**פונקציה היוריסטית #1: Distance**

מרחק X מהיציאה מוגדר כ- מספר הרכבים שחוסמים אותו.  
פונקציה זו היא אכן אדמסבילית כי היא תמיד נותנת ערך קטן או שווה לעלות האמתית.

**לוגיקה מאחורי הבחירה:** בעצם ככל שיש פחות רכבים שחוסמים את X, ככל שאנחנו מתקרבים לפתרון.

**הדגמות:**

**A**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  | X | X |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**B**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  | O |  |  |  |
|  |  | O | X | X |  |
|  |  | O |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**עיקובים אפשריים במהלך הריצה:** נסתכל במצב שבו הערימה מכילה K מצבים מהם 2 מינימליים N1, N2:

**N1**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | . | . | . |
|  |  | O |  |  | . |
|  |  | O | X | X | . |
|  |  | O |  |  |  |
|  |  |  |  |  |  |
|  |  | B | B |  |  |

**N2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | . | . | . |
|  |  | O |  |  | . |
|  |  | O | X | X | . |
|  |  | O |  |  |  |
|  |  |  |  |  |  |
|  |  |  | B | B |  |

במקרה זה בעצם האלגוריתם בוחר בצורה שרירותית מבין שני המצבים ולכן בתסריט הגרוע האלגוריתם יבחר את N1 יפתח אותו, ובכך זה יגדיל את כמות הצעדים עד להגיע למטרה (במקום לבחור את N2 ולהגיע לפתרון אחרי שני צעדים).

קוד:

**private** **void** calculate\_Fn(State node) {

**if**(node.isGoal()){

node.setFValue(-100.0);

}**else**{

**double** count = 0;

**char**[][] board = node.getBoard();

**int** CAR\_LENGTH = 2;

**for**(**int** i = node.cars.get("X").getPosition().y + CAR\_LENGTH ; i < 6;

i++){

**char** c = board[2][i];

**if**(c != 'X' && c >='A' && c <= 'Z'){

count += 10;

}

}

node.setFValue(count + node.g);

}

}

**פונקציה היוריסטית #2: Contra-Mobility**

פונקציה זו מחשבת עבור מצב כלשהו את מספר הצעדים האפשריים שאפשר לבצע על המצב, ונותנת את הערך הנגדי לו. פונקציה זו אדמסבילית (לפי הגדרה) היא תמיד נותנת ערך שלילי.

**לוגיקה מאחורי הבחירה:** ככל שיש יותר מצבים אפשריים שאפשר לעבור אליהם, ככל שעלתה דרגת החופש וככל שהתקרבנו לפתרון. על מנת לשמור על מונוטוניות (ככל שנוכל), לקחנו את המספר הנגדי כך שככל שיש יותר דרגת חופש (צעדים אפשריים) ככל שערך הפונקציה קטן.

**הדגמות (עם המקרה הגרוע שתואר עבור הפונקציה הראשונה):**

**N1**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | . | . | . |
|  |  | O |  |  | . |
|  |  | O | X | X | . |
|  |  | O |  |  |  |
|  |  |  |  |  |  |
|  |  | B | B |  |  |

**N2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | . | . | . |
|  |  | O |  |  | . |
|  |  | O | X | X | . |
|  |  | O |  |  |  |
|  |  |  |  |  |  |
|  |  |  | B | B |  |

אפשר לראות שבמקרה זה פונקציה זו מניבה מספר צעדים קטן יותר מהפונקציה הראשונה.

**עיקובים אפשריים במהלך הריצה:**

מהתוצאות שנציג למטה, מסתבר שה EBF של הפונקציה הזו גדול יותר מזה של הפונקציה הראשונה. (פירוט למטה).

**private** **void** calculate\_Fn2(State node) {

**if**(node.isGoal()){

node.setFValue(-100.0);

}**else**{

**double** count = node.nextStates.size();

node.setFValue(-count + node.g);

}

}

**פונקציה היוריסטית #3: Contra-Mobility + Distance**

פונקציה זו מבצעת ממוצע משוקלל בין שתי הפונקציות האחרות. כמו כן היא פונקציה אדמסבילית מכיוון שהיא ממוצע משוקלל של ערכים שהם קטנים מהעלות האמתית.

