סיכום AI שנת 2020

מרצה: ג'ף רוזנשטיין, מתרגל: יוני שר, סיכום: מירה פינקלשטיין

13 באפריל 2020

תוכן עניינים

חלק I

שיעור 2

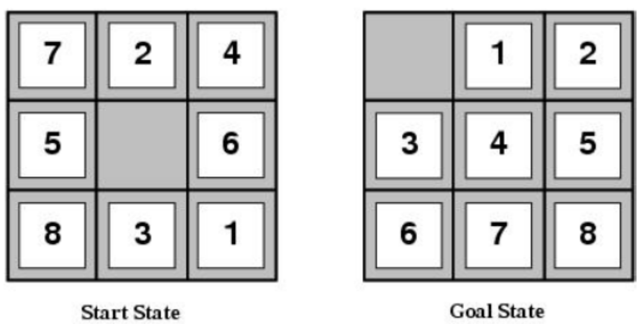
1 חיפוש - Best-first search

חיפוש הוא מאד חשוב - כי הרבה מהבעיות ניתן למדל כבעיות חיפוש. מרחב הפתרונות של בעיות שונות יכול להיות מאד גדול, לכן אנחנו מחפשים דרך להגיע לפתרון בתוך מרחב הפתרונות. נדבר על שתי בעיות מתוך בעיות החיפוש:

1. חיפוש שבו שדרך לפתרון היא חשובה.
2. חיפוש הוא הדרך לפתרון היא לא חשובה, העיקר שהגענו לפתרון (לדוגמא: בעיית צביעת מפה).

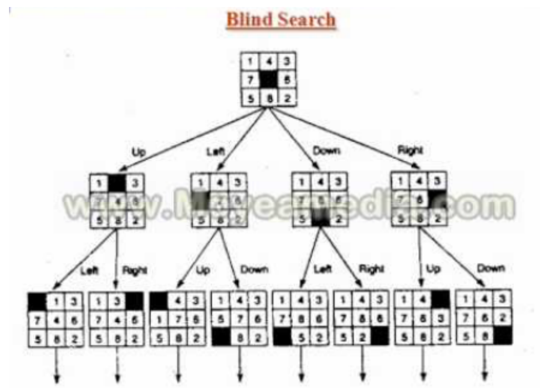
1.1 מרחב הפתרונות - Search Space

ניקח לדוגמא את המשחק הבא:



המטרה "להזיז" את הקוביה הריקה במרכז (ימינה, שמאלה, למעלה או למטה) בכדי להגיע למצב מימין.

בבעיה הזו אנחנו מעוניינים בכל דרך הפתרון - כלומר באפשרות 1. אפשרות אפשרית היא לעבור על כל הפתרונות כך:



וזה אומר שמרחב הפתרונות שלנו הוא **ענק**. אנחנו יכולים לחזור על אותם הקונפיגורציות, וכאן אנחנו מדברים על לוח קטן, 3x3.

ניתן להגיע לפתרון הבעיה ב־22 צעדים, כלומר, חיפוש כזה, כמעבר על כל האפשרויות יתן לנו 3.1×10^{10} מצבים.

היום נלמד יוריסטיקה שיכולה לעזור לנו להתמצא במרחב הפתרונות שלנו.

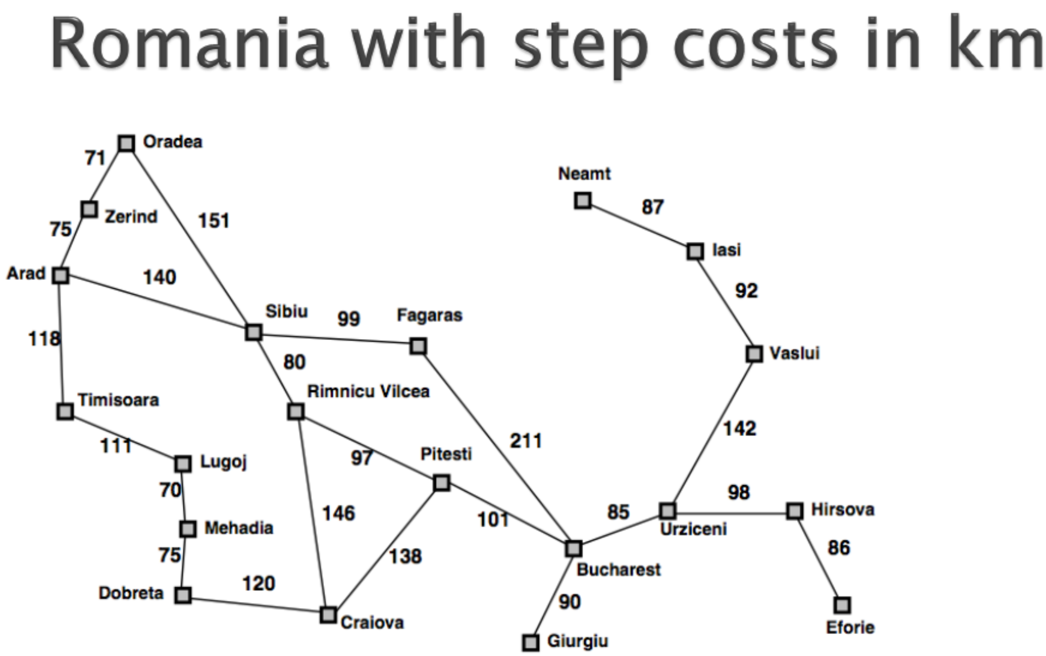
נזכר בעץ החיפוש בתרגול: ניקח קודקוד, ונרחיב אותו ־ כלומר נפרט את כל המצבים שאליהם ניתן להגיע מהמצב הנוכחי ע"י פעולה כלשהי. אז אנחנו מכניסים את הקודקודים לתוך תור עדיפות, כלומר שהוא יהיה ממויין בדרך מסויימת (fringe). ואז ניקח את הקודקוד הראשון בתור ־ אם הוא פותר את הבעיה, נסיים. אחרת ־ נמשיך ונרחיב את הקודקוד וכן הלאה (עד שאין לנו לאן להתרחב). זה מודל יחסית פשוט, וקל למימוש, וזה יהיה הבסיס למה שנדבר בשיעור הזה.

