

A mesterséges intelligencia alapjai

Az előadások mellé vetített anyag

Várterész Magda

A mesterséges intelligencia alapjai: Az előadások mellé vetített anyag

Várterész Magda



A tananyag a TÁMOP-4.1.2-08/1/A-2009-0046 számú Kelet-magyarországi Informatika Tananyag Tárház projekt keretében készült. A tananyagfejlesztés az Európai Unió támogatásával és az Európai Szociális Alap társfinanszírozásával valósult meg.



Nemzeti Fejlesztési Ügynökség <http://ujszechenyiterv.gov.hu/> 06 40 638-638



Table of Contents

1. A problémareprezentáció	1
1. Az állapottér-reprezentáció	1
2. Példák állapottér-reprezentációra	4
3. Az állapottérgráf	5
2. Megoldást kereső rendszerek	8
1. Nemmodosítható megoldáskereső rendszerek	9
2. Modosítható megoldáskereső rendszerek	11
2.1. A visszalépéses megoldáskereső algoritmus	11
2.2. Visszalépéses megoldáskereső köröket is tartalmazó gráfokban	13
2.3. Ág és korlát algoritmus	15
2.4. Keresőfával keresők	16
2.5. Szélességi és mélységi keresők	17
2.5.1. A szélességi kereső értékelése	18
2.5.2. A mélységi kereső értékelése	19
2.6. Optimális kereső	19
2.6.1. Az optimális kereső értékelése	20
2.7. Best-first algoritmus	21
2.7.1. A best-first algoritmus értékelése	21
2.8. Az A algoritmus	22
2.8.1. Az A algoritmus értékelése	23
2.9. A monoton A algoritmus	27
3. Kétszemélyes stratégiai játékok és lépésajánló algoritmusok	29
1. A játékok reprezentációja	29
2. A stratégia	31
3. Minimax algoritmus	32
3.1. Az algoritmus fő lépései	32
4. Negamax algoritmus	33
4.1. Az algoritmus fő lépései:	33
4. Problémamegoldás redukcióval	35
1. Problémaredukciós reprezentáció	35
2. Példák problémaredukciós reprezentációra	37
2.1. Hanoi tornyai	37
3. A problémaredukciós reprezentációt szemléltető gráf	38
4. Problémaredukcióval reprezentált feladatok megoldáskereső módszerei	40
4.1. Visszalépéses megoldáskereső ÉS/VAGY fák esetén	40
4.2. Keresőfával megoldáskereső ÉS/VAGY fák esetén	41

List of Tables

1.1. Egy-egy pozícióra hivatkozás: (sor,oszlop)	4
---	---

Chapter 1. A problémareprezentáció

A mesterséges intelligencia problémáinak megoldása a probléma megfogalmazásával kezdődik: a problémát leírjuk, *reprezentáljuk*. Az egyik legelterjedtebb reprezentációs technika az *állapottér-reprezentáció* (*state space representation*).

1. Az állapottér-reprezentáció

Legyen adott egy probléma, amit jelöljünk P -vel.

- Megkeressük P világának legalább egy, de véges sok — a probléma megoldása során fontosnak vélt — meghatározóját. (pl. objektum, pozíció, méret, hőmérséklet, szín, stb.) Tegyük fel, hogy m ilyen jellemzőt találtunk.
- Minden egyes jellemző P világát különböző értékekkel jellemzi. (pl. szín: fekete/fehér; hőmérséklet: $[-20^\circ, 40^\circ]$, stb)

Ha a megadott jellemzők épp rendre a h_1, \dots, h_m értékekkel rendelkeznek azt mondjuk, hogy

$$p \tag{1.1}$$

világa a (h_1, \dots, h_m) érték m -essel leírt *állapotban* (*state*) van. A világunk állapotainak halmaza az *állapottér* (*state space*).

Jelölje az i -edik jellemző által felvehető értékek halmazát H_i ($i = 1, \dots, m$). Ekkor P állapotai elemei a

$$H_1 \times \dots \times H_m \tag{1.2}$$

halmaznak.

Azokat a feltételeket, amelyek meghatározzák, hogy ebből a halmazból mely érték m -esek állapotok *kényszerfeltételeknek* nevezzük.

Az állapottér tehát az értékhalmazok Descartes-szorzatának a kényszerfeltételekkel kijelölt részhalmaza:

$$\mathcal{A} = \{a \mid a \in H_1 \times \dots \times H_m \text{ és kényszerfeltétel}(a)\} \tag{1.3}$$

Az \mathcal{A} állapottér azon állapotát, amit a probléma világa jellemzőinek kezdőértékei határoznak meg, *kezdőállapotnak* (*initial state*) nevezzük és *kezdő*-vel jelöljük.

A kezdőállapotból kiindulva a probléma világának sorban előálló állapotait rendre meg szeretnénk változtatni, míg végül valamely számunkra megfelelő ún. *célállapotba* (*goal state*) jutunk.

Jelölje $\mathcal{C} \subseteq \mathcal{A}$ a *célállapotok halmazát*. Megadása kétféleképpen történhet:

- felsorolással: $\mathcal{C} = \{c_1, \dots, c_l \mid c_i \in \mathcal{A}, i = 1, \dots, l, l \geq 1\}$
- *célfeltételek* megadásával: $\mathcal{C} = \{c \mid c \in \mathcal{A} \text{ és célfeltételek}(c)\}$

Általában $c \subset \mathcal{A}$, hiszen $kezdő \notin c$, különben nincs megoldandó feladat.

Hogy célállapotba juthassunk, meg kell tudnunk változtatni bizonyos állapotokat. Az állapotváltozásokat leíró leképezéseket *operátoroknak* (*operator*) nevezzük.

Nem minden operátor alkalmazható feltétlenül minden állapotra, ezért meg szoktuk adni az operátorok értelmezési tartományát az *operátoralkalmazási előfeltételek* segítségével.

