אלגוריתם A*

- האלגוריתם best-first אינו מגביל אותנו בבחירת הפונקציה המשמשת להערכת צמתים
- האלגוריתם יכול להסתכל רק קדימה על המרחק הצפוי למטרה (הפונקציה היוריסטית h)
- האלגוריתם יכול להסתכל רק אחורה על המחיר של המסלול שכבר עבר. (במקרה כזה נקבל למעשה חיפוש uniform-cost)

פונקצית ההערכה של A*

- אלגוריתם A* הינו למעשה best-first המתחשב בדרך שכבר עשה וגם בדרך הצפויה כדי להעריך צומת
- האלגוריתם מעריך צמת n באמצעות הפונקציה f(n)=g(n)+h(n) כאשר g(n) הינו מחיר המסלול הקצר ביותר לצומת n שמצאנו עד עתה, ו h(n) הינו המרחק המוערך למטרה.
- אם הפונקציה h(n) תמיד אופטימית (כלומר מעריכה תמיד שהצומת n קרוב למטרה יותר ממרחקו האמיתי), ניתן להוכיח ש- A* מחזירה פתרון בעל מחיר מינימלי.

הסכם

מעתה, כאשר נדבר על best-first , הכוונה תהיה לכזו המשתמשת רק בפונקציה h להערכת צמתים. כלומר, נתכוון לפרוצדורת חיפוש אופורטיוניסטית שמטרתה להגיע למטרה מהר ככל האפשר ללא התחשבות באורך הפתרון.

ביצועי A*

בעיה	סוג חיפוש	פותחו	זמן	אורך פתרון
1	uniform	658	25	11
	best	14	0.06	11
	hill	137	1	137
	A*	14	0.05	11
2	uniform	160	1.88	9
	best	14	0.033	11
	hill	42	0.21	לא נמצא פתרון
	A*	15	0.07	9
3	uniform	5041	1324	15
	best	129	1.4	71
	hill	300	2.66	לא נמצא פתרון
	A*	26	0.11	15
4	uniform	4618	1073	15
	best	361	9.05	109
	hill	195	1.63	לא נמצא פתרון
	A*	94	0.85	15
5	uniform	1623	143.4	13
	best	19	0.08	15
	hill	40	0.18	לא נמצא פתרון
	A*	68	0.41	13

אלגוריתם A*

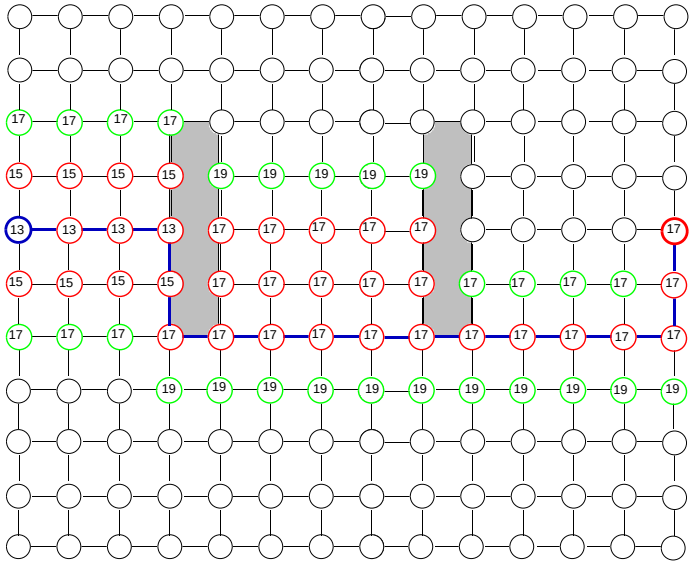
נתונים

1. מצב התחלתי S_i
2. פרדיקט לבדיקת מצב סופי $G(s) = TRUE \Leftrightarrow s \in S_g$
3. פונקצית מעבר $succ : S \to 2^S$ מקבלת מצב ומחזירה קבוצת מצבים
4. פונקצית מחיר $\cos t : \{\langle s_1, s_2 \rangle | s_1 \in S, s_2 \in succ(s_1)\} \to \Re^+$ החסומה מלמטה.