הרעיון: שימוש בפונקציית משקל $f\left(n\right)$ (יש הרבה אפשרויות לבחור אותה) לכל קודקוד. אנחנו צריכים כאן למדוד "כדאיות". נרחיב את הקודקוד ה"כדאי" ביותר (בדר"כ הראשון בתור).

מימוש: נדר את הקודקודים בתור\fringe בסדר יורד של כדאיות.

מצבים מיוחדים: greedy best-first search ,DFS ,BFS כאשר יש פונקציית יוריסטיקה ל־$f\left(n\right)$, A^* אותו הדבר רק עם פונקציה שונה.

ניקח לדוגמא את הבעיה הבאה. זו בעיית חיפוש כדלהלן ־ לפנינו מפת דרכים ברומניה, היעד שלנו הוא העיר **בוקרשט**, אנחנו מחפשים ־ מה הדרכים הכדאיות ביותר להגיע אליה:



- פונקציית המשקל\ההערכה היא הערכה של מחיר מ־n למטרה:

$$f(n) = \overbrace{h_{SLD}}^{\text{huristic}}(n) = \text{straight-line distance from n to Bucharest}$$

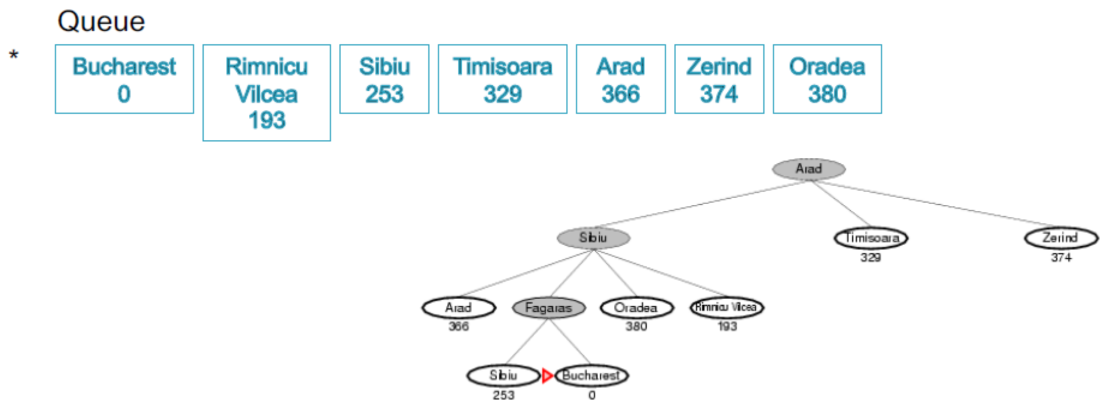
- האלגוריתם Greedy best-first search ירחיב את הקודקוד שהכי קרוב למטרה (במרחק הקרוב ביותר בקו ישר ־ שזה המינימום של המרחק הכי קרוב למטרה, וזה נותן לנו את היוריסטיקה, מה <u>יכול</u> להיות המקום הקרוב ביותר).

בנתיים לא נתעניין איך מוצאים את היוריסטיקה ־ אלא נתייחס אל זה כפונקציית קופסא שחורה שמקבלת מצב ומחזירה מספר. (במקרה שלנו הפונקציה מקבל עיר ־ ומחזירה מספר שהוא המרחק בקו ישר הכי קרוב לבוקרשט).

כלומר זה לא המסלול הקצר ביותר ־ אלא המרחק הקצר ביותר. לא להתבלבל.

במקרה של Greedy best-first search פונקציית ההערכה שלנו = לפונקציית היוריסטיקה (זה לא תמיד ככה?), כלומר הסדר של תור העדיפות שלנו של הקודקודים הוא פונקציית היוריסטיקה שלנו.

נחזור לדוגמא שלנו ־ נניח שאנחנו רוצים להריץ את האלגוריתם כאשר העיר שבה אנחנו מתחילים היא Arad בקצה העליון השמאלי של המפה. נגיע למצב הבא:



הגענו לבוקרשט. אבל - האם סיימנו?
אז כאן ישנה דקות, אמרנו שאנחנו בודקים את הקודקוד כשאנחנו מוציאים אותו מהתור, ולא כאשר אנחנו מכניסים אותו לתור. במקרה הספציפי הזה, בגלל שפונקציית היוריסטיקה שלנו שווה ל-$f(n)$ אז באמת סיימנו (זה לא תמיד המקרה - נראה זאת כבר בדוגמא הבאה).
הבעיה באלגוריתם - שהוא לא נתן לנו את המסלול הקצר ביותר, כי קצר יותר היה לסוע דרך Pitesti. זה קרה כיוון שבחרנו את פונקציית ההערכה להיות פונקציית היוריסטיקה, וזה לא היה מספיק כדי לתת לנו את התשובה האופטימלית.

פרמטרים להערכה - b הוא פקטור העינוף, m זה העומק המקסימלי של מרחב הפתרונות:

- זה פתרון **לא שלם** - כי אם הינו מגיעים לקצה, נכנס ללולאה.
- **זמן** - $O(b^m)$ כאשר יוריסטיקה טובה יכולה לתת לנו שיפור משמעותי.
- **זיכרון** - $O(b^m)$ - כיוון שאנחנו שומרים את כל הקודקודים בזכרון.
- וכפי שאמרנו, זהו פתרון **לא אופטימלי**.