Jelöljön az operátorok \mathcal{O} véges halmazából o egy operátort. Ekkor

$$\text{Dom}(o) = \{a \mid a \in \mathcal{A} \text{ és } o\text{-alkalmazásának-előfeltétele}(a)\} \\ \text{és} \tag{1.4}$$

$$\text{Rng}(o) = \{o(a) \mid a \in \text{Dom}(o) \text{ és } o(a) \in \mathcal{A}\}$$

Definíció

Legyen P egy probléma. Azt mondjuk, hogy a P problémát *állapotter-reprezentáltuk*, ha megadtuk az

$$\langle \mathcal{A}, kezdő, c, o \rangle \tag{1.5}$$

négyest, azaz

- az $\mathcal{A} \neq \emptyset$ halmazt, a probléma állapotterét,
- a $kezdő \in \mathcal{A}$ kezdőállapotot,
- a célállapotok $c \subset \mathcal{A}$ halmazát és
- az operátorok $\mathcal{O} \neq \emptyset$ véges halmazát.

Jelölése: $p = \langle \mathcal{A}, kezdő, c, o \rangle$.

Legyen p egy probléma, $\langle \mathcal{A}, kezdő, c, o \rangle p$ egy állapotter-reprezentációja és legyenek $a, a' \in \mathcal{A}$.

Definíció

Az a állapotból az a' állapot *közvetlenül elérhető*, ha van olyan $o \in \mathcal{O}$ operátor, hogy

$$o\text{-alkalmazásának-előfeltétele}(a) \text{ és } o(a) = a'. \tag{1.6}$$

Jelölése: $a \Rightarrow a'$, ha fontos, hogy o állítja elő a -ból a' -t: $a \Rightarrow_o a'$.

Definíció

Az a állapotból az a' állapot *elérhető*, ha vagy $a = a'$, vagy van olyan $a_1, \dots, a_k \in \mathcal{A}$ ($k \geq 2$) (véges) állapotsorozat, hogy

$$a = a_1, \quad a' = a_k \text{ és } a_i \Rightarrow a_{i+1} \text{ minden } 1 \leq i \leq k - 1 \text{ esetén.} \tag{1.7}$$

Jelölése: $a \Rightarrow^* a'$.

Ha $a_i \Rightarrow_{o_i} a_{i+1}$ ($1 \leq i \leq k-1$) esetén, akkor van olyan o_1, o_2, \dots, o_{k-1} operátorsorozat, hogy $a \xrightarrow{o_1, \dots, o_{k-1}} a$. Ilyenkor azt mondjuk, $a \xrightarrow{o_1, \dots, o_{k-1}} a$ állapot az o_1, o_2, \dots, o_{k-1} operátorsorozat segítségével elérhető. Jelölve:

Definíció

Legyen $p = (\mathcal{A}, kezdő, c, o)$. A p probléma *megoldható* ebben az állapottér-reprezentációban, ha van olyan $c \in \mathcal{C}$ célállapot, hogy $kezdő \xrightarrow{o_1, \dots, o_r} c$ ($r \geq 1$), akkor az o_1, \dots, o_r operátorsorozat a probléma egy *megoldása*.

A feladatunk lehet

- annak eldöntése, hogy megoldható-e a probléma az adott állapottér-reprezentációban,
- egy (esetleg az összes) megoldás előállítás,
- egy (esetleg az összes) célállapot előállítás,
- valamilyen minősítés alapján jó megoldás előállítás (a megoldások között különbséget tehetünk, pl. a megoldás költsége alapján).

Jelölje $költség(o, a)$ az o operátor a állapotra alkalmazásának a költségét.

Definíció

Legyen $p = (\mathcal{A}, kezdő, c, o)$ és $o_1, \dots, o_r \in O$ a probléma egy megoldása, azaz

$$kezdő \xrightarrow{o_1, \dots, o_r} c \quad (c \in \mathcal{C}). \tag{1.8}$$

Legyen rendre $a_i \Rightarrow_{o_i} a_{i+1}$ ($1 \leq i \leq r$), ahol $a_1 = kezdő$ és $a_{r+1} = c$. Ekkor a *megoldás költsége*:

$$\sum_{i=1}^r költség(o_i, a_i). \tag{1.9}$$

Ha $költség(o, a) \equiv 1$, akkor a megoldás költsége az alkalmazott operátorok száma.

Ha az állapottér-reprezentációt valamilyen programozási nyelv segítségével szeretnénk leírni, akkor meg kell adni

- az állapotok típusát, továbbá a kényszerfeltételeket leíró, logikai értéket visszaadó függvényt;
- a kezdőállapotot egy állapot típusú konstanssal;
- a célállapotok halmazát:
 - állapot típusú konstansok felsorolásával vagy
 - a célfeltételt leíró logikai értéket visszaadó függvénnyel;

- az operátorokat: függvényekkel vagy eljárásokkal. Egy-egy operátorhoz az alkalmazásának előfeltételét leíró logikai függvény is tartozhat. Alkalmos függvénnyel ki lehet számolni (vagy táblázattal meg lehet adni) az operátorok alkalmazási költségeit is.

2. Példák állapotér-reprezentációra

A nyolcas kirakó játék

Adott nyolc számozott négyzet alakú lapocskák, melyek egy táblán 3×3 -as sémában helyezkednek el. Egy lapocskányi hely a táblán így nyilván üres. Az üres hellyel szomszédos bármelyik lapocskát az üres helyre tolnak. A feladat az, hogy adott kezdőállásból kiindulva adott célállásnak megfelelő elrendezést alakítsunk ki.

A probléma világa: a tábla a számozott lapocskákkal.

A világ leírása: a tábla egyes pozícióin épp mely számozott lapocskák vannak.

Table 1.1. Egy-egy pozícióra hivatkozás: (sor,oszlop)

pozíció	(1, 1)	(1, 2)	...	(3, 3)
lapocskák	$l_{1,1}$	$l_{1,2}$...	$l_{3,3}$

ahol $0 \leq l_{i,j} \leq 8$ minden $1 \leq i, j \leq 3$ esetén.

Tehát $H_{i,j} = H = \{0, 1, \dots, 8\}$ minden $1 \leq i, j \leq 3$ -ra.