   כלומר, קיים $\delta > 0$ המקיים $\forall s \in S, \forall s' \in succ(s)[\cos t(s, s') \geq \delta]$
5. פונקציה יוריסטית $h : S \to \Re^+$

   מקבלת מצב ומחזירה הערכה של מחיר המסלול המינימלי (סכום מחירי הקשתות) למצב כלשהוא בקבוצת המטרה


```
ASTAR(state)
    OPEN ← (node(state,NIL,0,h(state),h(state)))    ; Definition: Node(state,parent,g,h,f)
    While OPEN ≠ ( )
        next ← pop(OPEN); push (next,CLOSE)
        if goalp(next.state) then return(trace(next))
        for s in succ(next.state)
            new-cost←next.g+cost(next.state,s)
            old-node ← find-state(s,OPEN)
            if old-node ≠ {} then
                if old-node.g > new-cost then
                        old-node.g ← new-cost; old-node.f ← old-node.g+old-node.
                        old-node.parent ← next; delete(old-node,OPEN);
                        insert(old-node,OPEN,f)  ; Reinsert old node according to new f
                end
            else old-node ← find-state(s,CLOSE)
                if old-node ≠ {} then
                    if old-node.g > new-cost then
                        old-node.g ← new-cost; old-node.f ← old-node.g+old-node.
                        old-node.parent ← next; delete(old-node,CLOSE);
                        insert(old-node,OPEN,f)
                end  else insert(node(s,next,new-cost,h(s),h(s)+new-cost),OPEN,f)
        end; end; return FAIL
```

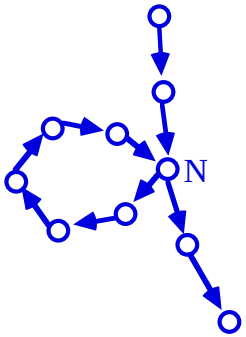



תכונות פורמליות של A*

- האלגוריתם עוצר על גרפים סופיים.
- האלגוריתם שלם על גרפים סופיים.
- האלגוריתם שלם גם על גרפים לא סופיים.
- האלגוריתם קביל [מחזיר פתרון בעל מחיר מינימלי] בתנאי ש-h קבילה.

A* עוצר על גרפים סופיים

האפשרות היחידה לאי עצירה היא מסלול מעגלי



אולם אין אפשרות ש- n יעבור מ-CLOSE ל-OPEN שכן מחיר המעגל גדול מ-0.

A* שלם על גרפים סופיים

(אלגוריתם חיפוש נקרא שלם אם הוא עוצר עם פתרון כאשר כזה קיים)

- נניח כי קיים פתרון, אבל A* נכשל
- A* מחזיר כשלון רק כאשר הרשימה OPEN ריקה.
- יהי n הצומת האחרון על מסלול לפתרון שהיה ב-OPEN (לפחות אחד כזה, S_i נכנס ל- OPEN)
- ל- n אין ילדים (אחרת הם היו ב OPEN)
- אבל לכל צומת על מסלול הפתרון יש לפחות ילד אחד (מלבד אולי לאחרון השייך ל Sg).
- סתירה

A* שלמה גם על גרפים אינסופיים

- אם קיים פתרון ו A* עוצרת (ללא פתרון) אזי קיימת סתירה כמו בהוכחה הקודמת
- אם אינה עוצרת משמע שהיא צועדת במסלול אינסופי.
- נזכור כי פונקצית המחיר חסומה מלמטה ע"י $\delta > 0$.
- קיים ב-OPEN לפחות צומת אחד n מהמסלול האופטימלי.
- לאחר מספר מספיק של צעדים על המסלול האינסופי ($\frac{f(n)}{\delta}$) המחיר המצטבר יהיה מספיק גדול כדי ש-A* תבחר ב n לפיתוח.

קבילות של A*

הגדרה:

אלגוריתם חיפוש הוא **קביל** (admissible) אם מובטח שהוא ימצא פתרון אופטימלי (בעל מחיר מינימלי) כאשר קיים פתרון.

סימונים:

$g^*(n)$ - המחיר הנמוך ביותר של מסלולים מהמצב ההתחלתי S_i למצב n.

