בס"ד

תרגיל 1 - אלגוריתמי חיפוש

מגישים- אביתר לובטון וזוהר פוטש

מבנה הפרויקט

Node.py – מחלקה המייצגת קודקוד בעץ החיפוש. מכילה את השדות parent (קודקוד הורה), LB (חסם תחתון להגעה לפתרון אופטימלי) ו-table (מצב הלוח הנוכחי). המחלקה מכילה שיטות למציאת כל המהלכים הבאים שניתן לבצע (expand) ושיטה המחזירה את העומק של הקודקוד (depth).

EightPuzzle.py – מחלקה המייצגת את המשחק 8-puzzle. מכילה שדות עבור המצב ההתחלתי (init_state) המצב הסופי (goal_state) ופתרון (solution) ב-Node. המחלקה מכילה את השיטה solve המקבלת אלגוריתם במחרוזת והיוריסטיקה כפונקציה ומחזירה פתרון. בנוסף המחלקה מכילה שיטות סטטיות לבדיקה האם מצב התחלתי הינו פתיר (is_solvable) והגרלת מצב התחלתי רנדומלי פתיר (init_table).

heuristic.py – סקריפט המרכז את כל היוריסטיקות שמימשנו, כאשר כל היוריסטיקה מקבלת מצב נוכחי ומצב סופי כ-ndarray ומחזירה סקלר המבטא את המרחק מהפתרון.

main.py – הסקריפט הראשי של העבודה. מכיל פונקציה המקבלת קלט מהמשתמש על האלגוריתם, והפונקציה היוריסטית בהם הוא רוצה שנבצע שימוש, מגרילה לוח רנדומלי, פותרת אותו ומדפיסה את הפתרון (solve_puzzle). בנוסף מכילה פונקציות למדידת זמני ריצה של אלגוריתמים שונים (compare_algorithms) והצגת התוצאות (plot_comparison).

אלגוריתמים

Branch & Bound - תחילה נאתחל משתנים: חסם עליון- אינסוף, $open\_list$- מכיל רק את המצב ההתחלתי, פתרון- $None$. כל עוד ה-$open\_list$ אינה ריקה, נוציא $node$ הראשון ברשימה. נבצע לו expend ועבור כל האחד מה"בנים" שלו נבדוק- האם הוא פתרון סופי. אם כן אז נבדוק אם הוא יותר טוב מה-UB ונעדכן UB. אם הוא לא פתרון סופי, נבדוק האם הוא לא קיים ב-$all\_parents$ (רשימת האבות שלו) על מנת למנוע חזרות אחורה בעץ. אם הוא לא קיים נחשב לו LB. רק במידה וה- LB קטן מה- UB ניצור $node$ חדש עבור ה"בן" ונכניס אותו ל-$open\_list$ לפי השיטה המוזנת- עבור קלט "DFS" נכניס את כל הבנים לתחילת הרשימה ועבור "BFS" לסוף הרשימה. האלגו' אופטימלי מכיוון שההיוריסטיקה הינה אדמיסבלית ולכן כל קטימה היא בהכרח נכונה, כלומר לא מיועד לצאת ממצב זה פתרון טוב יותר מהפתרון הנוכחי. ולכן כאשר סיימנו לקטום את כל העץ בהכרח הפתרון אותו אנו מחזיקים הינו הפתרון הטוב ביותר.

A* - תחילה נאתחל משתנים: $open\_list$- מכיל רק את המצב ההתחלתי, פתרון - $None$, $close\_set$- ריק. עד אשר מצאנו פתרון סופי, נוציא $node$ בעל ערך f הנמוך ביותר. במידה וזהו המצב הסופי נכריז על פתרון אופטימלי ונעצור. במידה ולא, נבצע לו $expand$ ונכניס ל-$open\_list$ כל בן שאיננו נמצא ב-$close\_set$ על מנת למנוע חזרות. עבור כל בן שנכנס ל-$open\_list$ נעדכן $f = g + h$, כאשר g הינו מספר הצעדים עד ל-$node$ הנוכחי ו-h הינו ערך היוריסטיקה עבור המצב הנוכחי. האלגו' אופטימלי מכיוון שבכל איטרציה הוא שולף את ה-Node הכי טוב כרגע. ולכן במידה והלאגו' שלף Node אשר מהווה פתרון בהכרח זהו הפתרון האופטימלי כיוון שלא קיים Node פתרון בערך יותר טוב ממנו.

