

2. Visszalépéses stratégia

- A visszalépéses kereső rendszer olyan KR, amely
 - globális munkaterülete:
 - út a startcsúcsból az aktuális csúcsba (ezen kívül a még ki nem próbált élek nyilvántartása)
 - keresés szabályai:
 - a nyilvántartott út végéhez egy új (ki nem próbált) él hozzáfűzése, vagy az legutolsó él törlése (visszalépés szabálya)
 - vezérlés stratégiája a visszalépés szabályát csak a legvégső esetben alkalmazza

1

- kiinduló értéke a startcsúcs, pontosabban a startcsúcsot tartalmazó nulla hosszúságú út.
- terminálási feltétel akkor következik be, ha az aktuális út végén megjelenik egy célcsúcs, vagy ha a startcsúcsból vissza akarunk lépni.

2

Visszalépés feltételei az aktuális útra

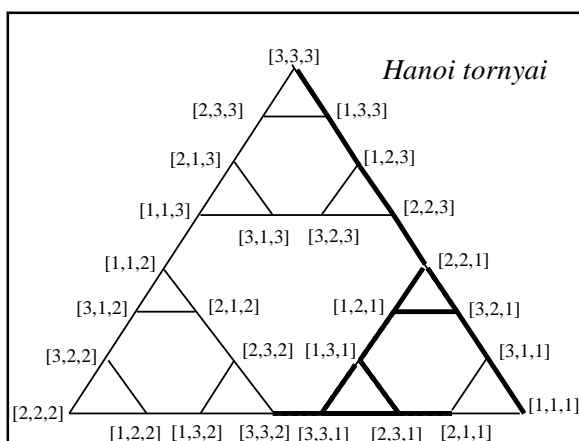
- zsákutca, azaz végpontjából nem vezet tovább út
- zsákutca torkolat, azaz végpontjából kivezető utak nem vezetnek célba
- kör, azaz végpontja megegyezik az út egy megelőző csúcsával
- mélységi korlátnál hosszabb

3

Heurisztikák

- sorrendi heurisztika:
 - sorrendet ad végpontból kivezető élek (utak) vizsgálatára
- vágó heurisztika:
 - megjelöli azokat a végpontból kivezető éleket (utakat), amelyeket nem érdemes megvizsgálni

4



2.1. Első változat

- A visszalépéses algoritmus első változata az, amikor a visszalépés feltételei közül az első kettőt építjük be a kereső rendszerbe.
- Rekurzív algoritmussal (VL1) szokták megadni
 - Indítás: $megoldás \leftarrow VLI(startcsúcs)$

6

Recursive procedure $VLI(csúcs)$ return megoldás

1. if $cél(csúcs)$ then return(nil) endif
2. $élek \leftarrow kivezető-élek(csúcs)$
3. while not $üres(élek)$ loop
4. $él \leftarrow kivesz-egy(élek)$
5. $megoldás \leftarrow VLI(vég(él))$
6. if $megoldás \neq hiba$ then
 return($hozzáfűz(él, megoldás)$) endif
7. endloop
8. return($hiba$)
- end

2.1. Tétel

- A VL1 véges körmentes irányított gráfokon (nem kell δ -gráf) mindig terminál, és ha létezik megoldás, akkor talál egy megoldást.
- Bizonyítás:
- Egy n csúcsot tartalmazó körmentes gráfban a startcsúcsból kivezető utak száma véges
 - (legfeljebb $n-1$ db egyhosszú út, $(n-1) \cdot (n-2)$ db kettőhosszú út, ..., és $(n-1)!$ db $n-1$ hosszú út van)
- amelyek közül a VL1 minden lépésben egyet megvizsgál.

8

Kitérés:

Sajátos tulajdonságú problémák

- Néhány probléma (n -királynő, gráfszínezés, keresztrejtvény, órarend) állapot gráfja bizonyos szabályosságot mutat:
 - fa,
 - azonos hosszú ágak,
 - egy szinten mindenhol azonos számú leágazás.
- Ezeket sajátos reprezentációs modell keretében fogalmazhatjuk meg: direktszorzat alakú problématerben relációkkal (korlátokkal, megszorításokkal) írjuk körül a megoldást.

