ \infty & ; \quad otherwise\end{cases}$$

אם בסיום ריצת $\mathcal{A}_1$ נמצא פתרון, מוחזר המצב הסופי (במרחב $\mathcal{P}(\mathcal{S})$), שהינו למעשה מסלול – סדרה של מצבים במרחב המקורי $\mathcal{S}_{MDA}$ (באותה התצורה שמוחזר מסלול ע"י ריצת A^* על המרחב המקורי $\mathcal{S}_{MDA}$).

בסעיפים היבשים בחלק זה, הנח שכל הנקודות במפת הכבישים הן נקודות ב- $\mathbb{R}^2$ והמרחק בין זוג נק' הוא המרחק האוקלידי. אם אתם מספקים דוגמא נגדית, היא צריכה להיות קטנה ככל הניתן.

32. יבש (3 נק'): הוכח/הפרך: אם קיים פתרון במרחב (המקורי), אלג' $\mathcal{A}_1$ בהכרח מחזיר פתרון.

הוכחה: נניח בשלילה שקיים פתרון במרחב המקורי ו$\mathcal{A}_1$ מחזיר ∞ לכל מסלול כלומר לא מחזיר פתרון

אם קיים פתרון - קיים לפחות מסלול אחד p כך ש $C^*_{dist} < \infty$ ו $p \in DistEpsOptimal$ עבור $\varepsilon > 0$ כלשהו.

מספר הדירות ומספר הדיירים חסומים, כמו כן מרחקי המסלולים חסומים, לכן קיים מינימום לקבוצת $DistEpsOptimal$ לפי שיעור ה $cost^{test\ travel}_{MDA}$ שהוא $\widetilde{C^*}$.

לכן $OptimalPaths$ לא ריקה ויהי p2 מסלול כלשהו שייך לקבוצה זו.

כלומר $cost^{dist}_{MDA}(p_2) \leq (1+\varepsilon) \cdot C^*_{dist}$ ולכן קיים מסלול שמחירו סופי, והאלגוריתם יחזיר אותו בסתירה להנחה בשלילה.

33. יבש (3 נק'): הוכח/הפרך: אם אלג' $\mathcal{A}_1$ מחזיר פתרון אז הפתרון המוחזר בהכרח אופטימלי ע"פ **הקריטריון המשולב** שהוגדר מעלה.

הטענה נכונה, הוכחה: נניח בשלילה שהאלגוריתם החזיר פתרון, וקיים מסלול אחר P1 בעל עלות קטנה ממש ע"פ הקריטריון המשולב.

אם האלגוריתם החזיר פתרון P אז העלות שלו היא $cost^{test\ travel}_{MDA}(p)$ כאשר מתקיים

$cost^{dist}_{MDA}(p) \leq (1+\varepsilon) \cdot C^*_{dist}$ כלומר המסלול נמצא סביבת אפסילון של המסלול הקצר ביותר מבחינת $cost^{dist}_{MDA}$, ולכן קיים מינימום לקבוצה $DistEpsOptimal$ בה יש לפחות מסלול אחד ולכן $OptimalPaths$ לא ריקה P1 מוכל בה לכן הוא הפתרון האופטימאלי. על פי ההנחה בשלילה גם P1 הוא פתרון אופטימאלי ולכן גם הוא שייך לקבוצה $OptimalPaths$ כלומר:

$cost^{test\ travel}_{MDA}(p1) = \widetilde{C^*} = cost^{test\ travel}_{MDA}(p)$ קיבלנו סתירה להנחה שעלות של P1 קטנה ממש מעלות P.

עתה נציע את אלג' $\mathcal{A}_2$ שפועל באופן הבא:

.i הרץ A^* (עם היוריסטיקה קבילה) על המרחב $\mathcal{S}_{MDA}$ עם פונק' העלות $cost^{dist}_{MDA}$. *MDAMSTAirDistHeuristic*
.ii שמור את עלות הפתרון המוחזר במשתנה C^*_{dist}.
.iii הרץ A^* (עם היוריסטיקה קבילה) על המרחב $\mathcal{S}_{MDA}$ עם פונק' העלות $cost^{test\ travel}_{MDA}$.
MDATestsTravelDistToNearestLabHeuristic במהלך הריצה, סכום בצמתי עץ החיפוש גם את העלות $cost^{dist}_{MDA}$ בשדה נפרד. במהלך הריצה, מיד לאחר יצירת צומת חיפוש חדש, הוסף את הבדיקה הבאה: אם העלות *dist* שלו גדולה מ- $C^*_{dist} \cdot (1+\varepsilon)$, מחק את הצומת הזה ואל תוסיף אותו ל- *open*.