$h^*(n)$ - המחיר הנמוך ביותר של מסלולים ממצב n למצב כלשהוא ב S_g

משתי הגדרות אלו נובע:

$C^* = h^*(S_i)$ - מחיר אופטימלי (מינימלי) לפתרון

$f^*(n) = g^*(n) + h^*(n)$ -

מחיר מינימלי של מסלול מ S_i למצב כלשהוא ב S_g (כאשר המינימום נלקח מעל כל המסלולים העוברים דרך n) .

פונקציות יוריסטיות קבילות

הגדרה:

פונקציה יוריסטית נקראת **קבילה** אם לכל n מתקיים:

$$0 \le h(n) \le h^*(n)$$

(h תמיד אופטימית בהערכת המחיר למטרה)

כמובן ש $h^*(s)$ לכל s בקבוצה S_g על כן פונקציה קבילה מקיימת בהכרח h(s)=0 לכל מצב סופי.

דוגמאות ליוריסטיקות קבילות

- עבור בעיות מפה, המרחק האוירי הינה יוריסטיקה קבילה
- עבור 8-puzzle , מספר הקוביות שאינן במקומן הינה יוריסטיקה קבילה. גם סכום-מרחקי-מנהטן הינה יוריסטיקה קבילה.
- עבור בעית הקניבלים והמסיונרים מספר האנשים בגדה השמאלית (פחות אחד) הינה יוריסטיקה קבילה.

מעתה ואילך אנו מניחים ש-h היא יוריסטיקה קבילה

- אנו נרצה להוכיח כי כאשר h קבילה, A* מוצאת פתרון אופטימלי.

למה 1

בכל שלב לפני ש A* עוצרת, קיים ב-OPEN צומת n השייך למסלול אופטימלי המקיים $f(n) \le C^*$

הוכחה:

1. יהי $P^* = n_1, \cdots, n_k$ מסלול אופטימלי כלשהוא כאשר $n_1 = S_i \;\&\; n_k \in S_g$
2. קיים לפחות צומת אחד ב- P* הנמצא ב OPEN.
   (אחרת היו כולם ב - CLOSED אבל מצב מטרה נכנס ל-CLOSED רק כשהאלגוריתם עוצר)
3. יהי n_j הצומת ב OPEN ∩P* הקרוב ביותר ל S_i (יתכן ויהיה יותר מצומת אחד השייך למסלול כיוון שצמתים יכולים לחזור מ-CLOSED ל-OPEN).
4. $n_1, \cdots, n_{j-1}$ נמצאים כולם ב CLOSED . [נניח שלא כולם ב-CLOSED. נסתכל על הצומת n_k במסלול $n_1, \cdots, n_{j-1}$ הרחוק ביותר מ S_i כך ש $n_1, \cdots, n_k$ ב-CLOSED. n_{k+1} חייב להיות ב-OPEN בסתירה לכך

ש-n_j הוא הצומת ב-OPEN הקרוב ביותר ל- S_i]

5. $n_1,\cdots,n_{j-1}$ אופטימלי (כיוון שהינו חלק ממסלול אופטימלי)

6. מ-4 ו-5 נובע ש $g(n_j)=g^*(n_j)$.

[6 אינו מידי. נראה לכאורה שיתכן סדר פיתוח צמתים ב- $n_1,\cdots,n_{j-1}$ שיגרום לכך שאחד ההורים לא יהיה מעודכן. נניח שקיים $n_{i'}$, $i'\le j-1$ שמחירו אינו מעודכן. זה יתכן רק אם קיים $n_{i''}$, $i''\le i'$ שעדיין לא פותח כאשר $n_{i'}$ נוצר. אולם, כאשר $n_{i''}$ יפותח יעודכן המחיר לבא אחריו והוא יכנס מחדש ל-OPEN.]

7. $$f(n_j)=h(n_j)+g(n_j)=h(n_j)+g^*(n_j)\le$$
$$h^*(n_j)+g^*(n_j)=f^*(n_j)=C^*$$

משפט 1: A* קביל.

