חיפושים לא מיודעים במרחבי מצבים

מבוא לבינה מלאכותית

© Shaul Markovitch 2005

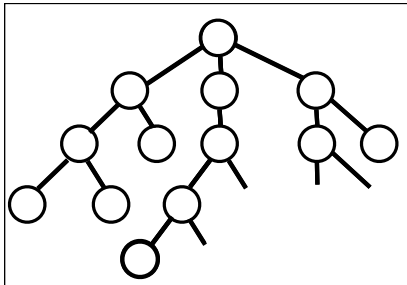
1

חיפוש לרוחב
Breadth-first search

- אסטרטגית חיפוש המפתחת את עץ החיפוש שכבה אחרי שכבה
- הצומת הבא לפיתוח: הצומת "הפתוח" הרדוד ביותר
- הצמתים שנשמרים בזכרון: כל הצמתים הפתוחים

© Shaul Markovitch 2005

2



© Shaul Markovitch 2005

3

```
Breadth-first-search(problem)
   OPEN ← ( make-node(problem[init-state], NIL) )
   while open ≠ ( ) do
      next-node ← pop(OPEN)
      loop for s in expand(next-node[state])
         new ← make-node(s,next-node)
         if problem[goal-test](s) then
            return(path(new))
         OPEN ← OPEN || (new)
```

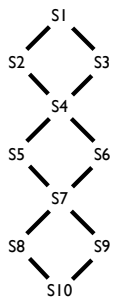
© Shaul Markovitch 2005

4

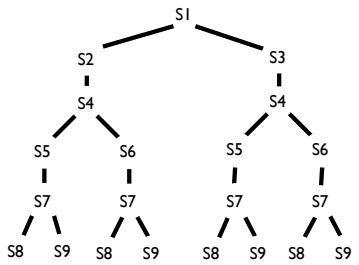
חיפוש בגרף מצבים

- האלגוריתם פורש במהלך החיפוש עץ
- האלגוריתם יעבוד נכון גם כאשר מרחב המצבים הוא גרף
- במקרה כזה יתכן שצמתים רבים ייצגו את אותו המצב
- יעילות האלגוריתם תיפגע מכיוון שהוא יבצע חיפושים חוזרים בתת-גרפים
- במקרה קיצוני יתכן וגודל עץ החיפוש יהיה אקפוננציאלי בגודל מרחב המצבים

© Shaul Markovitch 2005



© Shaul Markovitch 2005



© Shaul Markovitch 2005

התאמת BFS לחיפוש בגרפים

- ניתן למנוע חיפוש חוזר ע"י שמירת הצמתים שפותחו
- דרישות הזכרון לא תגדלנה מכיוון שכל צומת שלא פותח החזיק למעשה את כל הצמתים במסלול לשורש לשם שחזור הפתרון

© Shaul Markovitch 2005

```
Breadth-first-search-G(problem)
   OPEN ← ( make-node(problem[init-state], NIL) )
   CLOSE ← NIL
   while open ≠ ( ) do
      next-node ← pop(OPEN)
      CLOSE ← CLOSE ∪ {next-node}
      loop for s in expand(next-node[state])
         if not(find-state(s,OPEN) or
               find-state(s,CLOSE)) then
            new ← make-node(s,next-node)
            if problem[goal-test](s) then
               return(path(new))
            OPEN ← OPEN || (new)

find-state(state,node-list)
   for node in node-list
      if node[state]=state then return TRUE
   return FALSE
```

© Shaul Markovitch 2005

תכונות אלגוריתם BFS: שלמות

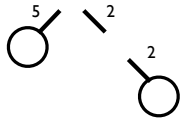
טענה: האלגוריתם שלם. כלומר מובטח שהוא ימצא פתרון לבעיה פתירה

הוכחה: קיים לבעיה לפחות פתרון אחד
יהי L אורך הפתרון המינימלי
יהי b מקדם הסיעוף של גרף המצבים
אחרי b^L צעדים בדק האלגוריתם
את כל המסלולים האפשריים באורך L
ולכן בהכרח מצא את הפתרון

© Shaul Markovitch 2005

אופטימליות של חיפוש לרוחב

במקרה הכללי לא מוצא האלגוריתם את מסלול הפתרון הזול ביותר



© Shaul Markovitch 2005

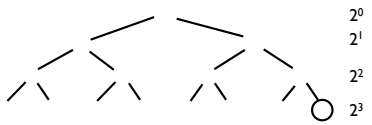
אופטימליות של חיפוש לרוחב

- האלגוריתם מוצא את המסלול הקצר ביותר
- אם מחיר כל קשת הוא 1 (או מחיר כלשהוא אחיד) אזי האלגוריתם הוא אופטימלי