A probléma világának egy-egy állapotát egy-egy olyan

$$\begin{pmatrix} l_{1,1} & l_{1,2} & l_{1,3} \\ l_{2,1} & l_{2,2} & l_{2,3} \\ l_{3,1} & l_{3,2} & l_{3,3} \end{pmatrix} \tag{1.10}$$

érték 9-es (3×3 -as mátrix) határozza meg, melyben az értékek H -beli elemek, és az érték 9-esben minden egyes érték H -ból pontosan egyszer fordul elő:

$$\forall i \forall j \forall s \forall o (\neg(i = s) \vee \neg(j = o) \supset \neg(l_{i,j} = l_{s,o})). \tag{1.11}$$

A kezdőállapot és a célállapot:

$$kezdő = \begin{pmatrix} 1 & 2 & 0 \\ 4 & 6 & 3 \\ 7 & 5 & 8 \end{pmatrix} \tag{1.12}$$

$$c = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \tag{1.13}$$

A számozott lapocskák tologatásai során az üres pozíciója mindig változik, az eredetihez képest *fel*, *le*, *jobbra* vagy *balra* kerül egy-egy pozícióval. Így tehát négy operátor segítségével leírhatjuk az állapotváltozásokat. A

$$fel : \begin{pmatrix} l_{1,1} & l_{1,2} & l_{1,3} \\ l_{2,1} & l_{2,2} & l_{2,3} \\ l_{3,1} & l_{3,2} & l_{3,3} \end{pmatrix} \mapsto \begin{pmatrix} l_{1,1}' & l_{1,2}' & l_{1,3}' \\ l_{2,1}' & l_{2,2}' & l_{2,3}' \\ l_{3,1}' & l_{3,2}' & l_{3,3}' \end{pmatrix} \quad (1.14)$$

operátort akkor alkalmazhatjuk, ha

$$\neg \exists i \exists j ((l_{i,j} = 0) \wedge (i = 0)). \quad (1.15)$$

Az alkalmazás eredménye, ha az éppen $l_{s,0} = 0$:

$$l_{i,j}' \Leftrightarrow \begin{cases} 0 & \text{ha } i = s - 1, j = 0 \\ l_{s-1,0} & \text{ha } i = s, j = 0 \\ l_{i,j} & \text{egyébként.} \end{cases} \quad (1.16)$$

3. Az állapottérgráf

Legyen a P probléma az $(\mathcal{A}, kezdő, c, o)$ állapottér-reprezentációval megadva. Ez a reprezentáció egy irányított gráfot határoz meg.

- Az \mathcal{A} állapottér elemei (az állapotok) a gráf csúcsai. Vezessük be az $a \in \mathcal{A}$ állapot által definiált csúcsra az n_a jelölést. Ekkor a gráf csúcsainak halmaza

$$N = \{n_a \mid a \in \mathcal{A}\}. \quad (1.17)$$

- A gráf csúcsai közül kitüntetett szerepet játszanak a kezdőállapotot szemléltető ún. *startcsúcs* (jele: $n_{kezdő}$ vagy S)
- és a célállapotokat szemléltető *terminális csúcsok*. A terminális csúcsok halmaza tehát:

$$T = \{n_c \mid c \in \mathcal{C}\}. \quad (1.18)$$

- Minden $a \in \mathcal{A}$ -t szemléltető csúcsból *irányított élt* húzunk az $a' \in \mathcal{A}$ -t szemléltető csúcsba, ha a -ból közvetlenül elérhető a' , azaz a gráf irányított éleinek halmaza a következő:

$$E = \{(n_a, n_{a'}) \mid a, a' \in \mathcal{A} \text{ és } a \Rightarrow a'\}. \quad (1.19)$$

Az

$$(N, s, T, E) \quad (1.20)$$

irányított gráfot a P probléma $(\mathcal{A}, kezdő, c, o)$ állapottér-reprezentációjához tartozó *állapottérgráfnak* vagy *reprezentációs gráfnak* nevezzük.

Lemma:

Legyen (N, s, T, E) a P probléma $(\mathcal{A}, kezdő, c, o)$ állapotér-reprezentációjához tartozó állapotérgráfja. Pontosabban akkor vezet az állapotérgráf n_a csúcsából az ettől különböző $n_{a'}$ csúcsba irányított út, ha az n_a állapotból az $n_{a'}$ állapot elérhető.

Bizonyítás:

1. Tegyük fel, hogy $a \Rightarrow^* a'$ ($a \neq a'$). Ez azt jelenti, hogy van olyan $a_1, \dots, a_k \in \mathcal{A}$ ($k \geq 2$) (véges) állapotsorozat, hogy

$$E = \{(n_a, n_{a'}) \mid a, a' \in \mathcal{A} \text{ és } a \Rightarrow a'\}. \quad (1.21)$$

Tehát a reprezentációs gráfban minden $1 \leq i \leq k-1$ -re az n_{a_i} csúcsból irányított él vezet az $n_{a_{i+1}}$ csúcsba, azaz

$$(n_{a_i}, n_{a_{i+1}}) \in E. \quad (1.22)$$

Ez viszont azt jelenti, hogy $n_{a_1} = n_a$ csúcsból irányított út vezet az $n_{a_k} = n_{a'}$ csúcsba.

2. Tegyük fel azt, hogy az állapotérgráf n_a csúcsából az ettől különböző $n_{a'}$ csúcsba irányított út vezet. Ez azt jelenti, hogy van olyan $(n_{a_1}, n_{a_2}), (n_{a_2}, n_{a_3}), \dots, (n_{a_{k-1}}, n_{a_k}) \in E$ ($k \geq 2$) irányított élsorozat az állapotérgráfban, hogy $n_{a_1} = n_a$ és $n_{a_k} = n_{a'}$. Ekkor

- egyrészt $a_1 = a$ és $a_k = a'$,
- továbbá $(n_{a_{i-1}}, n_{a_i}) \in E$ csak akkor, ha $a_{i-1} \Rightarrow a_i$.

Ez azt jelenti, hogy $a \Rightarrow^* a'$, tehát az n_a állapotból az $n_{a'}$ állapot elérhető.

Tétel:

Legyen (N, s, T, E) a P probléma $(\mathcal{A}, kezdő, c, o)$ állapotér-reprezentációjához tartozó állapotérgráfja. Pontosabban akkor megoldható P , ha van az állapotérgráfban a startcsúcsból valamelyik terminális csúcsba vezető irányított út.