וזה מה שמביא אותנו לאלגוריתם משופר:

2 חיפוש - A* search

- הרעיון: נמנע מלהרחיב מסלולים שכבר הורחבו בעבר.
- פונקציית ההערכה:

$$\overbrace{f(n)}^{\text{estimated total cost of path through n to goal}} = \overbrace{g(n)}^{\text{cost so far to reach n}} + \overbrace{h(n)}^{\text{estimated cost from n to goal}}$$

כלומר - מגדירים את הפונקציה להיות החיבור של המסלול הכי משתלם עד ל-n ועוד המסלול הכי משתלם מ-n למטרה. נזכיר שפונקציית ההערכה היא זו שקובעת איך יהיה מסודר תור העדיפות.

- להבדיל מהאלגוריתם הקודם - האלגוריתם הזה הוא **אופטימלי** (לא במובן של כמה קודקודים בודקים, אלא שהפתרון שאנחנו מגיעים אליו הוא האופטימלי).
- החיפוש A^* משתמש בהירסטיקה קבילה (admissable):
  - המקיימת $h(n) \leq h^*(n)$ כאשר h עם כוכבית זה המחיר האמיתי.
  - דורשת גם ש־$0 \leq h(n)$, כך ש־$h(G) = 0$ לכל מטרה G.

2.0.1 הירסטיקה קבילה - Admissible heusristics

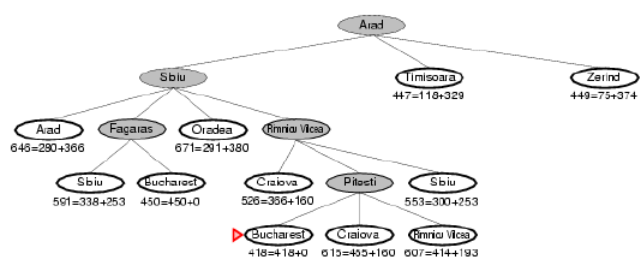
זוהי הירסטיקה "אופטימית" כלומר היא לעולם לא תעריך את המחיר יותר ממה שהוא. כלומר אנחנו תמיד חושבים שהגענו למטרה כמה שיותר מוקדם.

משפט אם $h(n)$ היא הירסטיקה קבילה, אזי אלגוריתם A^* שמשתמש ב־Tree-search הוא אופטימלי. אבל גרף החיפוש יכול להיות עדיין סב־אופטימלי כיוון שהשמטנו מצבים שביקרנו בהם כמה פעמים גם אם הם נמצאים על המסלול הקצר.

בדוגמא הקודמת נקבל את העץ הבא:

Queue

* Bucharest 418	Timisoara 447	Zerind 449	Bucharest 450	Craiova 526	Sibiu 553	Sibiu 591	Rimnicu Vilcea 607	Craiova 615	Arad 646	Oradea 671



כל שלב בחיפוש מושפע גם מהמסלול למטרה, וגם מהמסלול על המצב הנוכחי. נשים לב שבשלב הראשון נקבל את אותו התור עם אותם הקודקודים כמו באלגוריתם הקודם - רק **הסדר** שלהם יהי שונה.

נשים לב - כאן זה חשוב מתי בודקים אם הגענו למטרה - אם כמכניסים או מוציאים לתור. כיוון שאחרת הינו יכולים לקחת את המסלול היותר ארוך לבוקרשט(באורך 450).

2.1 הוכחת אופטימליות של A^*

פונקציית ההערכה -

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

נרצה להוכיח שנפתח כל קודקוד ב־fringe במסלול ליעד האופטימלי G לפני שנוציא את G' מה־fringe.

(בדוגמא שלנו G' זה היעד **מסלול לבוקרשט באורך 450** והיעד G הוא **מסלול לבוקרשט באורך 418**).

נניח שיש לנו יעד סב־אופטימלי G' שנמצא בתוך ה־fringe. יהיה n קודקוד שעוד לא הורחב בתוך ה־fringe כך ש־n נמצא על המסלול הקצר ליעד האופטימלי G.

- בגלל ש־$h(G')=0$ (כיוון שהגענו לבוקרשט) מתקיים ש־

$$g(G')=g(G')+h(G')=f(G')$$

.

- מההנחה ש־G' הוא סב־אופטימלי מתקיים ש־

$$g(G)<g(G')$$

- בגלל ש־h היא אדמיסבילית ("אופטימית") מתקיים ש־

$$f(n)\leq g(G)$$

כלומר מתקיים:

$$f(n)\leq g(G)<g(G')=f(G')$$

וכיוון ש־$f(n)<f(G')$, האלגוריתם A* לעולם לא יבחר את G' להרחבה ־ ולכן לא יוציא אותו מה־fringe ולא יבחר בו כאופטימלי.

קבילות של h היא יעיל בשימוש בחיפוש בעץ ולא בשימוש בגרף, לכך יש שני פתרונות:

- נרחיב את גרף החיפוש כך שלכל שני מסלולים לאותו הקודקוד הוא ישמיט את היקר ביותר ־ אבל זה דורש יותר מקום.
- נדרוש מ־h להיות גם קבילה וגם קונסיסטנטית.

2.1.1 היריסטיקה קונסיסטנטית/מונוטונית

(זה שקול לאי־שוויון המשולש, רק למצבים)

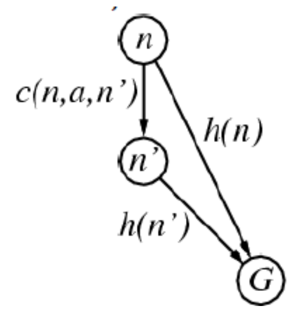
נאמר שהיריסטיקה היא קונסיסטנטית אם לכל קודקוד n, כל ילד n' של n שהתקבל מכל פעולה a אם היא מקיימת:

$$h(n)\leq c(n,a,n')+h(n')$$

- אם h קונסיסטנטית, נקבל:

$$f(n')=g(n')+h(n')=g(n)+c(n,a,n')+h(n')\geq g(n)+h(n)=f(n)$$

- כלומר - אם $h(n)$ היא קונסיסטנטית, $f(n)$ היא לא יורדת לאורך כל מסלול - כלומר המטרה הראשונה שנבחרת להרחבה באלגוריתם חייבת להיות אופטימלית, כיוון שכל הקודקודים שבאים אחריה יהיו לכל הפחות במחיר זהה.