34. רטוב + יבש (1.5 נק'): בשלב זה נממש ונריץ את $\mathcal{A}_2$. השלימו בקובץ main.py את הקוד תחת ההערה הרלוונטית לסעיף זה. צרפו את התוצאות שקיבלתם לדו"ח. השוו בטבלה לתוצאות הריצה מסעיפים קודמים (על אותה הבעיה עם שתי פונק' עלות השונות) והראו מספרית שהפתרון המתקבל בסעיף זה אכן מקיים איזון בין שני המדדים. חשבו וצרפו לדו"ח את הערך $1 - \frac{C^*_{dist}}{DistCost(ReturnedSolution)}$.

יוריסטיקה\מדד	A2 (סעיף 34)	TestsTravelDist ToNearestLabH (סעיף 31)	SumAirDist (סעיף20)
#dev	28811	29180	26966
|space|	40722	41799	38550
total_g_cost	104387	104387	58254
MDACost(dist)	89855	137546	58254
MDACost(tests-travel)	104387	104387	131811
|path|	15	18	13

לפי הגדרה C^*_{dist} הוא הערך המינימאלי לפי פונ' המחיר MDACost(dist)

הערך $DistCost(ReturnedSolution)$ הוא הערך המינימאלי לפי פונ' המחיר MDACost(tests-travel)

$$\frac{C^*_{dist}}{DistCost(ReturnedSolution)} - 1 = \frac{58254}{104387} - 1 = -0.44$$

אכן ניתן לראות שבתוצאות של A2 ערך MDACost(dist) יצא בקירוב באמצע בין תוצאות החישוב של יוריסטיקת הTestsTravelDistToNearestLabH שמשתמשת בMDACost(tests-travel) לבין תוצאות החישוב של SumAirDist המשתמשת בMDACost(dist) .
כמו כן ניתן לראות שאורך המסלול שהתקבל מהרצת האלגוריתם הוא גם בקירוב באמצע בין אורכי המסלולים של הסעיפים האחרים. ולכן ניתן לראות שהרצת A2 נותן תוצאות שמשפרות את ערך ה MDACost(tests-travel) של תוצאות הריצה של SumAirDist.

תוצאות הרצת A2:

MDA(moderate_MDA(8):TestsTravelDistance) A* (h=MDA-TimeObjectiveSumOfMinAirDistFromLab, w=0.500) time: 38.50 #dev: 28811 |space|: 40722 total_g_cost: 104387.48471 total_cost: MDACost(dist= 89855.645m, tests-travel= 104387.485m) |path|: 15 path: [(loc: initial-location tests on ambulance: [] tests transferred to lab: [] #matoshim: 3 visited labs: []) ==(visit Raven Woolum)==> (loc: test @ Raven Woolum tests on ambulance: ['Raven Woolum (2)'] tests transferred to lab: [] #matoshim: 1 visited labs: []) ==(go to lab Bouldin-Boyland)==> (loc: lab Bouldin-Boyland tests on ambulance: [] tests transferred to lab: ['Raven Woolum (2)'] #matoshim: 5 visited labs: ['Bouldin-Boyland']) ==(visit Hana Hockman)==> (loc: test @ Hana Hockman tests on ambulance: ['Hana Hockman (2)'] tests transferred to lab: ['Raven Woolum (2)'] #matoshim: 3 visited labs: ['Bouldin-Boyland']) ==(go to lab Woolum-Mulholland)==> (loc: lab Woolum-Mulholland tests on ambulance: [] tests transferred to lab: ['Raven Woolum (2)', 'Hana Hockman (2)'] #matoshim: 7 visited labs: ['Bouldin-Boyland', 'Woolum-Mulholland']) ==(visit Gussie Foran)==> (loc: test @ Gussie Foran tests on ambulance: ['Gussie Foran (2)'] tests transferred to lab: ['Raven Woolum (2)', 'Hana Hockman (2)'] #matoshim: 5 visited labs: ['Bouldin-Boyland', 'Woolum-Mulholland']) ==(visit Veronique Katz)==> (loc: test @ Veronique Katz tests on ambulance: ['Veronique Katz (1)', 'Gussie Foran (2)'] tests transferred to lab: ['Raven Woolum (2)', 'Hana Hockman (2)'] #matoshim: 4 visited labs: ['Bouldin-Boyland', 'Woolum-Mulholland']) ==(go to lab Neri-Basta)==> (loc: lab Neri-Basta tests on ambulance: [] tests transferred to lab: ['Veronique Katz (1)', 'Raven Woolum (2)', 'Hana Hockman (2)', 'Gussie Foran (2)'] #matoshim: 8 visited labs: ['Bouldin-Boyland', 'Neri-Basta', 'Woolum-Mulholland']) ==(visit Kurt Dockstader)==> (loc: test @ Kurt Dockstader tests on ambulance: ['Kurt Dockstader (4)'] tests transferred to lab: ['Veronique Katz (1)', 'Raven Woolum (2)', 'Hana Hockman (2)', 'Gussie Foran (2)'] #matoshim: 4 visited labs: ['Bouldin-Boyland', 'Neri-Basta', 'Woolum-Mulholland']) ==(go to lab Neri-Basta)==> (loc: lab Neri-Basta tests on ambulance: [] tests transferred to lab: ['Kurt Dockstader (4)', 'Veronique Katz (1)', 'Raven Woolum (2)', 'Hana Hockman (2)', 'Gussie Foran (2)'] #matoshim: 4 visited labs: ['Bouldin-Boyland', 'Neri-Basta', 'Woolum-Mulholland']) ==(visit Pierre Lowman)==> (loc: test @ Pierre Lowman tests on ambulance: ['Pierre Lowman (3)'] tests transferred to lab: ['Kurt Dockstader (4)', 'Veronique Katz (1)', 'Raven Woolum (2)', 'Hana Hockman (2)', 'Gussie Foran (2)'] #matoshim: 1 visited labs: ['Bouldin-Boyland', 'Neri-Basta', 'Woolum-Mulholland']) ==(go to lab Lowman-Kohn)==> (loc: lab Lowman-Kohn tests on ambulance: [] tests transferred to lab: ['Kurt Dockstader (4)', 'Pierre Lowman (3)', 'Veronique Katz (1)', 'Raven Woolum (2)', 'Hana Hockman (2)', 'Gussie Foran (2)'] #matoshim: 7 visited labs: ['Bouldin-Boyland', 'Neri-Basta', 'Woolum-Mulholland', 'Lowman-Kohn']) ==(visit Krysta Valentine)==> (loc: test @ Krysta Valentine tests on ambulance: ['Krysta Valentine (3)'] tests transferred to lab: ['Kurt Dockstader (4)', 'Pierre Lowman (3)', 'Veronique Katz (1)', 'Raven Woolum (2)', 'Hana Hockman (2)', 'Gussie Foran (2)'] #matoshim: 4 visited labs: ['Bouldin-Boyland', 'Neri-Basta', 'Woolum-Mulholland', 'Lowman-Kohn']) ==(go to lab Woolum-Mulholland)==> (loc: lab Woolum-Mulholland tests on ambulance: [] tests transferred to lab: ['Kurt Dockstader (4)', 'Pierre Lowman (3)', 'Veronique Katz (1)', 'Krysta Valentine (3)', 'Raven Woolum (2)', 'Hana Hockman (2)', 'Gussie Foran (2)'] #matoshim: 4 visited labs: ['Bouldin-Boyland', 'Neri-Basta', 'Woolum-Mulholland', 'Lowman-Kohn']) ==(visit Cleora Alaniz)==> (loc: test @ Cleora Alaniz tests on ambulance: ['Cleora Alaniz (4)'] tests transferred to lab: ['Kurt Dockstader (4)', 'Pierre Lowman (3)', 'Veronique Katz (1)', 'Krysta Valentine (3)', 'Raven Woolum (2)', 'Hana Hockman (2)', 'Gussie Foran (2)'] #matoshim: 0 visited labs: ['Bouldin-Boyland', 'Neri-Basta', 'Woolum-Mulholland', 'Lowman-Kohn']) ==(go to lab Lowman-Kohn)==> (loc: lab Lowman-