Bizonyítás:

1. Egyrészt ha P megoldható, van olyan $c \in \mathcal{C}$ célállapot, hogy $kezdő \Rightarrow^* c$. De ekkor az állapotérgráfban irányított út vezet az $n_{kezdő}$ startcsúcsból az $n_c \in T$ terminális csúcsba.
2. Másrészt ha van az állapotérgráfban az $n_{kezdő}$ startcsúcsból valamely terminális csúcsba vezető irányított út, a $kezdő$ állapotból elérhető ezen terminális csúcs által szemléltetett állapot. De a terminális csúcsok célállapotokat szemléltetnek. Tehát P megoldható.

A P probléma $(\mathcal{A}, kezdő, c, o)$ állapotér-re $(n_a, n_{a'})$ ában vegyük figyelembe az $a \Rightarrow_o a'$ operátorok költség (o, a) geit. Rendeljünk ekkor mindkét $(n_a, n_{a'})$ költséget: ha $c_{aa'} < c_{a'a}$, akkor legyen ezen él költsége (jelölve: $c_{aa'}$). Egy $(n_a, n_{a'})$ költséget: ha $c_{aa'} \geq c_{a'a}$, akkor legyen ezen él költsége (jelölve: $c_{a'a}$). Egy $(n_a, n_{a'})$ költséget: ha $c_{aa'} = c_{a'a}$, akkor legyen ezen él költsége (jelölve: $c_{aa'}$).

$$(n_{a_1}, n_{a_2}), (n_{a_2}, n_{a_3}), \dots, (n_{a_{k-1}}, n_{a_k}) \in E \quad (k \geq 2) \quad (1.23)$$

irányított út költsége a benne szereplő élek költségösszege

$$\sum_{i=1}^{k-1} költség(n_{a_i}, n_{a_{i+1}}). \quad (1.24)$$

Ha minden él költsége egységnyi, az irányított út költsége éppen az út éleinek a száma.

Egy állapotér-reprezentált probléma megoldásának sikerét jelentősen befolyásolja a reprezentációs gráf bonyolultsága:

- a csúcsok száma,
- az egy csúcsból kiinduló élek száma,
- a hurkok és körök száma és hossza.

Ezért célszerű minden lehetséges egyszerűsítést végrehajtani. Lehetséges egyszerűsítések:

- a csúcsok számának csökkentése — ügyes reprezentációval az állapotér kisebb méretű lehet;
- az egy csúcsból kiinduló élek számának csökkentése — az operátorok értelmezési tartományának alkalmas megválasztásával érhető el;
- a reprezentációs gráf fává alakítása — a hurkokat, illetve köröket „kiegyenesítjük”

Chapter 2. Megoldást kereső rendszerek

Az állapottérgráfban keressük a megoldást: a start csúcsból valamely terminális csúcsba vezető utat. Az állapottérgráfot implicit módon — az állapottér-reprezentáció megadásával — adjuk meg a megoldást kereső rendszereknek. Ezek a keresés során addig és úgy építik a gráfot, amíg megoldást nem találnak, vagy amíg valamilyen ok miatt kudarcot nem vallanak a kereséssel.

A megoldást kereső rendszerek *felépítése*:

- *Az adatbázis* az állapottérgráfnak a keresés során előállított része, amit kiegészíthetünk a hatékony kereséshez szükséges bizonyos információkkal.
- *A műveletek* módosítják az adatbázist, azaz az állapottérgráf adatbázisbeli részéből az állapottérgráf egy újabb (további) részét állítják elő. A rendszer alkalmazhat
 - állapottér-reprezentációs operátorokból származtatott műveleteket,
 - „technikai” műveleteket (pl. visszalépést).
A műveleteknek is vannak végrehajtási feltételeik.
- *A vezérlő* irányítja a keresést. Megmondja, hogy a megoldáskeresés folyamán az adatbázisra, annak mely részén, mikor, melyik a végrehajtási feltételeknek eleget tevő művelet hajtódjon végre. Figyeli azt is, hogy befejeződhet-e a keresés, azaz
 - megvan-e a probléma megoldása,
 - vagy kiderült, hogy nem megoldható a probléma.

A megoldáskereső vezérlője az alábbi séma szerint működik:

```
1
  procedure Kereső(A, kezdő, C, O)
    adatbázis ← Inicializál(kezdő)
    while Igaz do
5      if Megoldás-Talál(adatbázis) then
        break
      end if
      if Nem-Folytat(adatbázis) then
10       break
      end if
      művelet ← Választ(adatbázis,műveletek)
      adatbázis ← Alkalmaz(adatbázis,művelet)
    end while
    if Megoldás-Talál(adatbázis) then
15     Megoldás-Kiír(adatbázis)
    else
      print „Sikertelen keresés”
    end if
  end procedure
20
```

A megoldást kereső rendszerek különböző szempontok alapján *osztályozhatók*:

- Módosítható-e valahogy egy már alkalmazott művelet hatása?
 - nem: nemmódosítható megoldáskereső
 - igen: módosítható megoldáskereső
 - visszalépéses (*backtracking*) keresők
 - keresőfával keresők

- Használunk-e valamiféle speciális tudást a keresés során?
 - nem: irányítatlan (vak, szisztematikus) megoldáskereső
 - igen: heurisztikus megoldáskereső

„A *heurisztika* (heurisztikus szabály, módszer) olyan ökölszabály, stratégia, trükk, egyszerűsítés, vagy egyéb eszköz, amely drasztikusan korlátozza a megoldás keresését nagyméretű reprezentációs gráfokban.” (Feigenbaum és Feldman)

- Milyen irányú a keresés?
 - *előrehaladó (forward)* vagy adatvezérelt kereső rendszer: a kezdő állapotból kiindulva kerestünk célállapotba vezető utat.
 - *visszafelé haladó (backward)* vagy célvezérelt kereső rendszer: a célállapotból kiindulva — visszafelé haladva — próbáljuk rekonstruálni a kezdőállapotból odavezető utat.
 - *kétirányú (bidirectional)* kereső rendszer: mindkét irányból elindul, s valahol találkozik

A megoldáskereső rendszerek *értékelési szempontjai*:

- *Teljesség (completeness)*: A rendszer minden olyan esetben megtalálja-e a megoldást, amennyiben az létezik?
- *Optimalitás (optimality)*: Több megoldás létezése esetén a rendszer az optimális megoldást találja-e meg?
- *Időigény (time complexity)*: Mennyi ideig tart egy megoldás megtalálása?
- *Tárigény (space complexity)*: Mekkora tároló területre van szükség a megoldás megtalálásához?