- נניח בשלילה ש- A* עצר, ומצא מסלול פתרון (המגיע ל $s_g\in S_g$) עם מחיר לא אופטימלי כלומר
$$f(s_g)=g(s_g)>C^*$$
- A* בודק תנאי סיום רק של הצומת המועמד לפיתוח וזה תמיד בעל f מינימלי ב OPEN לכן $\forall n\in OPEN\left[f(s_g)\le f(n)\right]$
- לכן, לפני הבדיקה (לפני ש- A* הסתיימה) היה קיים מצב בו לכל n ב- OPEN מתקיים $\forall n\in OPEN\left[f(n)\ge f(s_g)>C^*\right]$ בסתירה ללמה 1.
- על כן הנחת השלילה אינה נכונה ובהכרח מתקיים $f(s_g)=g(s_g)=C^*$, כלומר, A* מוצאת תמיד פתרון אופטימלי.

הערות

- A* מבטיח מציאת מסלול אופטימלי - השאלה שנותרת היא באיזו מהירות.
- הפונקציה היוריסטית h קובעת את מהירות החיפוש. h מושלמת תהיה שקולה ל- h*. במקרה כזה A* ירוץ ישר למטרה.
- h=0 (חסרת אינפורמציה לחלוטין) תגרום לחיפוש עוור (Uniform-cost).
- בד"כ, h נמצאת בין h* ל-0.

השוואת יוריסטיקות

הגדרה∫

נאמר שפונקציה יוריסטית h_2 יותר מיודעת (more informed than) מפונקציה יוריסטית h_1 אם שתיהן קבילות, ולכל n שאינו מטרה $h_2(n)>h_1(n)$

- שימו לב ששויון מותר רק לגבי מצבי מטרה, לכן, לדוגמא, הפונקציה סכום-מרחקי-מנהטן אינה יותר מיודעת מהפונקציה מספר-המשבצות-שאינן-ממוקמות.
- משפט: A* המשתמש ב-h_1 יפתח כל צומת שיפתח A* המשתמש ב-h_2 (כלומר שימוש ביוריסטיקה יותר מיודעת מביא לחיפוש יעיל יותר).

יוריסטיקה מונוטונית

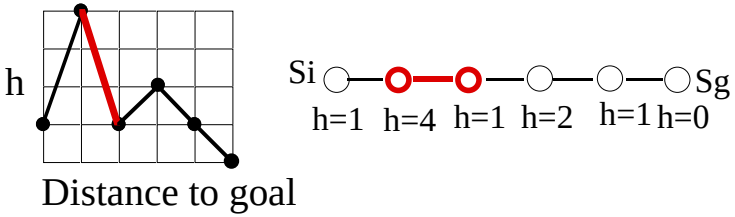
- פונקציה יוריסטית h תקרא מונוטונית אם

$$\forall s \in S \forall s' \in SUCC(s)$$

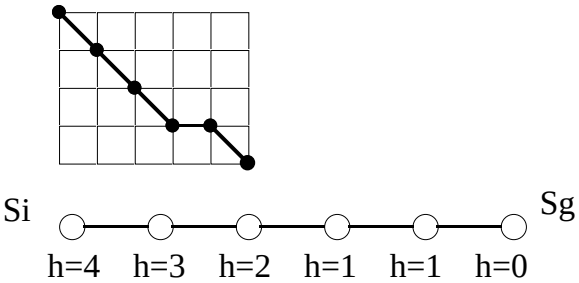
$$[h(s) - h(s') \le COST(s,s')]$$

- כלומר, איננו מסתפקים בכך שהפונקציה היוריסטית תהיה אופטימית באופן גלובלי אלא אנו דורשים שהיא תהיה אופטימית לוקלית ביחס לכל צעד בדרך למטרה.