9

Megszorítás kezelés

- Megszorítás kezelés vagy korlátozás kielégítés (constraint satisfaction) problémáról akkor beszélünk, amikor
- adott D_1, \dots, D_n véges halmazokra egy olyan $(x_1, \dots, x_n) \in D_1 \times \dots \times D_n$ érték-együttest kell találni (a $D_1 \times \dots \times D_n$ a problémater), amely
- megfelel az ugyancsak adott C_1, \dots, C_m relációknak (a helyes választ írják le), ahol $C_i \subseteq D_{i_1} \times \dots \times D_{i_{k_i}}$

10

Példa

- Gráfszínezés (n csúcs, m szín)
 - i -edik csúcs színei: $D_i = \{1, \dots, m\}$ ($i=1..n$)
 - Megszorítások a szomszédos (i,j) csúcspárokra:
 - $C_{ij}(x_i, x_j) = (x_i \neq x_j)$
- n -királynő probléma
 - i -edik sor pozíciói: $D_i = \{1, \dots, n\}$ ($i=1..n$)
 - Megszorítások minden (i,j) sorpárra:
 - $C_{ij}(x_i, x_j) = (x_i \neq x_j \text{ és } |x_i - x_j| \neq |i - j|)$

11

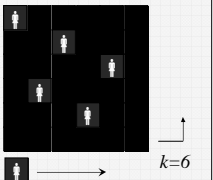
Szűrések

- A VL1-nél hatékonyabb keresést kapunk, ha a probléma terét előbb megsűrjük (reprezentációs gráfot vágjuk).
- A bináris megszorítások esetén az i -dik alaphalmazt a j -edik alaphalmaz alapján szűkíthetjük.
 - $S(i,j) : D_i \leftarrow \{e \in D_i \mid \exists f \in D_j : C_{ij}(e,f)\}$

12

VLI

A 3. reprezentáció szerint helyezzük el a királynőket, soronként egyet-egyét, minden sorban balról jobbra haladva



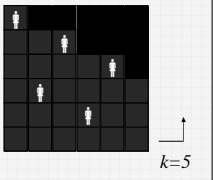
$k=6$

13

VLI

Miután (a k -edik sorban) elhelyezünk egy királynőt, a hátralevő sorokban ($k+1..n$) kiszűrjük az ütésben álló mezőket:

$\text{for } i=k+1..n \text{ do } S(i,k)$



$k=5$

ha egy királynőt nem tudunk szabad mezőre állítani, azaz $D_k=\emptyset$, akkor visszalépünk.

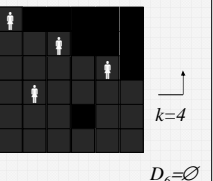
A 3. repr. feletti VL a fenti szűréssel kiegészítve éppen a 4. repr. felett végrehajtott VL lesz.

14

FC

FC algoritmus:
Az előbb bevezetett

$\text{for } i=k+1..n \text{ do } S(i,k)$



$k=4$

során, ha a hátralevő sorok valamelyike már nem tartalmaz szabad mezőt (azaz $D_i=\emptyset$), akkor azonnal visszalépünk.

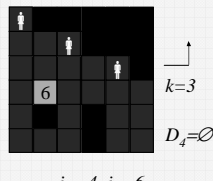
$D_6=\emptyset$

15

PLF

PLF algoritmus:
Az FC algoritmus után lefuttatjuk az alábbi szűrést

$\text{for } i=k+1..n \text{ do}$
 $\text{for } j=i+1..n \text{ do } S(i,j)$



$k=3$

$D_4=\emptyset$

$i=4, j=6$

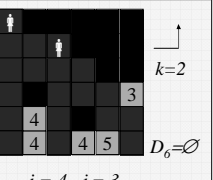
ha van a hátralevő sorok közt olyan, amelyre $D_i=\emptyset$, akkor visszalépünk.

16

LF

LF algoritmus:
Az FC algoritmus után lefuttatjuk az alábbi szűrést

$\text{for } i=k+1..n \text{ do}$
 $\text{for } j=k+1..n \text{ és } i \neq j \text{ do } S(i,j)$



$k=2$

$D_6=\emptyset$

és ha $D_i=\emptyset$, akkor visszalépünk.