1. Nemmodosítható megoldáskereső rendszerek

A MI módszereit nem használó, ún. hagyományos feladatmegoldási módoknál alkalmazzák. A MI problémák megoldása során nem tudjuk, hogy a reprezentációs gráf megfelelő — a megoldást is tartalmazó — részét építjük-e, ezért ritkán alkalmazunk nemmodosítható keresést az MI területén.

Legyen $p = (A, \text{kezdő}, C, O)$. Egy nemmodosítható megoldáskereső rendszer

- *adatbázisa* az állapottérgráf egyetlen csúcsa, az ún. aktuális csúcs;
- *műveletei* az állapottér-reprezentációs operátorok;
- *vezérlője*:

```

1  procedure Nemmodosithato-Kereso(A, kezdő, C, O)
    aktuális ← kezdő
    while Igaz do
5     if aktuális ∈ C then
        break
    end if
    O' ← {o | o ∈ O ∧ Előfeltétel(aktuális, o)}
    if not O' = ∅ then
10    operátor ← Választ(O')
        aktuális ← Alkalmaz(aktuális, operátor)
    else
        break
    end if
15  end while
    if aktuális ∈ C then
        print aktuális
    else
        print „Sikertelen keresés”
20  end if
    
```

end procedure

Csak olyan probléma megoldásánál alkalmazhatjuk, ahol egy a célfeltételeknek eleget tevő állapotot kell előállítani!

Ugyanazon probléma megoldásának keresése esetén a választás módjában lehet lényeges eltérés. Választhatunk:

- irányítatlanul, szisztematikusan
 - előre rögzített operátorsorrend alapján
 - véletlenszerűen: *próba-hiba módszer*
- heurisztikusan: *hegymászó módszer*

Becsüljük meg a $h : \mathcal{A} \rightarrow \mathbb{R}^+$ ún. *heurisztikus függvény*nel, hogy az egyes állapotokból legkevesebb hány operátor alkalmazásával érhetünk célállapotba:

$$h(a) = \begin{cases} 0, & \text{ha } a \in C \\ \infty, & \text{ha } \neg \exists c (c \in C \wedge a =^* c), \end{cases} \quad (2.1)$$

egyébként $h(a) > 0$.

Ha az a állapot az aktuális, becsüljük meg h segítségével, hogy a milyen „távol” van a hozzá legközelebbi céltól: $h(a)$.

Legyen

$$O' = \{o \mid o \in O \wedge o\text{-alkalmazásának-előfeltétele}(a) \wedge h(o(a)) \leq h(a)\}. \quad (2.2)$$

Azt az O' -beli *operátor* t fogjuk alkalmazni a -ra, amelyik a becslésünk szerint a legközelebbi viz valamelyik terminálishoz:

$$h(\text{operátor}(a)) = \min\{h(o(a)) \mid o \in O'\} \quad (2.3)$$

```

1  procedure Hegymászó-Algoritmus(A, kezdő, C, O )
    aktuális ← kezdő
    while Igaz do
5     if aktuális ∈ C then
        break
    end if
    O' ← {o | o ∈ O ∧ Előfeltétel(aktuális, o) ∧ h(Alkalmaz(aktuális, o)) ≤
h(aktuális)}
    if not O' = ∅ then
10     operátor ← Választ({o | o ∈ O' ∧ ∀o'(o' ∈ O' ⊃ h(Alkalmaz(aktuális, o)) ≤
h(Alkalmaz(aktuális, o'))}))
        aktuális ← Alkalmaz(aktuális, operátor)
    else
        break
    end if
15 end while
    if aktuális ∈ C then
        print aktuális
    else
        print „Sikertelen keresés”
20 end if
```

end procedure

A nemmódosítható megoldáskereső rendszerek értékelése:

- *Teljesség*: Nem teljeselek.
- *Próba-hiba módszer*: Ha olyan kört nem tartalmazó véges állapotérgráfokban keresünk, melyekben minden csúcsból vezet valamelyik terminális csúcsba irányított út, akkor előállít egy célállapotot.
- A hegymászó módszer esetén a heurisztika pontosságától függ, hogy a megoldást megtaláljuk-e vagy sem.
- *Tárigény*: Rendkívül kis adatbázissal dolgozik.

2. Módosítható megoldáskereső rendszerek

2.1. A visszalépéses megoldáskereső algoritmus

Legyen $p = \langle \mathcal{A}, \text{kezdő}, c, o \rangle$. Az alap visszalépéses megoldáskereső

- *adatbázisa* egy a startcsúcsból induló, az ún. aktuális csúcsba vezető utat, az aktuális utat tartalmazza, az út csúcsait és a csúccsal kapcsolatban lévő éleket nyilvántartó *csomópontokból* épül fel. Egy csomópont az alábbi információkat tartalmazza:
 - egy $a \in \mathcal{A}$ állapotot;
 - arra a csomópontra mutatót, mely a szülő állapotot (azt az állapotot, melyre operátort alkalmazva előállt a) tartalmazza;
 - azt az operátort, melyet a szülő állapotra alkalmazva előállt a ;
 - a -ra a keresés során már alkalmazott (vagy még alkalmazható) operátorok halmazát.
- *műveletei*
 - *az operátorokból származtatott műveletek*: egy O operátorra épülő művelet
 - alkalmazási előfeltétele: az aktuális csomópont állapotára alkalmazható O , de a keresés során erre az állapotra (ezen az úton) még nem alkalmaztuk.
 - hatása: az aktuális csomópont állapotára alkalmazzuk az operátort, az előálló állapotból új aktuális csomópontot készítünk az adatbázisban
 - *a visszalépés*
 - alkalmazási előfeltétele: van (aktuális) csomópont az aktuális úton.
 - hatása:
- *vezérlője* eldönti, hogy az adatbázisra mikor melyik műveletet kell végrehajtani, ha még nem teljesülnek a megállási feltételek.

```

1
procedure Alap-Backtrack-1(A, kezdő, C,O)
  Állapot[aktuális-csomópont] ← kezdő
  Szülő[aktuális-csomópont] ← Nil
5 Operátor[aktuális-csomópont] ← *
  Kipróbált[aktuális-csomópont] ← ∅
  while Igaz do