כל היריסטיקה קונסיסטנטית היא גם קבילה אבל לא ההפך.

משפט אם $h(n)$ היא קונסיסטנטית, אלגוריתם A^* המשתמש בגרף חיפוש הוא אופטימלי.

(למרות שקבילות לא גוררת קונסיסטנטית, למצוא היריסטיקה שהיא קבילה ולא קונסיסטנטית זה קשה).

2.2 המאפיינים של A^*

- **שלמות**: כן (למעט אם אנחנו במקרה של מס' קודקודים אינסופי עם $f \le f(G)$)
- **זמן**: אקספוננציאלי ב-[השגיאה היחסית ב-h × האורך של הפתרון]
- **מקום**: שומר בזיכרון את כל הקודקודים.
- **אופטימליות**: כן - לא יכול להרחיב את f_{i+1} עד ש-f_i הסתיים.
  - האלגוריתם A^* מרחיב את כל הקודקודים עם $f(n) < C^*$
  - האלגוריתם A^* מרחיב חלק מהקודקודים עם $f(n) = C^*$ (כלומר אם יש כמה מסלולים עם המחיר האופטימלי, לא בהכרח הוא ירחיב את כולם).
  - האלגוריתם A^* אינו מרחיב קודקודים עם $f(n) > C^*$

(כאשר C^* זה המחיר האופטימלי).

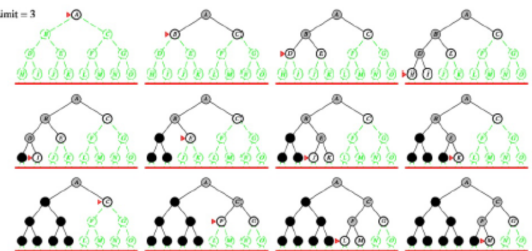
חיפוש עם זיכרון מוגבל - Memory-bounded heuristic search

לשמור את כל הקודקודים בזיכרון זה לא אופטימלי, ולכן ישנם כמה שיפורים לאלגוריתם A^*:

- אלגוריתם IDA=Iterative Deepening A*
- אלגוריתם RBFS=Recursive Best-First Search
- אלגוריתם (S)MA=(Simple) Memory-bounded A*

3 אלגוריתם IDA*

הרעיון ב־IDA הוא הרצה של DFS כל פעם עם הגבלת עומק (catoff depths).



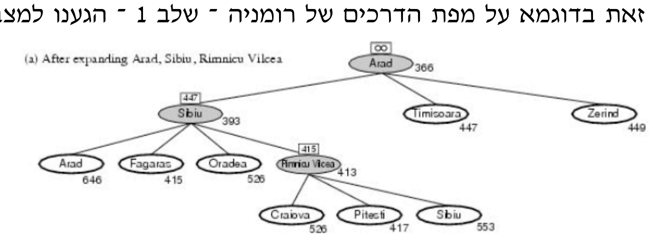
- הרעיון ב־*IDA הוא זהה למעט שבמקום הגבלה על העומק, הוא מגביל כל פעם את $f = g + h$ - f.
- בכל איטרציה הערך של החתך/ההגבלה הוא המחיר של f הכי נמוך על כל הקודקודים שחרדו את ההגבלה באיטרציה הקודמת.
- זה פרקטי - בתלות בצורה של **הצורה** של מרחב הפתרונות (ניגע בזה בהמשך).

4 אלגוריתם RBFS

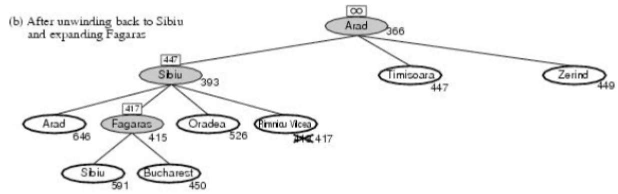
אנחנו רוצים להקטין את ה־fringe.
מה שהאלגוריתם עושה זה מעקב אחרי ה־f-value הבא בתור - כלומר בכל פעם, קודקוד האב זוכר את המסלול הבא לפי ה־f-value, ואם ה־f-value של הבנים שלו, גבוהה יותר ממה שהוא זוכר, הוא זוכר את ה־f-value הנמוך ביותר מבינהם, זורק אותם ועובר למסלול החלופי:

- אם ה־f-values הנוכחים חורגים מה־f-value האלטרנטיבי, אז אנחנו חוזרים למסלול האלטרנטיבי.
- בזמן החזרה - נשנה את ה־f-value להיות ה־f-value הטוב ביותר (הנמוך ביותר) של הילדים שלנו.
- נרחיב את הקודקוד עם הערך האלטרנטיבי שאליו חזרנו ונחזור על הפעולות.

נראה זאת בדוגמא על מפת הדרכים של רומניה - שלב 1 - הגענו למצב הבא:

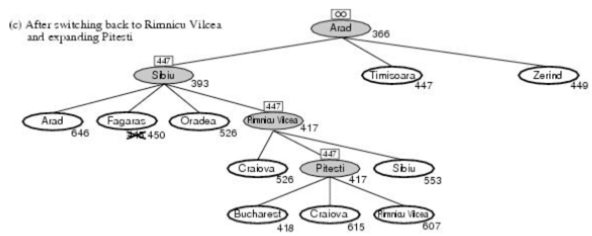


נשים לב כי מעל הערים יש מספר - המספר הזה מסמן את ה־f-value החלופי.
שלב 2:



השוונו את ה־f-value של Rumnicu Vilcea ל־f-value של הבנים שלו, ובגלל שהערכים שלהם גבוהים יותר ־ זרקנו אותם, והמשכנו למסלול החלופי.

