236501 - מבוא לבינה מלאכותית

תרגיל בית 1: $Blind\ Search$

הסבר הקוד

הקבצים, אותם התבקשנו להגיש בחלק היבש נמצאים בתיקייה db, הם:

- $abstractSpace.pkl$
  עבור $k = 0.005$. את שאר המרחבים לא התבקשנו להגיש. ניתן ליצור אותם ע"י הרצה של הקובץ $abstract\_space\_creation.py$.
- $centrality.csv$
- $dataSet.csv$
- $experiment.csv$

הקבצים, אותם התבקשנו להגיש בחלק הרטוב נמצאים כולם בתיקייה הראשית, הם:

- $main.py$:
  מכיל $import$ של קובץ $algorithms.py$ וכן מימוש $base$ ו־$betterWaze$ (המימוש הוא בסה"כ קריאה לפונקציה המתאימה מ־$algorithms$, כמו שהתבקשנו לעשות).
- $algorithms.py$:
  מכיל את המימושים של אלגוריתמי $base$ ו־bw אשר מחזירים גם את כל המידע לו היינו זקוקים בביצוע הניסוים (כלומר חוץ ממסלול גם את מספר ה־$nodes$ שפותחו ואת מחיר המסלול). בגרסאות אלה השתמשנו בעת ביצוע הניסוי. עבור המימושים ב־$main.py$ אנו בוחרים רק את המידע אותו אנו צריכים להחזיר למשתמש (רק המסלול שנמצא).
  מכיל גם גרסאות של $base$ ו־bw אשר לא טוענות את קובץ ה־$tlv.csv$ בהן עשינו שימוש בעת ביצוע הניסוי (לחסוך טעינה מספר פעמים של אותו מידע).
- $uniform\_cost\_search.py$:
  מכיל את מימוש $Uniform\ Cost\ Search$ (בפונקציה $\_uniform\_cost\_search\_aux$, אשר מקבל מספר פרמטרים כמו פונקצית בדיקת מטרה, פונקצית מחיר, הגבלה על מספר הפתרונות וכו'. שאר הפונקציות הן גרסאות שעושות בו שימוש, ע"י הגדרה של פרמטרים אלה על מנת להתאים את חיפוש ה־UCS לצרכינו (למשל חיפוש UCS "רגיל" ממקור למטרה, חיפוש UCS המוצא מספר (שקבענו) המרכזים הקרובים ומסלול אליהם, חיפוש UCS בגרף האבסטרקטי ועוד)
- $centrality\_creation.py$:
  ניתן להריץ קובץ זה על מנת ליצור את קובץ ה־$db/centrality.csv$
- $abstract\_space\_creation.py$:
  ניתן להריץ קובץ זה על מנת ליצור את קבצי ה־$db/abstractSpace\_k.pkl$ (כאשר $k \in K$).
  זקוק לקובץ $centrality.csv$ קיים ב־db.
- $dataset\_creation.py$:
  ניתן להריץ קובץ זה על מנת ליצור את קובץ ה־$db/dataSet.csv$.
- $experiment\_creation.py$:
  ניתן להריץ קובץ זה על מנת ליצור את קובץ ה־$db/experiment.csv$.
  זקוק לקבצי $dataSet.csv$ ו־$abstractSpace\_k.pkl$ (לכל $k \in K$) קיימים ב־db.