```

```

    if aktuális-csomópont = Nil then
        break
10    end if
    if Állapot[aktuális-csomópont] ∈ C then
        break
    end if
    O' ← {o | o ∈ O ∧ Előfeltétel(Állapot[aktuális-csomópont], o) ∧ o ∉
Kipróbált[aktuális-csomópont]}
15    if not O' = ∅ then
        operátor ← Választ(O')
        Kipróbált[aktuális-csomópont] ← Kipróbált[aktuális-csomópont] ∪ {operátor}
        Állapot[új] ← Alkalmaz(Állapot[aktuális-csomópont], operátor)
        Szülő[új] ← aktuális-csomópont
20    Operátor[új] ← operátor
        Kipróbált[új] ← ∅
        aktuális-csomópont ← új
    else
        aktuális-csomópont ← Szülő[aktuális-csomópont]
25    end if
    end while
    if not aktuális-csomópont = Nil then
        Megoldás-Kiír(aktuális-csomópont )
    else
30    print „Nincs megoldás”
    end if
end procedure

1
procedure Alap-Backtrack-2(A, kezdő, C,O)
    Állapot[aktuális-csomópont] ← kezdő
    Szülő[aktuális-csomópont] ← Nil
5    Operátor[aktuális-csomópont] ← *
    Alkalmazható[aktuális-csomópont] ← {o | o ∈ O ∧ Előfeltétel(Állapot[aktuális-
csomópont], o)}
    while Igaz do
        if aktuális-csomópont = Nil then
            break
10        end if
        if Állapot[aktuális-csomópont] ∈ C then
            break
        end if
        if not Alkalmazható[aktuális-csomópont] = ∅ then
15        operátor ← Választ(Alkalmazható[aktuális-csomópont])
            Alkalmazható[aktuális-csomópont] ← Alkalmazható[aktuális-csomópont] \
{operátor}
            Állapot[új] ← Alkalmaz(Állapot[aktuális-csomópont], operátor)
            Szülő[új] ← aktuális-csomópont
            Operátor[új] ← operátor
20        Alkalmazható[új] ← {o | o ∈ O ∧ Előfeltétel(Állapot[új], o)}
            aktuális-csomópont ← új
        else
            aktuális-csomópont ← Szülő[aktuális-csomópont]
        end if
    end while
25    if not aktuális-csomópont = Nil then
        Megoldás-Kiír(aktuális-csomópont )
    else
        print „Nincs megoldás”
30    end if
end procedure

```

Ugyanazon probléma megoldásának keresése esetén a választás módjában lehet lényeges eltérés. Választhatunk:

- irányítatlanul, szisztematikusan
- előre rögzített operátorsorrend alapján
- véletlenszerűen

- heurisztikusan: Becsüljük meg a $h : \mathcal{A} \rightarrow R^+$ heurisztikával, hogy az egyes csúcsok milyen távol vannak a hozzájuk legközelebbi terminális csúcstól. Legyen

$$o' = \{o \mid o \in O \wedge o\text{-alkalmazásának-előfeltétele}(a)\}. \quad (2.4)$$

Azt az o' -beli *operátor* t fogjuk alkalmazni a -ra, amelyik a becslésünk szerint a legközelebbi vizs valamelyik terminálishoz:

$$h(\text{operátor}(a)) = \min\{h(o(a)) \mid o \in o'\}. \quad (2.5)$$

Az alap visszalépéses megoldáskereső értékelése

- *Teljesség*: Ha a reprezentációs gráf köröket nem tartalmazó véges gráf, akkor az alap visszalépéses megoldáskereső véges sok keresőlépés megtétele után befejezi a keresést,
 - ha van megoldás, előállít egy lehetséges megoldást,
 - ha nincs megoldás, azt felismeri.
- *Tárigény*: Kis méretű az adatbázis.

2.2. Visszalépéses megoldáskereső köröket is tartalmazó gráfokban

- *Visszalépéses megoldáskereső körfigyeléssel*: Ha van megoldás, akkor van körmentes megoldás is. A vezérlő a visszalépést választja, ha az aktuális csúcs szerepelt már az aktuális úton.
- *Visszalépéses megoldáskereső úthosszkorláttal*: Úthosszkorlátot vezetünk be, mely megakadályozza, hogy a köröket „végtelen sokszor” járjuk be. A vezérlő a visszalépést választja, ha az aktuális út hossza eléri, vagy meghaladja az úthosszkorlátot.

```

1  procedure Körfigyeléses-Backtrack(A, kezdő, C, O)
    Állapot[aktuális-csomópont] ← kezdő
    Szülő[aktuális-csomópont] ← Nil
5   Operátor[aktuális-csomópont] ← *
    Alkalmazható[aktuális-csomópont] ← {o | o ∈ O ∧ Előfeltétel(Állapot[aktuális-
csomópont], o)}
    while Igaz do
        if aktuális-csomópont = Nil then
            break
10        end if
        if Állapot[aktuális-csomópont] ∈ C then
            break
        end if
        if VoltMár(Állapot[aktuális-csomópont]) then
15            aktuális-csomópont ← Szülő[aktuális-csomópont]
        end if
        if not Alkalmazható[aktuális-csomópont] = ∅ then
            operátor ← Választ(Alkalmazható[aktuális-csomópont])
            Alkalmazható[aktuális-csomópont] ← Alkalmazható[aktuális-csomópont] \
{operátor}
20            Állapot[új] ← Alkalmaz(Állapot[aktuális-csomópont], operátor)
            Szülő[új] ← aktuális-csomópont
            Operátor[új] ← operátor
            Alkalmazható[új] ← {o | o ∈ O ∧ Előfeltétel(Állapot[új], o)}
            aktuális-csomópont ← új
25        else
            aktuális-csomópont ← Szülő[aktuális-csomópont]
        end if
    end while
    if not aktuális-csomópont = Nil then
30        Megoldás-Kiír(aktuális-csomópont)