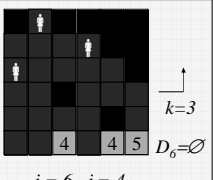
$i=4, j=3$
 $i=5, j=4$
 $i=6, j=4$
 $i=6, j=5$

17

LF még egyszer

LF algoritmus:
Az FC algoritmus után lefuttatjuk az alábbi szűrést is

$\text{for } i=k+1..n \text{ do}$
 $\text{for } j=k+1..n \text{ és } i \neq j \text{ do } S(i,j)$



$k=3$

$D_6=\emptyset$

és ha $D_i=\emptyset$, akkor visszalépünk.

$i=6, j=4$
 $i=6, j=5$

18

AC1

AC1 algoritmus:

Az FC algoritmus után lefuttatjuk az alábbi szűrést is

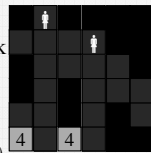
repeat

for $i=k+1 \dots n$ do

for $j=k+1 \dots n$ és $i \neq j$ do $S(i,j)$

until volt törlés

és ha $D_i = \emptyset$, akkor visszalépünk.



1. menet

$i = 6, j = 4$

19

AC1

AC1 algoritmus:

Az FC algoritmus után lefuttatjuk az alábbi szűrést is

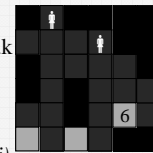
repeat

for $i=k+1 \dots n$ do

for $j=k+1 \dots n$ és $i \neq j$ do $S(i,j)$

until volt törlés

és ha $D_i = \emptyset$, akkor visszalépünk.



2. menet

$i = 5, j = 6$

20

AC1

AC1 algoritmus:

Az FC algoritmus után lefuttatjuk az alábbi szűrést is

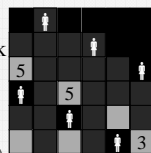
repeat

for $i=k+1 \dots n$ do

for $j=k+1 \dots n$ és $i \neq j$ do $S(i,j)$

until volt törlés

és ha $D_i = \emptyset$, akkor visszalépünk.



3. menet

$i = 3, j = 5$

$i = 4, j = 5$

$i = 6, j = 3$

21

ACx

- AC3 csak azokat az (i,j) párokat vizsgálja meg újra, ahol az D_j változott. Az előbbi példában az AC3 csak 27-szer hajtja végre a $S(i,j)$ -t, míg az AC4 48-szor.
- AC4 nem vizsgálja újra azokat az (i,j) párokat, ahol az D_j változott ugyan, de ez nincs hatással az D_i -re.
 - Ha minden $e \in D_i$ -nél megjelöltünk egy olyan $f \in D_j$ -t, amelyre $C_{ij}(e,f)$, majd egy nem jelölt elemet törölünk a D_j -ből, akkor nincs szükség újabb $S(i,j)$ -re.
- AC7 kihasználja a bináris megszorítások szimmetriáját.

22

Megjegyzés

□ Kérdések:

- Megoldhatóság feltétele:
 - él-konzisztencia, k -konzisztencia
- Meg lehet-e visszalépés nélkül oldani egy problémát?
- Hatékony algoritmusok keresése

23

Visszatérés: Másodlagos vezérlési stratégiák

□ Vágó stratégiák:

- Reprezentációs gráf specialitását kihasználó vágó algoritmust építünk be VL1-be. (ACx algoritmus)

□ Sorrendi stratégiák:

- A legkisebb tartományú változó kitöltését részesítjük előnyben.
- Az egymástól függő változók egymáshoz közel szerepeljenek a vizsgálatban.

24

2.2. Második változat

- ❑ A visszalépéses algoritmus második változata az, amikor a visszalépés feltételei közül mindet beépítjük a kereső rendszerbe.
- ❑ Rekurzív algoritmussal (VL2) adjuk meg
 - Indítás: $megoldás \leftarrow VL2(<startcsúcs>)$

25

Recursive procedure $VL2(út)$ return $megoldás$

1. $csúcs \leftarrow utolsó-csúcs(út)$
 2. if $cél(csúcs)$ then return(nil)
 3. if $hossza(út) \geq korlát$ then return($hiba$)
 4. if $csúcs \in maradék(út)$ then return($hiba$)
 5. $élek \leftarrow kivezető-élek(csúcs)$
 6. while not $üres(élek)$ loop
 7. $él \leftarrow kivesz-egy(élek)$
 8. $megoldás \leftarrow VL2(fűz(út, vég(él)))$
 9. if $megoldás \neq hiba$ then return($fűz(él, megoldás)$)
 10. endloop
 11. return($hiba$)
- end