$\boldsymbol{Completeness}$ – שני האלגוריתמים מבצעים חיפוש על גרף סופי מכיוון שאנו מונעים חזרתיות של Node בנתיב. ולכן במידה ולא קיים פתרון האלגו' יגיע לקצה הגרף בכל הנתיבים ויכריז שאין פתרון. במידה וקיים פתרון נמצא אותו באחד הנתיבים שנפתח.

היוריסטיקות

מרחק אוקלידי - בהיוריסטיקה זאת נחשב את סכום המרחקים האוקלידיים של כל ספרה במצב הנוכחי לעומת המצב הסופי. פונקציית המרחק הינה- $\sum\sqrt{(x_1-x_2)^2-(y_1-y_2)^2}$. היוריסטיקה זו הינה אדמיסבלית מכיוון שהמרחק האוקלידי מבטא את המרחק האווירי שבין המיקום הנוכחי למיקום המטרה. מכיוון שתנועת החלקים תמיד מקבילה לאחד הצירים ניתן להסיק (מאי-שוויון המשולש) שהליכה על הצירים גדולה שווה למרחק האווירי.

מרחק מנהטן - בהיוריסטיקה זו נחשב את סכום מרחקי מנהטן של כל ספרה במצב הנוכחי לעומת מצב המטרה. פונקציית המרחק הינה- $\sum|x_1-x_2|+|y_1-y_2|$. היוריסטיקה זו הינה אדמיסבלית מכיוון שמרחק מנהטן מדמה את התנועה

האמיתית למיקום המטרה אך בחישוב זה אנו מניחים שכל צעד הינו אפשרי. הצעד אפשרי רק כתלות המיקום של התא הריק ולכן רק במקרה הכי טוב לא נידרש לצעדים נוספים.

קשיים

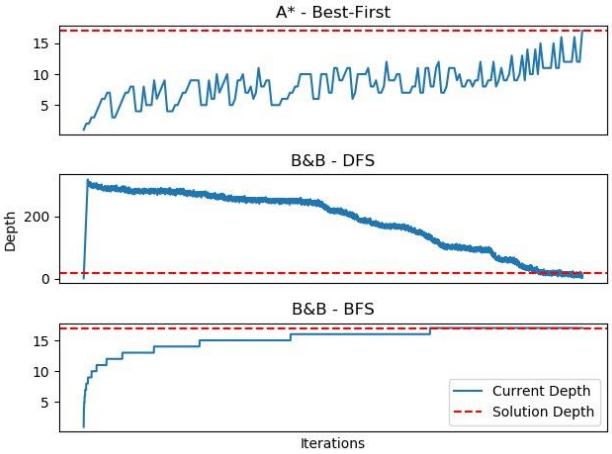
קושי: זמני ריצה ארוכים בשימוש היוריסטיקה number of misplaced אשר סופרת את המספר הספרות שלא ממוקמות במיקומן הסופי (סכום בינארי: ממוקמת 1, לא ממוקמת – 0).