שאלות

שאלה a

.1

הקובץ $db/tlv.csv$ מגדיר את מרחב החיפוש שלנו - $Roads$. הקובץ מייצג את רשת הכבישים של איזור תל־אביב כפי שתואר.
כל שורה בקובץ מייצגת צומת במפה, כאשר:

- אינדקס - עמודה 0:
  אינדקס הצומת. המספר נקבע ע"י מספר השורה. מהווה מזהה לצומת.
- קווי רוחב ואורך - עמודות 1,2:
  מציינים את מיקום הצומת במפה.
  - קו רוחב - עמודה 1.
  - קו אורך - עמודה 2.
- קישורים - שאר העמודות:
  שאר העמודות (3 והלאה, עד שמגיעים לתא ריק) מייצגות קישורים בין צמתים (כל עמודה מייצגת קישור אחד לכל היותר).
  על מנת להסביר את מבנה הקישור, נסמן i השורה בה אנו נמצאים (ולכן גם אינדקס הצומת הנוכחית).
  מבנה כל קישור הוא מהצורה $j@d@t$ כאשר:
  - j:
    הוא מספר צומת יעד. כלומר יש כביש (מכוון) בין i, j. כמו כן i, j שניהם מספר שלם בין 0 ל־97157 (מספר הצומת המקסימלי).
  - d:
    מייצג את אורך הקישור בין i, j במטרים.
  - t:
    מייצג את סוג הכביש. סוגי הכבישים מוגדרים בקובץ $ways/info.py$.

.2

```
Link distance: Stat(max=2288, min=1, avg=46.52155425051429)
Link type histogram: {0: 2470, 1: 4860, 2: 2460, 3: 1772, 4: 4482, 5: 643, 6: 5707, 7: 547, 8: 46837, 9: 719, 10: 4602, 11: 94838, 12: 4089}
Outgoing branching factor: Stat(max=5, min=0, avg=1.791164906646905)
Number of links: 174026
Number of junctions: 97158
```

מכיוון שמימוש $map\_statistics$ מופיע ב־$stats.py$ (אותו אנו לא מגישים), והתבקשנו להגיש כל קוד עזר שהשתמשנו בו, הנה המימוש של $map\_statistics$:

```python
def map_statistics(roads):
    '''return a dictionary containing the desired information
    You can edit this function as you wish'''

    Stat = namedtuple('Stat', ['max', 'min', 'avg'])

    number_of_links = sum(1 for link in roads.iterlinks())
    number_of_junctions = len(roads.junctions())

    return {
        'Number of junctions': number_of_junctions,
        'Number of links': number_of_links,
        'Outgoing branching factor': Stat(max(len(junction.links) for junction in roads.junctions()),
                                          min(len(junction.links) for junction in roads.junctions()),
                                          sum(len(junction.links) for junction in roads.junctions())
                                          / max(1, number_of_junctions)),
        'Link distance': Stat(max(link.distance for link in roads.iterlinks()),
                              min(link.distance for link in roads.iterlinks()),
                              sum(link.distance for link in roads.iterlinks()) / max(1, number_of_links)),
        'Link type histogram': dict(Counter(link.highway_type for link in roads.iterlinks())),
    }
```

שאלה b

הקובץ $centrality.csv$ שהתבקשנו להגיש נמצא בתיקייה db.

שאלה c

הקובץ $abstractSpace.pkl$ שהתבקשנו להגיש עבור $k = 0.005$, נמצא בתיקייה db.

שאלה d

הקובץ $dataSet.csv$ שהתבקשנו להגיש נמצא בתיקייה db.

שאלה e

.1

אלגוריתם $Uniform\ Cost\ Search$ פועל מצומת מקור יחידה. מכיוון שהגרף מכוון, ולא מובטח לנו שאם מסלול $a \sim\sim> b$ קיים אז $b \sim\sim> a$ קיים, לא ניתן להריץ את UCS מצומת המטרה. אם הגרף היה לא מכוון (או היה מובטח שלכל קשת $(a,b) \in E$ מתקיים $(b,a) \in E$) ניתן היה להריץ UCS מצומת המטרה, למצוא את המרכז הקרוב אליה (מבחינת סכום מרחקי הכבישים כפי שהוגדר), ומכיוון שהגרף כזה, מובטח לנו שהמסלול קיים גם בכיוון ההפוך והוא הזול ביותר (אחרת בשלילה היה ניתן למצוא מסלול טוב יותר ולהגיע לסתירה).