© Shaul Markovitch 2005

תכונות אלגוריתם BFS: סיבוכיות זמן וזכרון

- אם אורך הפתרון הקצר ביותר הוא L אזי במקרה הגרוע יפתח האלגוריתם עץ מלא בעומק L



© Shaul Markovitch 2005

13

תכונות אלגוריתם BFS: סיבוכיות זמן וזכרון

- מספר הצמתים שנוצרים יהיה לכן

$$b^0 + b^1 + b^2 + \ldots + b^L = \frac{b^{L+1} - 1}{b - 1} = O(b^L)$$

- מכיוון שכל הצמתים נשארים בזכרון זוהי סיבוכיות הזמן וגם סיבוכיות הזכרון
- במקרה של גרפים יתכן שמספר המצבים קטן בהרבה מהחסם ולכן תהיה סיבוכיות הזמן והזכרון

$$O(\min(b^L, |S|))$$

© Shaul Markovitch 2005

14

בעיית הזכרון

משאבי מחשב לחיפוש לרוחב כאשר מקדם הסיעוף הינו 10, המחשב מעבד 10,000 צמתים לשניה וכל צומת דורש 1000 בתים

זכרון	זמן	צמתים	עומק
111 KB	0.01 sec	111	2
11 MB	1.11 sec	11,111	4
1 GB	111 sec	10^6	6
100 GB	3 hrs	10^8	8
10 TB	12 days	10^{10}	10
1 PB	3 years	10^{12}	12
100 PB	300 years	10^{14}	14

© Shaul Markovitch 2005

15

חיפוש לעומק
Depth-first search

- הצומת הראשון לפיתוח הינו הצומת המייצג את מצב ההתחלה
- האלגוריתם עוצר כאשר הוא מגיע למצב מטרה
- **הצומת הבא לפתוח הינו הצומת העמוק ביותר**
- בין צמתים בעלי עומק שווה נבחר הצומת הבא שרירותית (למשל לפי סדר ההופעה)
- נשמרים בזכרון רק צמתים שלא נחקרו לגמרי כלומר שתת העץ תחתם לא פותח במלואו

© Shaul Markovitch 2005

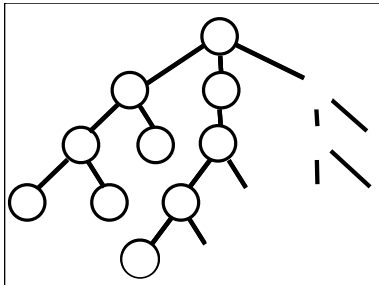
16

```
DFS (state, goal-test)
  if goal-test(state) then return((state))
  else
    for c in succ(state)
      solution ← DFS(c, goal-test)
      if solution ≠ FAIL then
        return(state || solution)
    return(FAIL)


DFS-search (problem)
  DFS(problem[init-state], problem[goal-test])
```

© Shaul Markovitch 2005

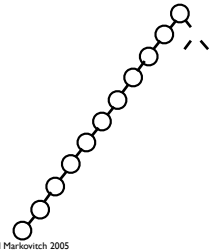
17



© Shaul Markovitch 2005

18

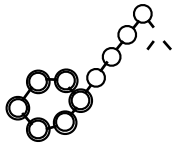
בעיה – ענף אינסופי



© Shaul Markovitch 2005

19

בעיה – מעגל



© Shaul Markovitch 2005

20

חיפוש לעומק – שלמות

מסקנה מבעיית הענף האינסופי ובעיית המעגל: האלגוריתם הבסיסי שהצגנו אינו שלם

© Shaul Markovitch 2005

אלגוריתם חיפוש לעומק עם הגבלת עומק

- בגלל הבעיות שהצגנו משתמשים בד"כ בהגבלה על עומק החיפוש
- הגבלת עומק החיפוש תמנע התקעות בענפים אינסופיים
- הגבלת עומק החיפוש תגביל את מספר הפעמים שנוכל להסתובב במעגל

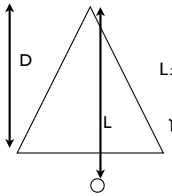
© Shaul Markovitch 2005


```
DFS-L (state, goal-test,depth)
  if goal-test(state) then return((state))
  else
    if depth=0 then return FAIL
    for c in succ(state)
       solution ← DFS(c, goal-test,depth-1)
       if solution ≠ FAIL then
          return(state || solution)
    return(FAIL)


DFS-L-search (problem)
  DFS(problem[init-state], problem[goal-test],L)
```