2.2. Tétel

- ❑ A VL2 δ -gráfban mindig terminál. Ha létezik a mélységi korlátnál nem hosszabb megoldás, akkor megtalál egy megoldást.
- ❑ Bizonyítás:
- ❑ Egy δ -gráfban a startcsúcsból kivezető, MK-nál nem hosszabb utak száma véges
 - (legfeljebb σ db egyhosszú út, σ^2 db kettő-hosszú út, ..., σ^{MK} db MK hosszú út van)
- ❑ amelyek közül minden lépésben egyet vizsgálunk.

27

Megjegyzés

- ❑ Ha csak a megadott mélységi korlátnál hosszabb megoldási út van, akkor az algoritmus bár terminál, de nem talál megoldást.
- ❑ A mélységi korlát önmagában biztosítja a terminálást körök esetére is.
 - A mélységi korlát ellenőrzése jóval gyorsabb és kevesebb memóriát igényel, mint a körfigyelés.

28

Értékelés

- | ❑ ELŐNYÖK | ❑ HÁTRÁNYOK |
|---------------------------|--|
| – könnyen implementálható | – nem ad optimális megoldást. (iterációba szervezhető) |
| – kicsi memória igényű | – kezdetben hozott rossz döntést csak sok visszalépés korrigál (visszaugrások keresés) |
| | – egy zsákutca részt többször is bejárhat a keresés |

29

GYAKORLAT

30

VL1 algoritmus az n-királynő probléma megoldására

```
bool BT1(Table& t) {
    if (t.Goal()){
        t.Write();
        return true;
    }
    Iterator it(t);    // heurisztika
    for (it.First(); !it.End(); it.Next()){
        Table tnew(t);
        if(tnew.Put(it.Current()))
            if (BT1(tnew)) return true;
    }
    return false;
}
```

31




Rákövetkező állapotok bejárása az n-királynő problémában

```
class Iterator{
public:
    Iterator(Table& t); // hiányzó rész
    int Current() const {return h[current]+1;}
    void First() {current = 0;}
    bool End() const {return current==n;}
    void Next() {current++;}
    ~Iterator() { delete[] h; }
private:
    int* h;    // bejárési sorrendet rögzíti
    int n;
    int current;
};
```

32

Heurisztikák az n-királynő problémára 1.



3. reprezentáció mellett

Neminformált:	Diagonális heurisztika:	Diagonális és pti-ps heurisztika
		
22 visszalépés	2 visszalépés	0 visszalépés

33

Heurisztikák az n-királynő problémára 2.

4. reprezentáció mellett

Neminformált:	Diagonális heurisztika:
	
4 visszalépés	0 visszalépés

34

```
class Iterator{ // diagonális heurisztika
public:
    Iterator(Table& t);
    int Current() const {return h[current]+1;}
    void First() {current = 0;}
    bool End() const {return current==n;}
    void Next() {current++;}
    ~Iterator() { delete[] h; }
private:
    int* h;    // sorrendet rögzíti
    int n;
    int current;
};
```

```
Iterator::Iterator(Table& t)
{
    n = t.n;
    h = new int[n];
    for (int i=0; i<n; i++) h[i] = i;
    int* d = new int[n];
    for (int j=0; j<n; j++) d[j] = diag(t.row+1, j+1, n);
    for (int i=0; i<n; i++){
        int ind = 0; int max = d[h[0]];
        for (int j=1; j<i; j++){
            if (max<d[h[j]]){
                max = d[h[j]];
                ind = j;
            }
        }
        int e = h[ind]; h[ind] = h[i]; h[i] = e;
    }
    delete[] d;
}
```

Heurisztikák a 8-as játékra

- Rossz helyen levő cellák száma:
 - $W(a) = \sum_{i,j} X(a_{i,j} \neq \text{cél}_{i,j})$
- Rossz helyen levő cellák minimális távolsága a célbeli helyüktől:
 - $P(a) = \sum_{i,j} |(i,j), \text{célhely}(a_{i,j})|$

37