.2

במקרים בהם שלב b נכשל במציאת מסלול מ־$Junc_2$ אל B, אפשר לנסות להמשיך למצוא $Junc_2$ חדש (שונה ממה שכבר מצאנו). כתלות בצפיפות המרחב האבסטרקטי, וגודל המרחב הרגיל זה יכול לשפר את הפתרון שלנו לעומת UCS רגיל מ־A אל B.
כמו כן מספר הפעמים שמנסים למצוא $Junc_2$ חדש נתון לשינוי. כלומר ניתן לבצע ניסוים (ממש כמו בתרגיל הנפלא הזה!), לראות אם ניתן לזהות מגמת שיפור ולהסיק על ערכי חזרה אופטימליים. כמו כן ניתן לשפר זאת ע"י סימון המרכזים שבהם ביקרנו עבור כל חיפוש מ־$Junc_2$ אל $target$, מכיוון שאם היה ניתן להגיע מהם אל $target$ אז היה ניתן להגיע מ־$Junc_2$ אל $target$ (בסתירה לכך שלא הצלחנו למצוא מסלול). כלומר זה חוסך לנו בדיקות של מצבי $Junc_2$.

.3

במקרים בהם שלב c נכשל, ניתן לחזור על שלב b ואז c שוב. כמו בסעיף הקודם ניתן לבדוק את מספר החזרות (במקרה של כשלונות חוזרים) אשר יביא למצב אופטימלי.
לחילופין במקום לחזור על שלבים b, c, ניתן היה בזמן החיפוש (או פשוט לבצע הרצה נפרדת) לשמור מרכזים שהגענו אליהם וקרובים (אפשר לחשוב על הגדרות שונות לקרובים. וכן ערכים שהחל מהם מוגדר "קרוב") ל־$Junc_2$. אפשר לבחור אותם גם מספיק קרובים כדי לאפשר לנו לנצל את המסלולים האבסטרקטיים, אך מספיק רחוקים כך שהסבירות, שהם ($Junc_2$ והמרכז שאנו מחפשים) מחוברים במרחב הרגיל אך לא במרחב האבסטרקטי, תגדל. לאחר שמצאנו צומת כזאת, ניתן להריץ UCS רגיל ממנה אל $Junc_2$ ואז יש לנו מסלול מ־$Junc_1$ שעובר במרכז שמצאנו ונגמר ב־$Junc_2$. שוב כתלות בגודל המרחב האבסטרקטי וכמה המסלולים האבסטרקטיים חוסכים לנו, יתכן שיהיה שווה "לספוג" את הצעדים הנוספים שאנו עושים.