© Shaul Markovitch 2005

חיפוש מוגבל עומק – שלמות

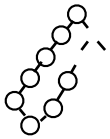


- בגלל הגבלת העומק מובטח לנו שהאלגוריתם יעצור
- האלגוריתם שלם כאשר $L \le D$
- האלגוריתם אינו שלם כאשר $L > D$ – הוא יעצור ללא פתרון כאשר קיים פתרון

© Shaul Markovitch 2005

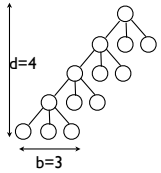
חיפוש מוגבל עומק – אופטימליות

ברור שהאלגוריתם לא מוצא את הפתרון הזול ביותר וגם לא את הפתרון הקצר ביותר



© Shaul Markovitch 2005

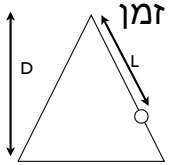
סיבוכיות זכרון



- b מקדם סיעוף
- d עומק נוכחי
- D הגבלת העומק
- $b \cdot d$ מספר הצמתים שנשמר בכל רגע
- $O(b \cdot D)$ סיבוכיות הזכרון

© Shaul Markovitch 2005

סיבוכיות זמן



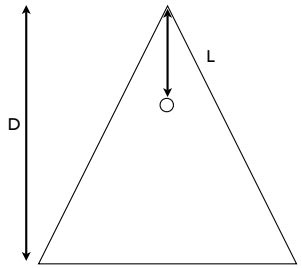
- במקרה הגרוע האלגוריתם מייצר עץ מלא בעומק D
- לכן סיבוכיות הזמן של האלגוריתם הינה $O(b^d)$

© Shaul Markovitch 2005

רגישות להגבלת העומק

- ישנן בעיות כמו בעיית הסוס בהן ידוע בדיוק עומק הפתרון
- בבעיות בהן לא ידוע עומק הפתרון יש להטיל מגבלת עומק גדולה מספיק כדי להבטיח שלמות
- הבעיה היא שהגבלת עומק גדולה מידי גורמת לבזבוז עצום
- אם מקדם הסיעוף הינו 10 אזי חיפוש לעומק עם הגבלה של 6 יותר מעומק המטרה יגרום לחיפוש שצורך פי מליון זמן מאשר חיפוש לרוחב

© Shaul Markovitch 2005

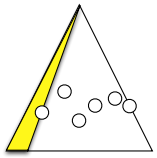


© Shaul Markovitch 2005

29

יעילות

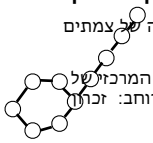
האלגוריתם יעיל במיוחד כאשר קיימים מצבי מטרה רבים המפוזרים על פני המרחב



© Shaul Markovitch 2005

30

חיפוש לעומק בגרף



- ניתן להוסיף לאלגוריתם רשימה של צמתים שפותחו כמו בחיפוש לרוחב
- תוספת כזו מבטלת את היתרון המרכזי של חיפוש לעומק על פני חיפוש לרוחב: זכרון לינארי לעומת אקפוננציאלי
- מכיוון שהמסלול מהשורש לצומת הנוכחי שמור ממילא בזכרון, ניתן לבדוק באם צומת חדש נמצא במסלול זה ולמנוע מעגלים
- תוספת כזו תייקר את מחיר הפיתוח של כל צומת

© Shaul Markovitch 2005

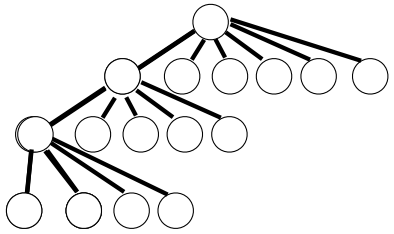
31

חיפוש backtracking

- דומה לחיפוש לעומק
- במקום לפתח את כל הבנים של צומת, מייצר בן אחד, חוקר אותו לעומק, מייצר בן שני וכו
- ניתן להשתמש בו כאשר פונקציית המעבר נתונה באמצעות קבוצת אופרטורים
- חסכוני יותר בזכרון מחיפוש לעומק מכיוון שרק בן אחד נשמר בכל צומת
- כדאי להשתמש בו כאשר מקדם הסיעוף גדול