35. יבש (3 נק'): הוכח/הפרך: אם קיים פתרון במרחב, אלג' $\mathcal{A}_2$ בהכרח מחזיר פתרון.
טיפ: כדי לקבל קצת יותר אינטואיציה, אתם יכולים להריץ את הדוגמא מסעיף קודם עם ערכי ε שונים.

הטענה לא נכונה, הפרכה: נרצה לייצר גרף שכולל הן מעבדות והן דירות. כך שבגרף קיימים שני מסלולים. בנוסף, הפרש מחירי המסלולים לפי MDACost(dist) הוא גדול מאפסילון (נבחר את אפסילון להיות 0.3). ואז נוצר מצב שA2 לא מוצא פתרון כיוון שבוחר את הצומת הבא לפיתוח ששייך למסלול הזול יותר מבחינת MDACost(dist), אבל שעבורו מחיר לפי MDACost(tests-travel) הוא אינסוף. בצורה הזו, למעשה המסלול השני שלא נבחר על ידי A2 הוא הפתרון.

כלומר A2 לא מחזיר פתרון למרות שקיים פתרון במרחב.

36. יבש (3 נק'): הוכח/הפרך: אם אלג' $\mathcal{A}_2$ מחזיר פתרון אז הפתרון המוחזר בהכרח אופטימלי ע"פ **הקריטריון המשולב** שהוגדר מעלה.
הוכחה : נניח בשלילה שA2 החזיר פתרון P וגם קיים פתרון טוב יותר (עם עלות קטנה יותר לפי MDACost(tests-travel) לפי הקריטריון המשולב). כלומר $P \in DistEpsOptimal$ ולכן הוא בסביבת אפסילון של הפתרון בעל העלות המינימאלית לפי $cost_{MDA}^{dist}$.
וכיוון שפתרון זה נבחר על יד A2 הוא בעל $cost_{MDA}^{test\ travel}$ מינימאלי מבין המסלולים שבסביבת האפסילון הנ"ל. ולכן הוא הפתרון האופטימאלי לפי הקריטריון המשולב בסתירה להנחה בשלילה.

37. יבש (2 נק'): ציין והסבר בקצרה יתרון של $\mathcal{A}_2$ ע"פ $\mathcal{A}_1$ במובנים של זמני ריצה.
היתרון בזמן ריצה של A2 על פני A1 מתבטא בעיקר בצריכת זיכרון. A2 יותר יעיל מבחינת סיבוכיות מקום המתבטאת בזמן הריצה של האלגוריתמים. הסיבה לכך היא ש A1 מפתח את כל הצמתים הרלוונטיים ל open ואילו A2 מפתח רק צמתים שעומדים בקריטריון C_{dist}^{*}.
בנוסף, מכיוון שבA1 נכנסים יותר צמתים ל open קיימים יותר צמתים לפתח ולבדוק מה שעולה בזמן ריצה. אך לא בהכרח יוביל לפתרון יותר טוב (מאותה סיבה שנפסלו על ידי הרקטיון כניסה ל open של A2)

חלק ח' – מימוש האלג' $A^*\varepsilon$ והרצתו (1.5 נק' יבש)