- דוגמא לפונקציה יוריסטית קבילה שאינה מונוטונית:



- דוגמא לפונקציה יוריסטית קבילה מונוטונית:



פונקצית f מונוטונית

- במצב ההתחלתי $f(s_i) = h(s_i)$
- במצב הסופי $f(s_g) = h^*(s_i) \ge h(s_i) = f(s_i)$
- איך תתנהג f בין שני המצבים?
- שימוש ביוריסטיקה מונוטונית מבטיח שהפונקציה f תהיה מונוטונית עולה

COST(n, n') + h(n') ≥ h(n)

f(n')=h(n')+g(n')=

g(n) + $\boxed{\text{COST(n,n') + h(n')}}$ ≥

g(n)+h(n)=f(n)

⇓

f(n')≥f(n)

- פירוש הדבר שהפונקציה תתנהג בצורה חלקה - לא יהיו "בורות" במהלך החיפוש (ועל כן החיפוש קל יותר - אין מינימום לוקלי)
- ניתן להוכיח שחיפוש כזה לא יוביל אותנו לצמתים בדרכים שאינם אופטימליות

משפט 3:

עבור A* המשתמש ב-h מונוטונית מתקיים:

$$\forall n \in CLOSED[g(n) = g^*(n)]$$

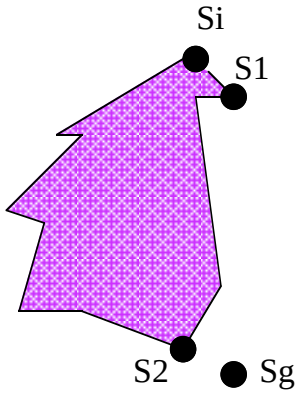
(כלומר מובטח שכל צומת שפותח מצביע למסלול האופטימלי למצב ההתחלה)

מסקנה: אין צורך להעביר צמתים מ- CLOSED ל- OPEN ופיתוחם מחדש נחסך.

- קיים משפט האומר כי עבור יוריסטיקה מונוטונית, A* היא גם אופטימלית במשאבי חיפוש [לא ניתן לוותר על פיתוח צומת שהיא מפתחת ולהבטיח קבילות]

החלשת דרישת האופטימליות

- במקרים רבים דרישת האופטימליות חזקה מידי
- A* מבזבז זמן רב כדי להבטיח שהפתרון שימצא אכן אופטימלי אולם פעמים רבות מספיק לנו פתרון קרוב לאופטימלי.
- פותחו מספר שיטות לחיפוש פתרונות טובים שאינם בהכרח אופטימליים.



g(S2)=10
g(S1)=1
h(S2)=1
h(S1)=9
f(S2)=11
f(S1)=10

חיפוש עם משקלות

- ללא h, A* הוא חיפוש לרוחב (או Uniform-cost)
- תפקידה של h להסתכל קדימה (מוסיפה מימד של Best-first) על מנת לבחור צמתים קרובים למטרה.
- ללא g, A* אינו מתחשב ב"היסטוריה" של הצמתים.
- תפקידה של g לדאוג ש A* יפתח צמתים על מסלולים קצרים.
- בחיפוש עם משקלות אנו מעניקים לאחת משתי הפונקציות משקל גדול יותר בהערכת צמתים.

פונקצית הערכה עם משקלות

$$f_w(n) = (1-w)\cdot g(n) + w\cdot h(n)$$
$$0 \le w \le 1$$

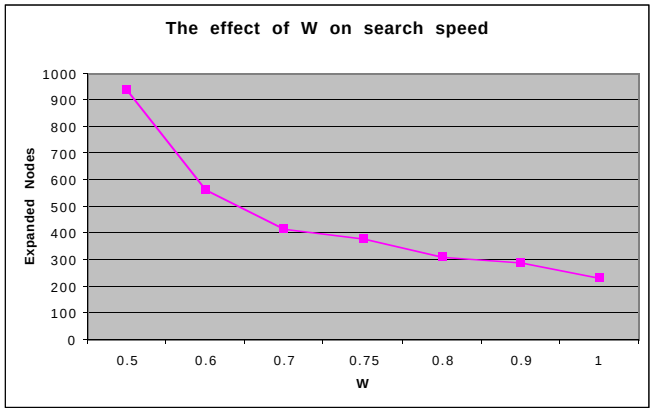
w=0 פירושו חיפוש Uniform-cost
w=0.5 פירושו חיפוש A* רגיל
w=1 פירושו חיפוש Best-first

קביעת w בין 0 ל-1 מאפשרת למזג מינון מדויק של "שמרנות" (w קטן) ו "העזה" (w גדול)

- ניתן להוכיח ששימוש ב f_w קביל עבור h קביל ו- $0 \le w \le 0.5$.
- עבור $w > 0.5$, שימוש ב- f_w אינו בהכרח קביל גם אם h קבילה.