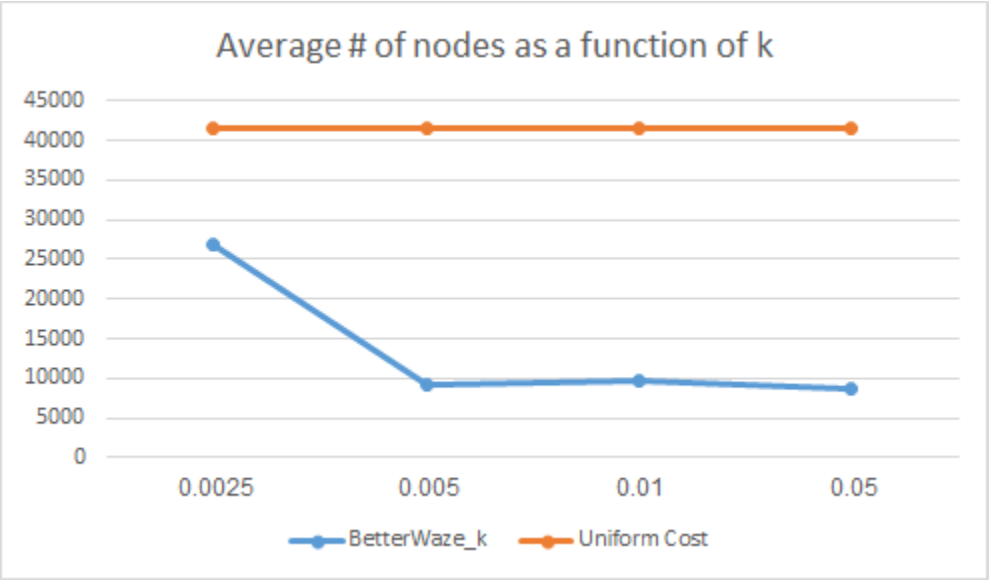
שאלה f

הקובץ $experiment.csv$ שהתבקשנו להגיש נמצא בתיקייה db.

שאלה g

יעילות האלגוריתמים

א.



ב.

BetterWaze_k	Uniform Cost	k
26800.9	41586.45	0.0025
9203		0.005
9709.6		0.01
8669.55		0.05

ג.

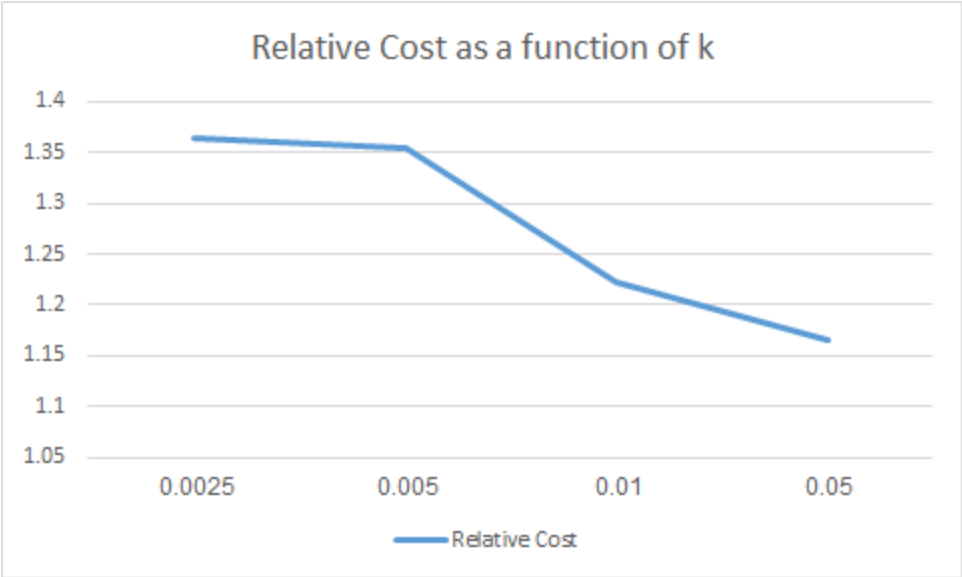
ערך ה־k (מבין הערכים שבדקנו) היעיל ביותר (מבחינת מספר הצמתים שפותחו בממוצע) הוא 0.05, עבורו קיבלנו מספר ממוצע (על הזוגות שבדקנו) של צמתים שפותחו 8669.55.

ד.