38. רטוב: ממשו את החלקים החסרים של אלג' $A^*\varepsilon$ בקובץ framework/graph_search/astar_epsilon.py ע"פ ההנחיות המופיעות שם.
39. רטוב + יבש (1.5 נק'): מימשנו היוריסטיקה קבילה (MST) והיוריסטיקה לא קבילה אך מיודעת יותר (Sum). הבעיה היא שאין לנו אף הבטחה על איכות הפתרון שמניב A* עם היוריסטיקה שאינה קבילה. נרצה לנצל את הבטחת איכות הפתרון של $A^*\varepsilon$ כדי לעשות שימוש מועיל בהיוריסטיקה שאינה קבילה במטרה לחסוך במספר הפיתוחים מבלי לפגוע באופן דרסטי באיכות הפתרון. השלימו בקובץ main.py את הקוד תחת ההערה הרלוונטית לסעיף זה. צרפו את התוצאות שקיבלתם לדו"ח (אל תצרפו את המסלולים עצמם). האם חסכנו בפיתוחים? אם כן, בכמה? הסבירו למה בכלל ציפינו מראש ש- $A^*\varepsilon$ יוכל לחסוך במס' הפיתוחים בתצורה שבה הרצנו אותו.

Results for A*eps

MDA(small_MDA(5):Distance)   A*eps (h=MDA-MST-AirDist, w=0.500)   time: 6.73   #dev: 564 |space|: 933   total_g_cost: 31528.65909   total_cost: MDACost(dist= 31528.659m, tests-travel= 52112.429m) |path|: 8

Results for A*

MDA(small_MDA(5):Distance)   A* (h=MDA-MST-AirDist, w=0.500)   time: 0.55   #dev: 575 |space|: 947   total_g_cost: 31528.65909   total_cost: MDACost(dist= 31528.659m, tests-travel= 52112.429m) |path|: 8

באלגוריתם A* פיתחנו 575 צמתים לעומת 564 ב-A*eps, בנוסף ניתן לראות שמרחב החיפוש קטן מ 947 ל933 אבל לא באופן משמעותי, סה"כ חסכנו פיתוח של 11 צמתים. לא היה שיפור לשאר המדדים.

הA*eps מפתח פחות צמתים במהלך הריצה שלו. הסיבה לכך היא ש A* מפתח את כל הצמתים הרלוונטיים ל open ואילו A*eps מפתח רק צמתים מתוך קבוצת ה focal, כאשר מספר הצמתים הנמצאים בקבוצה הזו חסום מראש על ידי max_focal_size ובנוסף הם קרובים לפתרון עד כדי אפסילון, לכן מאפשרים פיתוח של פחות צמתים מחד, וגמישות בצמתים שניתן לפתח מאידך. בעקבות שני השינויים האילה אנו מצפים לקבל מספר פיתוחים נמוך יותר ב אלג' A*eps.

חלק ט' – מימוש האלג' Anytime A* והרצתו (1.5 נק' יבש)

בסעיף זה נממש וריאציה של אלג' Anytime A*. האלג' יפעל בצורה הבאה: נריץ את אלג' wA* על הבעיה על ערכי w שונים. בכל הרצה של wA* נגביל אותו למס' פיתוחים קבוע מראש (המחלקה BestFirstSearch והאלג' היורשים ממנה יודעים לקבל ב- constructor שלהם פרמטר אופציונלי בשם max_nr_states_to_expand שעוצר את החיפוש לאחר חריגה ממספר פיתוחים זה). נבצע "חיפוש בינארי" על ערכי $w \in [0.5,0.9]$ ונחפש את הפתרון הכי טוב מבין הפתרונות המוגבל במס' הפיתוחים כאמור (ושאנו מצליחים למצוא במסגרת שיטה זו). כמו בכל חיפוש בינארי, נתחזק גבול תחתון ועליון במהלך החיפוש. הגבול העליון יאותחל להיות 0.9 והתחתון יהיה 0.5. לאורך החיפוש תישמר האינווריאנטה הבאה: לא נמצא פתרון עבור ערכי w הקטנים או שווים לגבול התחתון (במסגרת הגבלת מס' פיתוחים), אך כן נמצא פתרון כזה עבור ערך w של הגבול העליון. בכל איטרציה של החיפוש נריץ את wA* על הבעיה עם ערך w ששווה למחצית הגבול התחתון והעליון ועם מגבלת מס' פיתוחים כאמור. נעדכן את הגבולות (בהתאם לקיום או העדר של פתרון) ע"מ לשמור על האינווריאנטה. בכך בכל איטרציה נצמצם את ההפרש בין הגבולות באופן אקספוננציאלי כיאה לחיפוש בינארי. בכל מקרה, נשמור את הפתרון הטוב ביותר שנמצא עד כה ואת הערך w שהוביל אליו. נמשיך כך עד שערכי הגבולות התחתון והעליון יתקרבו זה לזה מספיק.