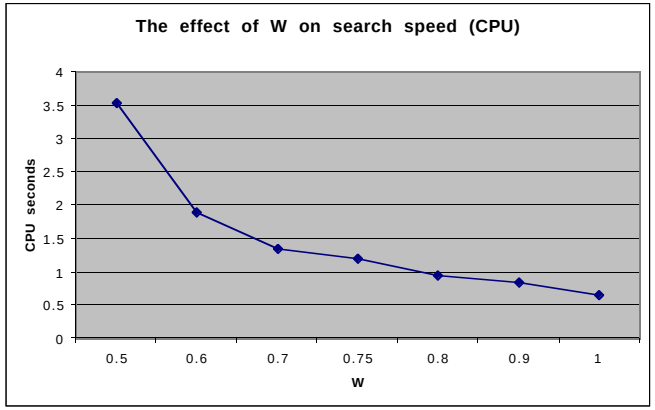
חיפוש עם משקלות - תוצאות ניסוייות

- קבוצה של 1000 בעיות פאזל 3×3.
- יוריסטיקה: סכום מרחקי מנהטן.

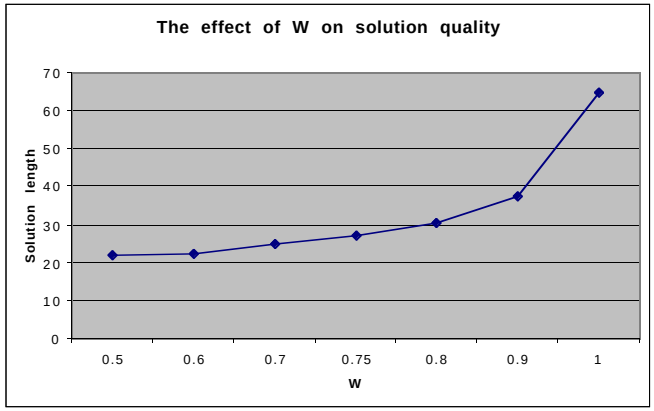


- כצפוי, ככל שהמשקל עולה, החיפוש מהיר יותר.

- מדידת מהירות באמצעות זמן ריצה אינה אמינה, אולם התוצאות כאן הינן דומות מאוד.



איכות הפתרון (אורכו) מתנהגת כצפוי בצורה הפוכה



מציאת פתרונות כמעט אופטימליים

- פעמים רבות איננו דורשים בהכרח פתרון אופטימלי אולם אנו רוצים להיות בטוחים שהפתרון אינו גרוע בהרבה מהפתרון האופטימלי.
- ויתור על דרישת האופטימליות יחסוך משאבי חיפוש.

הגדרה: יהי C^* המחיר האופטימלי של הפתרון ויהי $\varepsilon \geq 0$. אלגוריתם שמבטיח מציאת פתרון בעל מחיר של לכל היותר $(1+\varepsilon)C^*$ נקרא e-קביל

משקלות דינמיים

- יהי N חסם עליון על עומק צמתי הפתרון (עומק במובן מספר קשתות בגרף). נגדיר $d(n)$ כעומק הצומת n (מספר הקשתות במסלול מהשרש).
- נגדיר:

$$f_{dw}(n) = g(n) + h(n) + \varepsilon(1 - \frac{d(n)}{N})h(n)$$

- משפט: אם h קבילה אזי A* שמשתמש ב f_{wd} הינו ε-קביל, כלומר ימצא פתרון בעל מחיר של לכל היותר

(1 + ε)·C*

- האלגוריתם מתחיל עם משקל גדול יותר ל-h.