© Shaul Markovitch 2005

32



© Shaul Markovitch 2005

33

חיפוש לרוחב מול חיפוש לעומק

- חיפוש לעומק מאפשר לעבוד עם זכרון לינארי אך תלוי בהגבלת העומק והמסלול שמוצא אינו אופטימלי
- חיפוש לרוחב אינו תלוי בהגבלת עומק ומוצא מסלול קצר ביותר אך דורש זכרון אקפוננציאלי

© Shaul Markovitch 2005

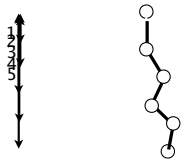
34

חיפוש העמקה הדרגתית
Iterative Deepening

- חיפוש העמקה הדרגתית משלב את היתרונות של חיפוש לרוחב וחיפוש לעומק
- האלגוריתם מפעיל חיפוש לעומק עם הגבלת עומק 1
- אם אינו מוצא פתרון מפעיל חיפוש לעומק עם הגבלת עומק 2 וכך הלאה
- האלגוריתם אינו זוכר דבר בין החיפושים מלבד הגבלת העומק באיטרציה הקודמת

© Shaul Markovitch 2005

35



© Shaul Markovitch 2005

36

שלמות

- אלגוריתם החיפוש ע"י העמקה הדרגתית הינו שלם
- אם עומק הפתרון הוא L
  אזי האלגוריתם ימצא אותו באיטרציה ה L

© Shaul Markovitch 2005

אופטימליות

- אלגוריתם ההעמקה הדרגתית מוצא את הפתרון הקצר ביותר
- נניח שמצא פתרון באורך L
- לא יתכן שקיים פתרון קצר יותר אחרת היה מוצא אותו באיטרציה קודמת

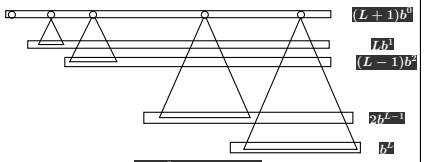
© Shaul Markovitch 2005

סיבוכיות זכרון

- דרישת הזכרון היא לינארית כמו חיפוש לעומק
- בעוד שחיפוש לעומק לינארי בהגבלת העומק, חיפוש העמקה הדרגתית לינארי באורך הפתרון הקצר ביותר

© Shaul Markovitch 2005

סיבוכיות זמן



$$t = \sum_{i=0}^{L}(L+1-i)b^i$$

© Shaul Markovitch 2005

חיפוש העמקה הדרגתית
לעומת חיפוש לרוחב – דוגמא

$$b = 4, L = 20$$

$$Time(BFS) = \frac{4^{21} - 1}{4 - 1} = 1,466,015,503,701$$

$$Time(ID) = \sum_{i=0}^{20} (20 + 1 - i)4^i = 1,954,687,338,261$$

$$Memory(BFS) = Time(BFS) = 1,466,015,503,701$$

$$Memory(ID) = 4 \cdot 20 = 80$$

$$\frac{Time(ID)}{Time(BFS)} = 1.33333$$

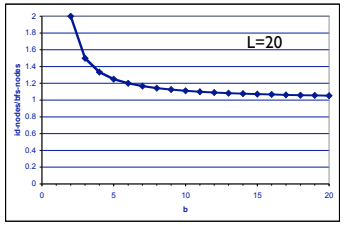
$$\frac{Memory(BFS)}{Memory(ID)} = 1.8 \cdot 10^{10}$$

כלומר: ירידה בזכרון בפקטור של ביליון
תמורת תוספת של
33 אחוז בזמן

© Shaul Markovitch 2005

יחס הזמן בין העמקה הדרגתית לבין חיפוש לרוחב

$$\frac{\sum_{i=0}^{L}(L+1-i)b^i}{\frac{b^{L+1}-1}{b-1}}$$



© Shaul Markovitch 2005

הערה

- השקף האחרון הראה למעשה את היחס בין מספר הצמתים שנוצרים בכל אחת מהשיטות
- אם היינו מחדדים זמן אזי הפער היה מצטמצם עוד יותר בגלל הבדלי מימוש
- בחיפוש לרוחב יש להחזיק רשימה של מצבים פתוחים. כשיוצרים מצב חדש צריך ליצור העתק של מצב האבא
- בחיפוש לעומק אפשר להפעיל את האופרטורים על המצב המקורי ובנסיגה לבטל