יש לנו כאן יחס בין עבודה ותמורה. באלגוריתם הבסיסי של $Uniform\ Cost\ Search$ אנו מקבלים פיתרון אופטימלי (האלגוריתם קביל עבור פונקציית מחיר חיובית. במקרה זה מרחק), אך אנו צריכים "לעבוד קשה".
כדי לחסוך עבודה, אנו מחשבים מרכזים ומסלולים ביניהם פעם אחת, ואח"כ יכולים להשתמש בהם כדי לחסוך עבודה, כשרק נותר לנו למצוא 2 מסלולים קצרים - מ־A אל $junc_1$ ומ־$junc_2$ אל B, וכמובן המסלול בין $junc_1$ ל־$junc_2$, שאת רובו כבר "יש לנו" - המסלולים שאנו "זוכרים בע"פ" ורק נותר לחבר.
כלומר ע"י חישוב ראשוני אנו חוסכים מאמץ אח"כ, ולכן ככל שנשתמש בחישובים הראשוניים שלנו יותר, נהיה יותר יעילים (עלות החישוב הראשוני תהיה זניחה).
ככל שיש לנו יותר מרכזים, ויותר מסלולים ביניהם שאנו זוכרים בע"פ, כמות העבודה שאנו חוסכים גדלה - פחות חיפושים כי צפיפות המרכזים גדולה יותר, וקל יותר למצוא מרכזים.
לכן 0.05 הערך היעיל מבין הערכים שבדקנו.
לרווח זה כמובן יש גם מחיר. פרט למחיר הראשוני שאנו משלמים ביצירת המרחבים, אנו צריכים להחזיק את המרחבים בזיכרון. כלומר צריך לזכור את כל המסלולים, ולכן אנו משלמים במחיר בזיכרון.
ככל שנגדיל את k ואת m (עד 1) כנראה שנחסוך פיתוח צמתים (אלא אם כן אנו מפספסים משהו), אך במחיר של להעמיס על הזיכרון, כך שבשלב מסוים אפשר להחליט שזה כבר לא יעיל לנו.

יעילות האלגוריתמים

א.



ב.

Relative Cost	k
1.364282318	0.0025
1.354198941	0.005
1.222036559	0.01
1.165004424	0.05

ג.

ערך ה־k (מבין הערכים שבדקנו) עבורו היחס המתקבל הוא האופטימלי הוא 0.05, יחס ממוצע של ~ 1.16.

ד.

שוב כמו בסעיף ד' עבור יעילות האלגוריתם, גם כאן יש תשלום שאנו משלמים תמורת פתרון טוב יותר. המחיר הוא אותו מחיר (חישוב ראשוני ארוך והחזקת מידע רב בזיכרון). התמורה שאנו מקבלים היא פתרון מדויק יותר.
ככל שצפיפות המרכזים עולה, עולה הסיכוי למצוא בממוצע מרכזים קרובים לצמתי המקור\יעד שלנו ולכן טובים יותר. המסלולים בין המרכזים הם אופטימלים, וככל שמשקלם במסלול גדול יותר, כך מורגשת יותר תרומתם. עבור צומת יעד למשל שהמרכז הקרוב אליה לא עובר באחד המסלולים האופטימלים, ואף אולי רחוק מאחד כזה, יגרום לכך שנסטה בהרבה ממסלול אופטימלי. לכן ככל שנעלה את ערך k, m (עד 1) כנראה שנקבל פתרון טוב יותר, עד שנגיע ל־$k, m = 1$ ונקבל מסלול אופטימלי. אך זאת כמובן במחיר כנראה גדול מדי.

חלק ד' - שאלת בונוס

- ניתן לבצע את המהלך הרנדומלי ללא חזרה על צמתים באותו מסלול.
למשל שני הצמתים המרכזיים ביותר שקיבלנו הם שני צמתים שמחוברים אך ורק אחד לשני. כלומר יש לנו כאן מעגל, שמנפח את חשיבות הצמתים האלה בשיטה שבה השתמשנו בתרגיל, למרות שבפועל אין לנו בהם שימוש כמעט.
- ניתן לחלק את המפה לריבועים ובכל ריבוע למצוא צמתים מרכזיים - אולי אוכף פיזור טוב יותר של הצמתים.
- במקום באופן רנדומלי לבחור תחילת מסלול, אפשר פשוט להתחיל מסלול רנדומי מכל צומת (ניתן לחזור מספר פעמים).
- ניתן להתחשב במידע נוסף כגון סוג איזורים - מרכז מסחרי למשל כנראה יהיה לנו חשוב יותר.
- וכו'.