```

```

else
  print „Nincs megoldás”
end if
end procedure
35
1
procedure Úthosszkorlátos-Backtrack(A, kezdő, C,O, korlát )
  Állapot[aktuális-csomópont] ← kezdő
  Szülő[aktuális-csomópont] ← Nil
  5 Mélység[aktuális-csomópont] ← 0
  Operátor[aktuális-csomópont] ← *
  Alkalmazható[aktuális-csomópont] ← {o | o ∈ O ∧ Előfeltétel(Állapot[aktuális-
csomópont], o)}
  while Igaz do
    if aktuális-csomópont = Nil then
10      break
    end if
    if Állapot[aktuális-csomópont] ∈ C then
      break
    end if
15    if Mélység[aktuális-csomópont] = korlát then
      aktuális-csomópont ← Szülő[aktuális-csomópont]
    end if
    if not Alkalmazható[aktuális-csomópont] = ∅ then
      operátor ← Választ(Alkalmazható[aktuális-csomópont])
20    Alkalmazható[aktuális-csomópont] ← Alkalmazható[aktuális-csomópont] \
{operátor}
    Állapot[új] ← Alkalmaz(Állapot[aktuális-csomópont], operátor)
    Szülő[új] ← aktuális-csomópont
    Mélység[új] ← Mélység[aktuális-csomópont] + 1
    Operátor[új] ← operátor
25    Alkalmazható[új] ← {o | o ∈ O ∧ Előfeltétel(Állapot[új], o)}
    aktuális-csomópont ← új
  else
    aktuális-csomópont ← Szülő[aktuális-csomópont]
  end if
30  end while
  if not aktuális-csomópont = Nil then
    Megoldás-Kiír(aktuális-csomópont )
  else
    print „Sikertelen keresés”
35  end if
end procedure

```

Értékelés

- *Teljesség:*
 - A körfigyeléses backtrack ha a reprezentációs gráf véges, akkor véges sok keresőlépés megtétele után befejezi a keresést, és
 - ha van megoldás, előállít egy körmentes megoldást,
 - ha nincs megoldás, azt felismeri.
 - Az úthosszkorlátos backtrack tetszőleges reprezentációs gráf esetén véges sok keresőlépés megtétele után befejezi a keresést,
 - ha van az úthosszkorlátnál nem hosszabb megoldás, előállít egy ilyen megoldást.
 - Ha keresés nem egy megoldás megtalálásával ér véget, akkor az úthosszkorlátnál csak hosszabb megoldás lehet a reprezentációs gráfban. (Vagy nincs megoldás, vagy az úthosszkorlát túl kicsi.)
- *Időigény:*
 - A körfigyeléses backtrack időigényes (főleg hosszú körök esetén).

- *Tárigény:*

- Az úthosszkorlátos backtrack adatbázisa legfeljebb úthosszkorlátnyi elemet tartalmaz. A megtalált megoldás nem feltétlen körmentes.

2.3. Ág és korlát algoritmus

A backtrack alkalmas optimális (legrövidebb) megoldás keresésére is.

- Egy induló úthosszkorlátnál nem hosszabb megoldást keresünk.
- Ha találunk ilyen, tároljuk, majd ennek hosszát új úthosszkorlátnak választva folytatjuk a keresést.

Úthosszkorlát helyett költségkorlátot, csomópont mélysége helyett pedig az addig tartó út költségét is használhatjuk az algoritmusban. Ekkor legkisebb költségű megoldás előállítására lehet a cél.

```

1
procedure Ág-és-Korlát(A, kezdő, C,O, korlát )
    talált ← Hamis
    Állapot[aktuális-csomópont] ← kezdő
5    Szülő[aktuális-csomópont] ← Nil
    Mélység[aktuális-csomópont] ← 0
    Operátor[aktuális-csomópont] ← *
    Alkalmazható[aktuális-csomópont] ← {o | o ∈ O ∧ Előfeltétel(Állapot[aktuális-
csomópont], o)}
    while Igaz do
10     if aktuális-csomópont = Nil then
        break
    end if
    if Állapot[aktuális-csomópont] ∈ C then
        talált ← Igaz
15     Megoldás-Feljegyez(aktuális-csomópont )
        korlát ← Mélység[aktuális-csomópont]
    end if
    if Mélység[aktuális-csomópont] = korlát then
        aktuális-csomópont ← Szülő[aktuális-csomópont]
20     end if
    if not Alkalmazható[aktuális-csomópont] = ∅ then
        operátor ← Választ(Alkalmazható[aktuális-csomópont])
        Alkalmazható[aktuális-csomópont] ← Alkalmazható[aktuális-csomópont] \
{operátor}
        Állapot[új] ← Alkalmaz(Állapot[aktuális-csomópont], operátor)
25     Szülő[új] ← aktuális-csomópont
        Mélység[új] ← Mélység[aktuális-csomópont] + 1
        Operátor[új] ← operátor
        Alkalmazható[új] ← {o | o ∈ O ∧ Előfeltétel(Állapot[új], o)}
        aktuális-csomópont ← új
30     else
        aktuális-csomópont ← Szülő[aktuális-csomópont]
    end if
    end while
    if talált then
35     Megoldás-Kiír
    else
        print „Sikertelen keresés”
    end if
    end procedure
40

```

Értékelés

- *Optimalitás:* Az ág és korlát algoritmus tetszőleges reprezentációs gráf esetén véges sok keresőlépés megtétele után befejezi a keresést,
 - ha van az induló úthosszkorlátnál nem hosszabb megoldás, a legrövidebb megoldást állítja elő,
 - ha a keresés nem megoldás megtalálásával ér véget, akkor az induló úthosszkorlátnál csak hosszabb megoldás lehet a reprezentációs gráfban. (Vagy nincs megoldás, vagy az induló úthosszkorlát túl kicsi.)

- *Tárigény:*
 - Az ág és korlát adatbázisa legfeljebb kétszer az induló úthosszkorlátnyi csomópontot tartalmaz.