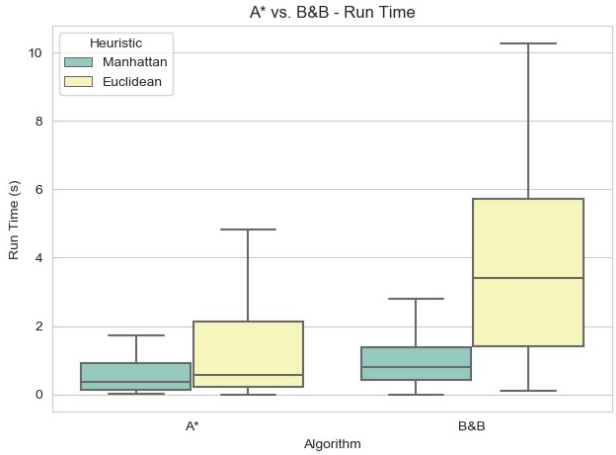
התמודדות: מימוש היוריסטיקות מרחק (אוקלידי ולאחר מכן מנהטן) אשר מספקות חסם תחתון הדוק יותר לבעיה.
ההיוריסטיקה number of misplaced סיפקה פתרון עם זמני ריצה ארוכים כיוון שהתוספת שלה לחסם התחתון קטנה ביחס להיוריסטיקות האחרות וכבל שערך החסם התחתון גבוה יותר כך נוכל לקטום מהר יותר.

קושי: הרצת האלגוריתם על מצבים התחלתיים מסוימים לא הניבו פתרון גם לאחר זמן ריצה ארוך מאוד.

התמודדות: גילנו שלא כל המצבים ההתחלתיים הינם פתירים. מימשנו פונקציה אשר בודקת את הפתרון ההתחלתי המוגרל ובמידה והוא אינו פתיר הפונקציה תמשיך להגריל פתרונות עד למציאת לוח התחלתי פתיר.

ניתוח זמני ריצה





תחילה רצינו לבצע 'בדיקת שפיות' לאלגוריתמים שממישנו. הוצאנו גרף המייצג את עומק ה-Node שבודקים בכל איטרציה והוספנו את עומק הפתרון. ניתן לראות כי ה-A* עוצר בפעם הראשונה שמגיע לעומק הפתרון, ה-DFS מתחיל עמוק והולך וקוטם את העץ עד הגעה לפתרון וקטימת כל שאר העץ, וה-BFS עולה במדרגות עד הגעה לעומק שבו נמצא הפתרון ומציאתו. בנוסף ראינו כי אלגו' ה-B&B שמבוסס על BFS מוצא מהר (מאוד) פתרונות קצרים, אך כאשר הפתרון עובר את רף ה-20/21 צעדים לוקח לו הרבה מאוד זמן. דבר זה נובע מהעובדה שבשיטת BFS נפרוס לרוחב את כל העץ ובעומק 20/21 גודל העץ נהיה גדול מדי. מכיוון שרוב הפתרונות הינם גדולים מהערכים הנ"ל החלטנו לבצע השוואה בין A* לבין B&B מבוסס DFS.

לצורך השוואת זמני ריצת האלגוריתמים עם כל אחת מההיוריסטיקות, יצרנו פונקציה אשר מריצה כל אחד מהאלגוריתמים עם היוריסטיקות ומפעילה טיימר לפני תחילת הריצה. ביצענו זאת עבור 50 מצבי התחלה שונים ופתירים (הגרף הימני).
באלגוריתם ה-B&B חיפשנו חסם עליון ראשוני הדוק יותר לצורך שיפור זמני הריצה. ראינו מאמר[1] אשר הוכיח בעזרת פריסת כל הפתרונות האפשריים כי חסם עליון לבעיה הינו 31. בעקבות כך הרצנו את אלגוריתם ה-B&B עם ערך התחלתי של $UB = 31$. ניתן לראות כי היוריסטיקת מנהטן טובה יותר מאשר מרחק אוקלידי עבור שני האלגוריתמים. תוצאות אלו מתיישבות עם הידיעה כי ערך פונקציית ההיוריסטיקת מנהטן תמיד יהיה גדול מערך ההיוריסטיקה האוקלידית. בעקבות כך ה-LB/f יקבל ערך גבוה יותר ובכך נגיע לפתרון מהר יותר. זמן הריצה של אלגו' ה-B&B ארוך יותר מכיוון שעל פי הגדרת האלגו' מספר ה-Node שיפתחו יהיה גדול יותר מאשר ב-A*.

[1] Complete Solution of the Eight-Puzzle and the Benefit of Node Ordering in IDA*, Alexander Reinefeld