**לוגיקה מאחורי הבחירה:** לצורך חישובים סטטיסטיים והשוואות.

**פונקציה היוריסטית #4: Dijkstra**

**המקרה הכי מעניין, פונקציה זו תמיד נתנה פתרון עם מספר צעדים זהה לזה שבקובץ הנתון.**

**תוצאות הניסויים שביצענו:**

**ניסוי #1.1 – השוואה בין מספר הצעדים שהאלגוריתם הניב עם היוריסטיקות שתיארנו לעיל:**

**FIGURE 1**

אפשר לראות בגרף שהתוצאות הסופיות שהניב האלגוריתם בעזרת הפונקציה השנייה הן יותר טובות. מספר הצעדים היה זהה למספר הנתון עבור כל בעיה (לשמחתנו).

לעומת זאת, פונקציית המרחק (H1) הניבה תוצאות לא רעות אבל פחות טובות מאלה שהניבה הפונקציה השנייה.

**ניסוי #1.2 – השוואה בין תוצאות ה Branching Factor:**

**FIGURE 2**

אפשר לראות מתוצאות החלק הזה שה-BF של הפונקציה הראשונה (Distance) קטן יותר מזה של הפונקציה השנייה (Contra-Mobility). וזה בגלל העובדה שה- Contra-Mobility נותנת ערכים שליליים וזה גורם להימצאות המון צמתים עם ערך שווה בתוך הערימה של ה-Opened-List.

כמו כן, אפשר לראות בשתי הרגרסיות שה- Branching factor הולך וקטן ככל שהרמה עולה ובפרט בסוף הגרף רואים שהרגרסיה של הפונקציה השנייה (Contra-Mobility) יורדת אפילו מתחת לזאת של הפונקציה הראשונה – וזה מעניין.

אז למה רואים שה-BF הולך ונהיה שווה כמעט בשתי הפונקציה ככל שהרמה עולה?

תשובה: זה בגלל שהפונקציה השנייה (Contra-Mobility) נותנת מספר h: . כלומר בבעיות שדורשות מספר צעדים קטן יחסית אז יהיו המון התנגשויות (בערכים) בתוך הערימה וזה גורר פיתוח מצבים שהם לא בהכרח הכי טובים. אבל ככל שעולים ברמה וככל שמספר הצעדים הקטן ביותר שדרוש לפתור רמה מסוימת עולה אז גם הפונקציה מתחילה להתייצב והערכים בתוך הערימה מתחילים להתרחק זה מזה.

**פירוט נוסף:**

**נזכור ש-** Contra-Mobility בעצם נותנת את המספר הנגדי של כמות הצעדים עבור מצב מסוים,

***– כמות הצעדים שעשינו מהשורש (ביחידות של 1).***

**ולכן בבעיות קלות שדורשות מספר קטן של צעדים כלומר כאשר ה קטן מ- 10 אז עלולים לקבל ערכים שווים בחישוב של .**

**הערה: מספר 10 זה כמות צעדים טיפוסית מקסימלית (בערך) עבור מצב מסוים. יכול להיות מצב שיש בו יותר אבל אנחנו מסתכלים רק על מקרים מעניינים.**

**ניסוי #2.2 – שילוב בין שתי הפונקציות:**

בגלל התוצאות שקיבלנו לעיל, תהינו, מה אם נערבב בין שתי הפונקציות ונבחן את התנהגות השילוב. ולכן חיברנו פונקציה חדשה שהיא בעצם ממוצע משוקלל של שתי הפונקציות שהזכרנו לעיל + הוספת ערך שנדבר עליו בהמשך.

וכמו שהזכרנו לעיל:

בעצם, עם הוספת פונקציה זו, אפשר לעשות אינספור קומבינציות. בחנו כמה מקרים ובסוף בחרנו שלושה מהם:

אפשר לראות בתוצאות שלהלן:

**FIGURE 3**

**FIGURE 4**

* רואים שהבחירה השלישית בעצם נתנה ביצועים יותר טובים מהשתיים הראשונות.
* אמנם מספר הצעדים (באיור 4) קצת יותר מהפתרון הנתון אבל ה- BR כן ירד ביחס לתוצאות של הפונקציה .

**נבחר את הפונקציה עם הקומבינציה השלישית בתור , ולסיום – השוואת הביצועים של שלושת הפונקציות:**