שלב 3:



4.1 הערכה של RBFS

- האלגוריתם מעט יעיל יותר מ־*IDA מבחינת מקום.
- כמו ב־*A ה־$h(n)$ האופטימלי קביל.
- **מקום**: היא ־ $O(bd)$
- **זמן**: קשה לאפיין, זה תלוי בדיוק של $h(n)$ וכמה תכוף המסלולים משתנים.
- שני האלגוריתמים ־ *IDA ו־RBFS משתמשים במעט מדי זיכרון (כלומר גם אם היה יותר זיכרון, הם לא היו משתמשים בו).

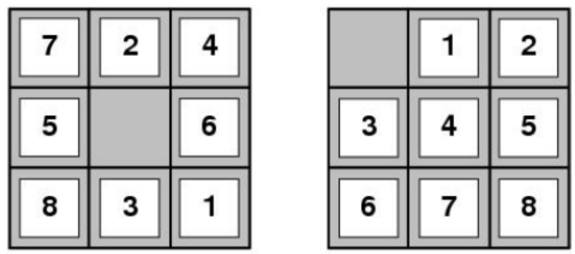
5 אלגוריתם *(S)MA

משתמש בכל הזיכרון שזמין לו.

- האלגוריתם מרחיב את העלים עד שהזיכרון שזמין לו מתמלא. לאחר שהוא מתמלא ־ האלגוריתם משמיט את העלה עם ה־f-value הגבוהה ביותר. כמו ב־RBFS הוא שומר את הקודקוד שמושמט אצל ההורה.
- אם לשני עלים יש את אותו הערך של־f-value ־ נתעדף להרחיב את הקודקוד ה"חדש" יותר ב־fringe.
- **שלמות**: שלם אם ניתן להגיע לפתרון.
- **אופטימליות**: אופטימלי אם ניתן להגיע לפתרון אופטימלי.

6 פונקציות היריסטיקה

נתבונן הפאזל הבא:



Start State   Goal State

הפתרון הממוצע יקח לנו 22 צעדים (עם פרמטר עינוף - branching factor של ±3) חיפוש ומעבר על כל הפתרונות כפי שציינו כבר יתן לנו עומק של 3.1×10^{10}. פונקציית יוריסטיקה טובה תשפר לנו את העומק באופן משמעותי.
נתובנן בשתי פונקציות אפשריות:

1. ספירה של אבנים שלא במקומם:

$$h_1\left(s\right) = 8$$

2. סכימה של המרחק של כל אבן מהמטרה שלה ללא התחשבות באבנים האחרות:

$$h_2\left(s\right) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$$

נשים לב ששתי הפונקציות מתייחסות לבעיה קלה יותר, זו דרך מקובלת למצוא פונקציות יוריסטיקה.

6.0.1 דומיננטיות - Dominance

נאמר ש־h_2 דומננטי יותר מ־h_1 אם שתיהן קבילות, ולכל n מתקיים - $h_1\left(n\right) \leq h_2\left(n\right)$ (h_2 טוב יותר לחיפוש).

6.1 איכות של פונקציית היריסטיקה

6.1.1 פקטור עינוף b* - branching factor

פקטור עינוף אפקטיבי b* שיש לעץ אחיד מעומק d כדי שיכיל N+1 קודקודים:

$$N + 1 = 1 + b^* + \left(b^*\right)^2 + ... \left(b^*\right)^d$$

כאשר הוא קרוב ל־1 זה אומר שהוא יותר טוב. נוכל לחשב את b* ובכך להשוות את ההיריסטיקות השונות.

היריסטיקה ובעיות NP־קשות

- למדנו בקורסים אחרים במדעי המחשב, על כך שהרבה בעיות בעולם בעלות גידול אקספוננציאלי בסיבוכיות שלהן. אבל התייחסנו למקרים הגרועים ביותר.
- לחלק מהבעיות, הרבה מהמופעים שלהן יכולים להפתר בזמן פולינומיאלי, למעט אולי חלק קטן מאד שמגיעים לגידול אקספוננציאלי.
- היריסטיקות יכולים לעיתים, לספק פתרון פרקטי ויעיל ־ גם אם הבעיה במקרה הגרוע ביותר בעלת סיבוכיות אקספוננציאלית.

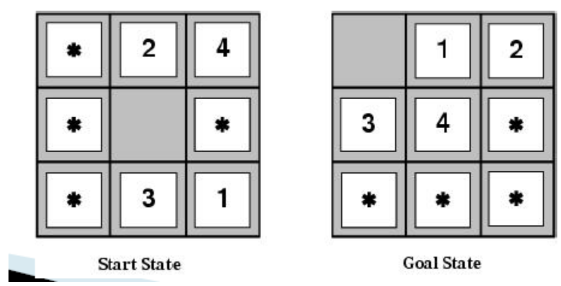
שחרור הבעיה למציאת היריסטיקה

- כדי למצוא היריסטיקה מתאימה, לעיתים נרצה לשחרר/להקל קצת את הבעיה.
- המחיר של הפתרון האופטימלי לבעיה הקלה יותר, היא היריסטיקה קבילה לבעיה המקורית.
- דוגמא: אם נסתכל על בעיית הפאזל
  - $h_1(n)$ היא עם הנחה מקלה שניתן להרים אבן ולהניח במקומה
  - $h_2(n)$ היא עם הנחה מקלה כי כל אבן יכולה לזוז לכל ריבוע צמוד גם אם יש שם אבן אחרת.

6.1.2 ה־Pattern databases

ה־Pattern databases מאחסנית את הפתרון המדוייק עבור כל מופע של תת־בעיה אפשרי.