(1 + ε)

- כאשר עומק החיפוש גדל דואג האלגוריתם למנוע הסתבכויות בענפים עמוקים מדי ע"י הקטנת משקלו היחסי של h

A*ε

- אלגוריתם שעובד בצורה דומה ל-A*. ההבדל הוא בבחירת הצומת הבא לפיתוח.
- נגדיר:

$$focal = \left\{ n \in OPEN \,\middle|\, f(n) \le (1+\varepsilon) \min_{n' \in OPEN} f(n') \right\}$$

- כלומר, $focal$ היא קבוצת כל הצמתים ב-OPEN שטובים כמעט כמו הצומת האופטימלית (עד כדי ε)
- האלגוריתם אינו מכתיב את מי מהקבוצה לבחור.
- ניתן להראות שהפתרון שימצא ע"י A*ε לא יחרוג מפתרון אופטימלי ביותר מאשר בפקטור של $+\varepsilon$ (כלומר A*ε הוא ε-קביל)
- אסטרטגיה סבירה לבחירה מתוך $focal$ הינה בחירת הצומת בעלת ה-h המינימלי.

משקל דינמי לעומת A*ε

- שני האלגוריתמים מאפשרים שליטה על סטיה מהפתרון האופטימלי
- יתרון של A*ε - אינו דורש ידע מוקדם על עומק הפתרון.
- יתרון של A*ε - מאפשר שימוש ביוריסטיקה נוספת המעריכה את מאמצי החיפוש (במקום את המרחק למטרה).

Iterative-deepening A*

- אחד מחסרונותיו של A* הוא דרישת הזכרון שלו
- האלגוריתם הבא הינו שיפור של iterative deepening המאפשר שימוש ביוריסטיקה ומבטיח פתרונות קבילים.
- דרישות הזכרון של האלגוריתם הינן צנועות (לינאריות במחיר הפתרון האופטימלי)

אלגוריתם IDA*

1. קבע את h(Si) כעומק מקסימלי
2. ערוך חיפוש לעומק. עצור בכל ענף עבורו f גדולה מהסף. אם מצאת פתרון - החזר אותו.
3. אם בחיפוש לעומק לא נמצא פתרון הגדל את הסף בחריגה המינימלית שנמצאה בצעד 2, וחזור לצעד 2 (עריכת חיפוש לעומק עם סף חדש גדול יותר)

יתרונות IDA*

- ניתן להראות ש-IDA* הינו אלגוריתם קביל.
- מאוד חסכוני בזכרון.
- השימוש ביוריסטיקה מזרז את החיפוש באופן ניכר (לעומת ID הלא יוריסטית)


```
DFS-C (state, g, path)
    f ← g+h(state)
    if f > max-f then new-max-f ← min(new-max-f,f)
    else
        if goalp(state) return(path)
        else
            for c in succ(state)
                DFS-C(c, g+cost(state,c), path || state)

IDA*(state)
    max-f ← h(state)
    new-max-f ← infinity
    loop while resources are available
        solution ← DFS-C(state,0,())
        if solution <> () then return (solution)
        max-f ← new-max-f
```


- דוגמא להרצת IDA* על פאזל 4x4 רנדומי:

```
* (ida* #(14 13 15 7 11 12 9 5 6 0 2 1 4 8 10 3))
 Initial F: 41
Iteration:   0 cur-f:    43 Nodes:            213 Time:      0.00
Iteration:   1 cur-f:    45 Nodes:          1,708 Time:      0.00
Iteration:   2 cur-f:    47 Nodes:         12,541 Time:      0.01
Iteration:   3 cur-f:    49 Nodes:         85,142 Time:      0.05
Iteration:   4 cur-f:    51 Nodes:        537,078 Time:      0.27
Iteration:   5 cur-f:    53 Nodes:      3,294,900 Time:      1.68
Iteration:   6 cur-f:    55 Nodes:     19,850,323 Time:      9.94
Iteration:   7 cur-f:    57 Nodes:    117,507,765 Time:     58.84
Iteration:   8 cur-f:    59 Nodes:    599,952,132 Time:    299.88
(3 2 1 1 0 3 3 3 2 1 1 1 0 3 3 0 0 1 2 1 2 3 3 0 3 2 2 1 0 1 0 3 3 0 1 1
1 2 3 3 3 0 1 1 1 2 3 2 3 3 0 0 1 2 2 2 3)
*
```

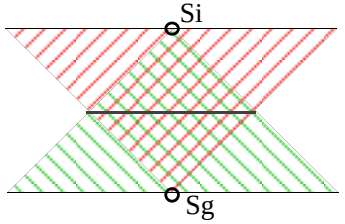
- מקדם הסיעוף הממוצע בפאזל 4x4 הינו 2.13. שימוש ב-ID יוביל ל-2.13^57=5.4*10^18 צמתים - פי עשרה מיליארד מאשר IDA*. זהו כוחה של היוריסטיקה.