שימו לב: בכיתה למדתם כלל אצבע לפיו "ככל ש- w קטן יותר כך הפתרון איכותי יותר ומס' הפיתוחים גדול יותר". הכלל הנ"ל מצביע על מגמה כללית, אך ציינו בחלקים הקודמים שכלל זה איננו נכון באופן גורף. לכן כשאנו מעדכנים את הגבול התחתון, אין למעשה הבטחה אמתית שעבור כל ערכי w שקטנים

מהגבול החדש לא יימצא פתרון העונה על הדרישות. כלומר האלג' שלנו לא באמת מוצא ערך w מינימלי שמקיים את האמור, אלא הוא מנסה לקרב אותו ככל הניתן תוך הנחה על המגמה הכללית של הקשר בין w לבין מס' הפיתוחים (כלל האצבע).

הערה: ייתכן שהפתרון האופטימלי לאו דווקא הגיע מערך ה- w הקטן ביותר עבורו הרצנו wA* וקיבלנו פתרון. לכן אנו מעדכנים את המשתנה ששומר את הפתרון הטוב ביותר בזהירות (לאחר בדיקה לקיום שיפור באיכות הפתרון).

40. רטוב: השלימו את המימוש של אלג' AnytimeA* בקובץ framework/graph_search/anytime_astar.py ע"פ ההוראות המופיעות שם וע"פ ההערות שכתובות בראש המחלקה.
41. רטוב + יבש (1.5 נק'): השלימו בקובץ main.py את הקוד תחת ההערה הרלוונטית לסעיף זה. צרפו את התוצאות שקיבלתם לדו"ח (אל תצרפו את המסלולים עצמם). הסבירו איך עזר לנו להריץ את הווריאציה הזו של Anytime A* במקרה זה. מה בעצם קיבלנו? שימו לב לגודל הבעיה אותה פתרנו. חזרו לחלק א' והיזכרו כמה זמן ייקח למחשב בודד לעבור על כל הסידורים האפשריים.

Results for Anytime-A*

MDA(moderate_MDA(8):Distance) Anytime-A* (h=MDA-MST-AirDist, w=0.800) time: 2.99 #dev: 1027 |space|: 740 total_g_cost: 64055.65000 total_cost: MDACost(dist= 64055.650m, tests-travel= 131870.337m) |path|: 13

Results for A*

MDA(moderate_MDA(8):Distance) A* (h=MDA-Sum-AirDist, w=0.500) time: 15.95 #dev: 26966 |space|: 38550 total_g_cost: 58254.18667 total_cost: MDACost(dist= 58254.187m, tests-travel= 131811.935m) |path|: 13

באלגוריתם A* פיתחנו 26966 צמתים לעומת 1027 ב-Anytime-A*, בנוסף ניתן לראות שמרחב החיפוש קטן מ 38550 ל740 באופן משמעותי. עם זאת שילמנו על כך בעלות של הפתרון שגדלה מ 58254 ל 64055, מספר הצמתים במסלול האופטימאלי זהה.

היתרון המשמעותי הנוסף הוא סיבוכיות הזמן שיורדת דרסטית מ16 ל3. הסיבה לכך היא הפעלת החיפוש הבינארי שמפעיל סריקה לוגריתמית במקום ליניארית.

k	$\#possiblePaths$	$\log_2(\#possiblePaths)$	Calculation time
13	13! = 6227020800	32.53	5.8 [sec]

נבחין כי מספר הצמתים במסלול האופטימאלי הוא 13 עבורם קיימים 6227020800 שמתוכם האלג' Anytime-A* פיתח ובדק 1027 צמתים בלבד. ואכן זמן החישוב שבוצע בפועל (3 שניות) הוא בסדר גודל של זמן החישוב הצפוי (6 שניות).