כלומר במקרה של הפאזל, נסתכל על תת הבעיה ש־4 מהאבנים הן כוכביות, ולכן לא צריך לסדר אותן:



בעיה כזו קל יותר קל יותר לפתור ־ וכך ניתן למצוא את ההירסטיקה הקבילה המתאימה לבעיה המלאה. איך עושים את זה:

- נניח כי הרצנו את האלגוריתם על הבעיה משמאל. וקיבלנו שההירסטיקה שלנו מחזירה לנו את המספר 25.
- כעת כל פעם שנגיע לקונפיגורציה הזו בבעיה המקורית (לא משנה איפה מופיעים המספרים 5־8), נשתמש במספר 25 כהיריסטיקה.

וזה נכון כי עשיתי חיפוש לתת הבעיה ־ ואת הפתרון שמרתי במאגר נתונים. וזה נותן לנו פתרון בעיות הרבה יותר מהיר.

7 אלגוריתמי חיפוש מקומיים - Local search algorithms

- הרבה פעמים בבעיות - לא תמיד הדרך לפתרון היא חשובה, אלא רק הפתרון עצמו.
- אז מרחב המצבים = קבוצה של קונפיגורציות "שלמות" (לדוג' בבעיית צביעת המפות - אנחנו מחפשים קונפיגורציה של מצבים שבהם הכל צבוע באופן חוקי).
- במקרים כאלו אנחנו יכולים להשתמש באלגוריתמי חיפוש מקומיים או אלגוריתמי שיפור איטרטיביים - iterative improvement algorithms.

7.0.1 חיפוש מקומי - Local search

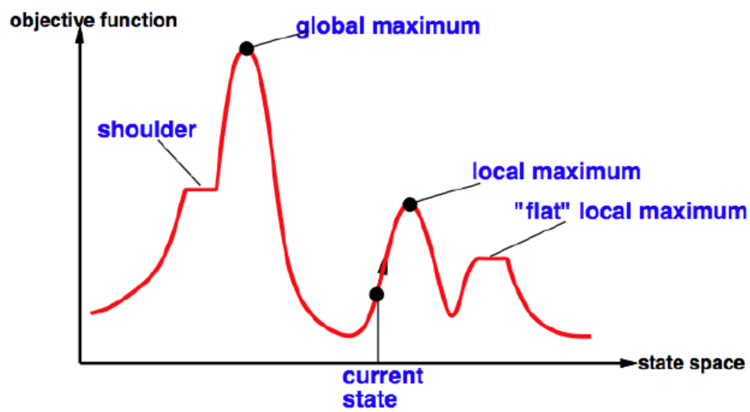
משתמש במצב נוכחי יחיד, ומנסה לשפר אותו ע"י הזזה למצבים שכנים.

יתרונות:

- משתמש במעט מאד מקום - מקום קבוע.
- לרוב מוצא פתרונות טובים במרחבי פתרונות גדולים או אינסופיים.
- טוב גם לבעיות אופטימיזציה - למצוא את המצב הכי טוב בהתאם לפנקציית מטרה כלשהי.

7.1 חיפוש טיפוס הרים - Hill-climbing search

שימושי כאשר מתחשבים ב<u>צורה של מרחב המצבים</u> -



Random-restart hill climbing overcomes local maxima—trivially complete

Random sideways moves 😈escape from shoulders ☹loop on flat maxima

תקרא גם - gradient ascent\descent

- זוהי לולאה שתמיד נעה לכיוון הערך שהולך וגדל, עוצרת כאשר מגיעים לפסגה.
- האלגוריתם הזה לא מסתכל על השכנים של המצב הנוכחי. אלא בוחר באופן אקראי מתוך הבנים הטובים ביותר אם יש יותר מאחד.

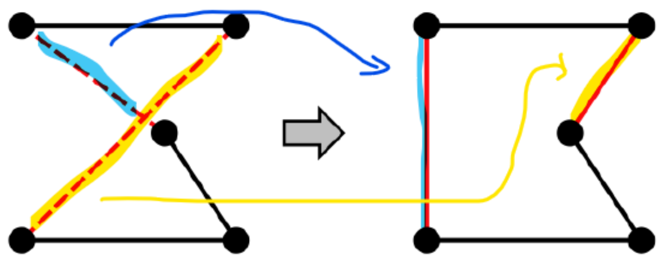
• זהו אלגוריתם חיפוש מקומי חמדן.