דוגמא: IDA* לעומת ID

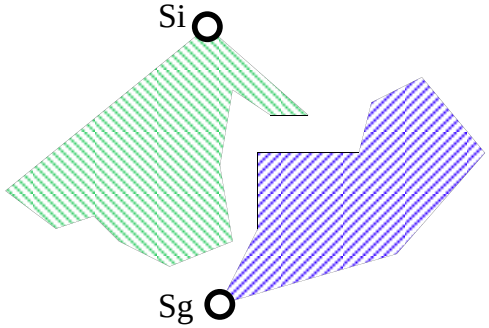
```
* (id #(5 1 3 6 0 7 2 8 4))
Depth:   0 Nodes:              1 Time:      0.00
Depth:   1 Nodes:              6 Time:      0.00
Depth:   2 Nodes:             19 Time:      0.00
Depth:   3 Nodes:             40 Time:      0.00
Depth:   4 Nodes:             77 Time:      0.00
Depth:   5 Nodes:            146 Time:      0.00
Depth:   6 Nodes:            279 Time:      0.00
Depth:   7 Nodes:            492 Time:      0.00
Depth:   8 Nodes:            865 Time:      0.00
Depth:   9 Nodes:          1,510 Time:      0.00
Depth:  10 Nodes:          2,699 Time:      0.00
Depth:  11 Nodes:          4,640 Time:      0.00
Depth:  12 Nodes:          8,085 Time:      0.01
Depth:  13 Nodes:         13,914 Time:      0.01
Depth:  14 Nodes:         24,511 Time:      0.02
Depth:  15 Nodes:         42,004 Time:      0.04
Depth:  16 Nodes:         73,289 Time:      0.07
Depth:  17 Nodes:        125,774 Time:      0.12
Depth:  18 Nodes:        220,659 Time:      0.21
Depth:  19 Nodes:        378,120 Time:      0.35
Depth:  20 Nodes:        660,733 Time:      0.59
Depth:  21 Nodes:      1,133,122 Time:      1.01
Depth:  22 Nodes:      1,985,063 Time:      1.74
Depth:  23 Nodes:      3,402,236 Time:      2.97
Depth:  24 Nodes:      5,949,873 Time:      5.12
Depth:  25 Nodes:     10,201,398 Time:      8.86
Depth:  26 Nodes:     10,949,317 Time:      9.50
(0 1 2 3 3 0 1 1 2 2 3 0 1 0 3 2 3 2 1 0 3 0 1 2 3 2)
* (ida* #(5 1 3 6 0 7 2 8 4))
 Initial F: 16
Iteration:   0 cur-f:    18 Nodes:             21 Time:      0.00
Iteration:   1 cur-f:    20 Nodes:             99 Time:      0.00
Iteration:   2 cur-f:    22 Nodes:            380 Time:      0.00
Iteration:   3 cur-f:    24 Nodes:          1,524 Time:      0.00
Iteration:   4 cur-f:    26 Nodes:          5,509 Time:      0.01
Iteration:   5 cur-f:    28 Nodes:          8,195 Time:      0.01
(0 1 2 3 3 0 1 1 2 2 3 0 1 0 3 2 3 2 1 0 3 0 1 2 3 2)
```

חיפוש דו כיווני

- ישנן בעיות עבורן יש בידינו את הפונקציה ההפכית לפונקצית המעבר.
- במקרים כאלו ניתן לחפש בשני כיוונים במקביל: ממצב ההתחלה לכיוון המצב הסופי ומהמצב הסופי לכיוון מצב ההתחלה.
- במידה ומקדם הסיעוף בשני הכיוונים זהה, חיפוש לרוחב דו-כיווני יהיה



יעיל יותרמאשר חיפוש לרוחב בכיוון אחד.

חיפוש יוריסטי דו-כיווני

- בחיפוש יוריסטי חזיתות החיפוש עלולות ל"פספס" האחת את השניה.