© Shaul Markovitch 2005


```
? (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:      0.00
Depth:  13 Nodes:         13,914 Time:      0.00
Depth:  14 Nodes:         24,511 Time:      0.00
Depth:  15 Nodes:         42,004 Time:      0.01
Depth:  16 Nodes:         73,289 Time:      0.01
Depth:  17 Nodes:        125,774 Time:      0.02
Depth:  18 Nodes:        220,659 Time:      0.05
Depth:  19 Nodes:        378,120 Time:      0.06
Depth:  20 Nodes:        660,733 Time:      0.11
Depth:  21 Nodes:      1,133,122 Time:      0.21
Depth:  22 Nodes:      1,985,063 Time:      0.33
Depth:  23 Nodes:      3,402,236 Time:      0.58
Depth:  24 Nodes:      5,949,873 Time:      0.99
Depth:  25 Nodes:     10,201,398 Time:      1.73
Depth:  26 Nodes:     10,949,317 Time:      1.86
? (id #(3 3 0 1 1 2 2 3 0 1 0 3 2 3 2 1 0 3 0 1 2 3 2)
```


© Shaul Markovitch 2005

חיפוש מחיר-אחיד
Uniform-cost search

- חיפוש לרוחב מוצא פתרון אופטימלי כאשר פונקצית המחיר אחידה - כלומר מחיר כל הקשתות זהה
- חיפוש מחיר אחיד מוצא פתרונות אופטימליים - בעלי מחיר מינימלי - גם במרחבי חיפוש עם פונקציית מחיר לא אחידה

© Shaul Markovitch 2005

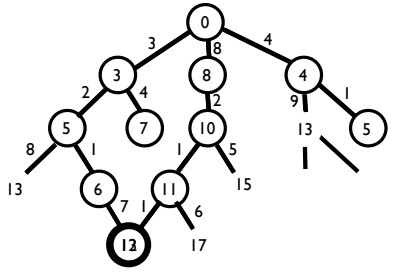
חיפוש אחיד - האלגוריתם

- האלגוריתם שומר כמו חיפוש לרוחב רשימה של צמתים פתוחים
- הצומת הבא לפיתוח הוא הצומת שמחיר המסלול אליו מינימלי
- האלגוריתם עוצר כאשר צומת בעל מחיר מסלול מינימלי הינו צומת מטרה
- למבנה הנתונים של צומת מוסיפים שדה נוסף שמכיל את המחיר לצומת

© Shaul Markovitch 2005


```
Uniform-cost-search(problem)
    OPEN ← ( make-node(problem[init-state], NIL,0) )
    CLOSE ← NIL
    while open ≠ ( ) do
        next ← pop(OPEN)
        CLOSE ← CLOSE ∪ {next}
        if problem[goal-test](next) then
            return(path(new))
        loop for s in expand(next[state])
            if not(find-state(s,CLOSE) then
                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[parent] ← next
                        insert(old-node,OPEN-{old-node},g)
                                ;Insert an item into a sorted list according to field g
                else                    ; no node with same state in open
                    new ← make-node(s,next,new-cost)
                    insert(new,OPEN,g)
        return(FAIL)                    ; OPEN is empty - no solution
```

© Shaul Markovitch 2005



© Shaul Markovitch 2005

שלמות

ללא תנאי נוסף האלגוריתם אינו שלם

© Shaul Markovitch 2005

שלמות

אם פונקציית המחיר חסומה מלמטה בחסם חיובי, כלומר

$\exists \delta > 0, \forall s \in S, \forall t \in Succ(s) | cost(s,t) \geq \delta |$

אזי חיפוש מחיר–אחיד הינו שלם

הוכחה

נניח שקיים פתרון שמחירו C

ונניח שמקדם הסיעוף הינו b

אזי מובטח שלאחר $\frac{b^{\lceil \frac{C}{\delta} \rceil + 1} - 1}{b - 1}$

צעדים האלגוריתם יעצור ויחזיר פתרון

© Shaul Markovitch 2005

אופטימליות

האלגוריתם מחזיר את מסלול הפתרון בעל המחיר המינימלי

הוכחה

- האלגוריתם עוצר ומחזיר פתרון כאשר הצומת הראשונה ברשימת הפתוחים היא צומת מטרה
- מכיוון שהרשימה ממוינת לפי מחיר המסלולים, מובטח שכל המסלולים החלקיים הם בעלי מחיר גדול או שווה למחיר שנמצא

© Shaul Markovitch 2005

סיבוכיות מקום וזמן

$\frac{b^{\lceil \frac{C}{\delta} \rceil + 1} - 1}{b - 1}$

© Shaul Markovitch 2005