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node

   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
      neighbor ← a highest-valued successor of current
      if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
      current ← neighbor
   end
```

דוגמא:

בעיית הסוכן הנוסע - נתחיל מכל מסלול שלם, ונבצע החלפות של זוגות של צלעות:



יש הרבה החלפות שאפשר לעשות, אבל זוהי החלפה ***מקומית***. מסתכלים על פונקציית h על החלפה של הזוגות ובוחרים את הטוב ביותר.

יש כמה אפשרויות ל־Hill-climbing:

1. **סטוכסטי - Stichastic hill-climbing**: בחירה אקראית בין המהלכים במעלה ה"גבעה" (לאו דווקא העלייה התלולה ביותר). כאשר הסתברות החבירה משתנה עם תלילות העלייה.

2. **ה־First-choice hill-climbing**: מממש את הסטוכסטי ע"י יצירת בנים באופן אקראי עד שנמצא בן טוב למצב הנוכחי. שימושי במיוחד אם יש בנים רבים.

3. **אתחול אקראי - Random-restart hill-climbing**: מנסה להתחמק מלהתקע במקסימום לוקלי. מייצר סדרה של חיפושים בטיפוס הרים שכל אחד מתחיל ממצב אקראי אחר. עוצר או לאחר הגעה למקסימום לוקאלי, או אחרי זמן מסויים. ההצלחה/כשלון של השיטה הזו תלויה מאד בצורה של מרחב המצבים.

חלק II
תרגול 2

8 ה־Informed search

(התרגול היה ברובו המוחלט חזרה על השיעוא)

8.0.1 בעיית חיפוש פורמאלית

- זו חמישיה $\langle S, s_0, G, A, G, C\rangle$
- S הוא קבוצה של מצבים.
  - $s_0 \in S$ הוא המצב ההתחלתי
    * $G \subset S$ זה קבוצה של מצבי המטרה/יעד.
    * A זה קבוצה של פעולות
    * F זה פונקציית המעברים ־ $F : S \times A \to S$
    * C זו פונקציית המחיר/משקל ־ $C : S \times A \to \mathbb{R}_{\geq 0}$

8.0.2 פתרון

- הוא זוג $\left\langle \{s_i\}_{i=0}^{n}, \{a_i\}_{i=0}^{n-1} \right\rangle$, כאשר מחיר הפתרון = $\sum_{i=0}^{n-1} C\left(s_i, a_i\right)$
- s_0 הוא המצב ההתחלתי
  - $s_n \in G$
  - מתקיים $s_i = F\left(s_{i-1}, a_{i-1}\right)$ לכל $1 \leq i \leq n$

8.0.3 אסטרטגיית החיפוש

- היא בחירה של הסדר של הרחבת הקודקודים מתוך ה־fringe ־
- סידור ה־fringe ־ לבחור איזה קודקוד יהיה מתאים יותר כדי להוביל לפתרון.
  - הוספת קודקודים נוספים לfringe ־ לא כל הקודקודים רלוונטים לחיפוש שלנו.
  - איתחול ה־fringe ־ איתחול חלקי עם עידכון עפ"י היריסטיקה.

8.0.4 עץ חיפוש

- זה עץ "מה אם" כאשר:
- s_0 ־ המצב ההתחלתי הוא השורש.
  - הילדים של קודקוד מתאימים להרחבה של הקודקודים.
  - כל קודקוד מכיל מצב.
  - לרוב הבעיות ־ אנחנו לא נרצה לבנות את כל העץ.

8.0.5 פונקציית הערכה

- מסמנים בצורה הבאה - $f(v) = g(v) + h(v)$
  - $h(v)$ - פונקציית ההירסטיקה - המחיר המוערך להגעה לקודקוד היעד מקודקוד v (זה מה שאנחנו בעצם מחפשים).
  - $g(v)$ - המחיר המדוייק של ההגעה מהשורש לקודקוד v (אנחנו יודעים את המחיר, כי כבר הגענו מהשורש לקודקוד v).
  - כלומר בהינתן עץ חיפוש (V,E), ובהינתן פונקציית מחיר $C : E \to \mathbb{R}_{\geq 0}$, עבור המסלול - $(v_0, ..., v_n)$, מתקיים:

$$g(v_n) = \sum_{i=1}^{n} C(e_i) \ , \ e_i = (v_{i-1}, v_i) \ , \ h : V \to \mathbb{R}_{\geq 0}$$

8.0.6 חיפוש - BFS

- הרעיון: להשתמש בפונקציית הערכה לכל קודקוד - ולפיה להעריך את הכדאיות של הרחבת הקודקודים ולהרחיב לפיה.
  - מימוש: לסדר את הקודקודים ב-fringe בסדר יורד של כדאיות:
    * חיפוש Uniform-cost - משתמש ב-$g(v)$
    * חיפוש Greedy best-first - משתמש ב-$h(v)$
    * חיפוש A* - משתמש ב-$f(v) = g(v) + h(v)$
    * חיפוש Weighted-A* משתמש ב-$g(v) + w \cdot h(v)$ כאשר $w > 1$

8.0.7 חיפוש Greedy best-first

- פונקציית הערכה היא $h(v)$, כלומר הערכה של המחיר מהמצב הנוכחי v ליעד.
  - מרחיב את הקודקוד שמופיע הכי קרוב ליעד (לדוגמא המרחק האווירי).
  - מאפיינים:
    * <u>שלמות</u> - לא שלם. יכול להתקע בלולאה אינסופית במקרה האינסופי. שלם במקרה הסופי כשבודקים שלא חזרנו על מצבים.
    * <u>נאות</u> - לא נאות. יכול להתקע בלולאה.
    * <u>זמן</u> - $O(b^m)$ היריסטיקה טובה יכולה לתת שיפור משמעותי.
    * <u>מקום</u> - $O(b^m)$ שומר את כל הקודקודים בזיכרון.
    * <u>אופטימלי</u> - לא.

8.0.8 חיפוש A*

- <u>הרעיון</u> - להמנע ממסלולים שכבר הרחבנו. שימוש בפונקציית הערכה $f(v) = g(v) + h(v)$
  - <u>אסטרטגיה</u> - פונקציית היריסטיקות: $f(v)$, סדר ה-fringe עולה ב-f, ללא איתחול.

מאפייני ההיריסטיקות:

- אדמיסיביליות/קבילות - לכל $v_0 \in V$, לכל מסלול מ-v_0 ליעד -$(v_0, ..., v_n)$ כאשר $v_n \in G$:

$$h(v_0) \le \sum_i^n c(e_i)$$

היריסטיקה אדמיסיבילית לעולם לא תעריך יתר על המידה את המחיר כדי להגיע ליעד, כלומר היא הערכה אופטימית.

- קונסיסטנטית - לכל צלע $e = (v, v') \in E$, מתקיים:

$$h(v) \le c(e) + h(v') \quad \text{and} \quad \forall\, v \in G,\ h(v) = 0$$

במילים אחרות מדובר על אי שוויון המשולש כפי שציינו בהרצאה.

- דומיננטיות - h_2 דומיננטית על h_1 אם לכל $v \in V$ מתקיים:

$$h_1(v) \le h_2(v)$$

מאפייני החיפוש:

- שלמות - כן. (אלא אם כן יש קודקוד n שניתן לעבור דרכו אינסוף פעמים עם $f(n) \le f(\text{goal})$.
- נאותות - לא. כי זה יכול להיות מרחב אינסופי שהיעד קיים. אם הוא וספי אז הוא נאות.
- זמן - אקספוננציאלי ביחס ל-[שגיאה ב-h × האורך של הפתרון]
- מקום - $O(b^m)$ שומר את כל הקודקודים בזכרון.
- אופטימליות - אופטימלי בעץ חיפוש במקרה של היריסטיקה אדמיסיבילית, ובחיפוש בגרף במקרה של היריסטיקה קונסיסטנטית.

- הוכחת אופטימליות של A^* עבור עצים - בהרצאה.

- הוכחת אופטימליות של A^* עבור גרפים:

טענה - אם h היא קונסיסטנטית אז כל פעם ש-A^* מרחיב קודקוד, הוא כבר נמצא על המסלול האופטימלי למצב של הקודקוד:

- יהי v קודקוד ו-v' הילד שלו.
- בגלל ש-h קונסיסטנטית, מתקיים:

$$h(v) \le c(v, v') + h(v')$$

- ולכן:

$$f(v) = g(v) + h(v) \le g(v) + c(v, v') + h(v') = g(v') + h(v') = f(v')$$

- כלומר $f(v) \leq f(v')$, ולכן אם f היא לא־יורדת לאורך כל מסלול, אז כשבורחים קודקוד להרחבה, נמצא כבר המסלול האופטימלי לאותו הקודקוד.

  לסיכום: A^* מרחיב קודקודים לפי הסדר של הערך העולה של f, כלומר:

  - ירחיב את כל הקודקודים שמקיימים $f(v) < C^*$
  - ירחיב חלק מהקודקודים שמקיימיים $f(v) = C^*$
  - לא ירחיב קודקודים שמקיימיים $f(v) > C^*$

9 חיפוש מקומי - Local search

חיפוש מקומי זו היריסטיקה לפתרון של מחלקה של בעיות אופטימיזצה שקשה לפתור אותה מהבחינה החישובית.

בהרבה מבעיות האופטימיזציה, המסלול ליעד לא רלוונטי, אלא רק ההגעה עצמה ליעד. מרחב הפתרונות במקרה הזה הוא מרחב המועמדים לפתרון (ולא מרחב של מסלולים) והחיפוש הלוקלי יעבור מפתרון לפתרון במרחב עד שיגיע לפתרון האופטימלי.

החקר של מרחב הפתרונות נעשה על ידי תנועה מהפתרון הנוכחי, לפתרון "שליד" (בשביל זה צריך להגדיר איזשהו יחס שכנות).

יתרונות:

- משתמש במעט מאד זיכרון (בפועל משתמשים רק בזיכרון בגודל של קבוע)
- מוצא לרוב פתרון במרחב אינסופי או מאד גדול של פתרונות.

9.0.1 שיטת Hill-climbing

נקרא גם Greedy Local Search כיוון שהוא לוקח את השכן/המצב "הכי טוב" בלי לחשוב על המצבים העתידיים, פורמאלית:

- current $\leftarrow$ MAKE_NODE(INITIAL_STATE)
  - בלולאה:
    - neighbor $\leftarrow$ highest values successor of current
    - if $f(\text{neighbor}) < f(\text{current})$: return $\text{STATE}(\text{current})$
    - else: current $\leftarrow$ neighbor

  אם אנחנו נלך רק למעלה - אז יכול להיות שנתעק במקסימום לוקאלי. לכן נרצה לעיתים ללכת טיפה למטה, כדי לעלות הרבה יותר למעלה בהמשך. אחת השיטות לעשות את זה נקראת Annealing.

9.0.2 ה־Simulated Annealing

ה־Annealing - לברוח ממקסימום מקומי על ידי כך שנאפשר תנועות "רעות" אבל נקטין את הגודל ואת התדירות שלהם.

- היריסטיקה - פונקציית המטרה $F : S \to \mathbb{R}$
  - ה־fringe - מסודר לחלוטין לפי F.

– קודקודים חדשים - בן אחד של הקודקוד הנוכחי שנבחר באקראי.

– אתחול - מקבלים את הקודקוד החדש אם:

$$F\,(\text{current}) < F\,(\text{new})$$

אחרת: הקבוצה הפעילה משוחזרת להיות הקבוצה הקודמת (כלומר current) - האלגוריתם המפורט במצגת ובהרצאה.

9.0.3 ה־Local beam search

עושים מעקב אחר k מצבים במקום מצב אחד.

- אתחול - k מצבים אקראיים.
  - צעד - כל הקודקודים בנים של k המצבים.
  - אם אחד מהבנים הוא היעד - נסיים. אחרת - נבחר את k הבנים הכדאיים ביותר ונחזור על התהליך.

יתרונות:

- המידע משותף לכל k הענפים של החיפוש.
- יכול לסבול מחוסר ב־"diversity". אבל זה ניתן לפתור על ידי שימוש באפשרות הסטוכסטית - בחירה של k מצבים בהסתברות מסויימת.

10 אלגוריתמים גנטים - Genetic algorithms

ימצא תמיד את הפתרון האופטימלי - אבל יכול להיות שזה יקח המון זמן. באלגוריתם הזה אנחנו מחליפים את ה־fringe המסודר מאד - ב־fringe שיכול לא להכיל בכלל חוקיות נראית לעין לסדר שלו.

- כל מצב יקרא Individual, המיוצד בוקטור $v \in \mathbb{R}^n$
- מתחילים מקבוצה של אינדיוידואלים אקראיים, הנקראת population.
- הפעולות האפשריות:
  - בחירה Reproduction selection - ההסתברות שיבחרו בך תלויה שפונקציה שנותן דירוג לכל אינדיוידואל.
  - הצלבה Crossover - נקודת המיזוג של שני וקטורים.
  - מוטציה Mutation - כל מיקום הוא יעד של מוטציה אקראית בהסתברות אקראית בלתי תלויה (כדי להכניס יותר אקראיות).