2.4. Keresőfával keresők

Legyen $p = (A, \text{kezdő}, c, o)$. A keresőfával kereső rendszerek

- *adatbázisa* a reprezentációs gráf már bejárt részét feszítő fa, az ún. *keresőfa*. A keresőfa csúcsait és a velük kapcsolatban lévő éleket (explicit vagy implicit módon) nyilvántartó csomópontok az alábbi információkat tartalmazzák:
 - egy $a \in A$ állapotot;
 - arra a csomópontra mutatót, mely a szülő állapotot tartalmazza;
 - azt az operátort, melyet a szülő állapotra alkalmazva előállt a ;
- *státusz:*
 - *zárt*, ha a utódait tartalmazó csomópontokat a keresés során már előállítottuk;
 - *nyílt*, egyébként.
- *művelete a kiterjesztés:* a keresőfát annak egy nyílt csúcsán (egy nyílt csomóponton) keresztül kibővíti.
 - alkalmazási előfeltétele, hogy a keresőfában legyen nyílt csomópont.
 - hatása:
 - alkalmazzuk az összes alkalmazható operátort a nyílt csomópont állapotára,
 - az előálló állapotok közül
 - amelyek még nem szerepeltek a keresőfa egyetlen csomópontjában sem, azokból a keresőfába felfűzött új nyílt csomópont készül,
 - amelyek már szerepeltek a keresőfa valamely csomópontjában, azok sorsa keresőfüggő.
 - a kiterjesztett csomópont zárttá válik.
- *vezérlő* megmondja, hogy melyik nyílt csomópont legyen a következő lépésben kiterjesztve.
 - Ha a kiválasztott nyílt csomópont állapota teljesíti a célfeltételeket, a keresőfában a szülőre mutatók mentén elő tudunk állítani egy megoldást is.
 - Nincs megoldás, ha egyetlenegy nyílt csomópont sincs a keresőfában.

```

1  procedure Keresőfával-Kereső(A, kezdő, C, O)
    Állapot[csomópont] ← kezdő
    Szülő[csomópont] ← Nil
5  Operátor[csomópont] ← *
    nyíltak ← {csomópont}; zártak ← ∅
    while Igaz do
        if nyíltak = ∅ then
            break
10        end if
        csomópont ← Választ(nyíltak)
        if Állapot[csomópont] ∈ C then
            break
        end if

```

```

15   Kiterjeszt(csomópont, nyíltak, zártak)
      end while
      if not nyíltak = ∅ then
        Megoldás=Kiír(csomópont )
      else
20   print „Nincs megoldás”
      end if
end procedure

```

Ugyanazon probléma megoldásának keresése esetén lényeges eltérés lehet

1. a választás módjában. A vezérlő választhat

- irányítatlanul, szisztematikusan
- a csomópontok keresőgráfbeli mélysége alapján: szélességi és mélységi keresők;
- a csomópontok állapotait előállító költség alapján: optimális kereső;
- heurisztikusan:
 - best-first algoritmus;
 - A algoritmusok.

2. abban, hogy mi történik, ha a keresőgráf egy csúcsához a keresés során újabb odavezető utat tár fel a vezérlő.

3. a célfeltételek vizsgálatának időpontjában.

2.5. Szélességi és mélységi keresők

1. Egy csomópont előállításakor követjük, hogy a csomópontban nyilvántartott csúcs a keresőfában milyen „mélyen” van:

$$\text{mélység}(m) = \begin{cases} 0 & \text{ha } m = s \\ \text{mélység}(n) + 1 & (n, m) \in E \end{cases} \quad (2.6)$$

Kiterjesztésre

- a szélességi kereső vezérlője a *legkisebb mélységi számú*
- a mélységi kereső vezérlője a *legnagyobb mélységi számú*

nyílt csomópontot választja ki.

2. Ha a vezérlő a keresőgráf egy csúcsához a keresés során *újabb odavezető utat* tár fel, ezt *nem tárolja*, „elfelejti”.

3. A célfeltételek teljesítésének vizsgálatát előre hozhatjuk az állapot előállítását követő időpontra.

```

1   procedure Kiterjeszt(A, kezdő, C,O, csomópont, nyíltak, zártak)
      for all o ∈ O do
        if Előfeltétel(Állapot[csomópont ], o) then
5   állapot ← Alkalmaz(Állapot[csomópont],o)
        ny ← Keres(nyíltak, állapot)
        z ← Keres(zártak, állapot)
        if ny = Nil and z = Nil then
10  Állapot[új-csomópont] ← állapot
        Szülő[új-csomópont] ← csomópont
        Operátor[új-csomópont] ← o
        Mélység[új-csomópont] ← Mélység[csomópont] + 1
        nyíltak ← nyíltak U {új-csomópont}

```

```

    end if
15  end if
    end for
    nyíltak ← nyíltak \ {csomópont}
    zártak ← zártak ∪ {csomópont}
end procedure
20

1
procedure Szélességi-Kereső(A, kezdő, C, O)
    Állapot[új-csomópont] ← kezdő
    Szülő[új-csomópont] ← Nil
5  Operátor[új-csomópont] ← *
    Mélység[új-csomópont] ← 0
    nyíltak ← {új-csomópont}
    zártak ← ∅
    while Igaz do
10  if nyíltak = ∅ then
        break
    end if
    csomópont ← Választ({cs | cs ∈ nyíltak ∧ ∀cs'(cs' ∈ nyíltak ⊃ Mélység[cs] ≤
Mélység[cs']})
    if Állapot[csomópont] ∈ C then
15  break
    end if
    Kiterjeszt(A, kezdő, C, O, csomópont, nyíltak, zártak)
    end while
    if not nyíltak = ∅ then
20  Megoldás-Kiír(csomópont)
    else
        print „Nincs megoldás”
    end if
end procedure
25

1
procedure Mélységi-Kereső(A, kezdő, C, O)
    Állapot[új-csomópont] ← kezdő
    Szülő[új-csomópont] ← Nil
5  Operátor[új-csomópont] ← *
    Mélység[új-csomópont] ← 0
    nyíltak ← {új-csomópont}
    zártak ← ∅
    while Igaz do
10  if nyíltak = ∅ then
        break
    end if
    csomópont ← Választ({cs | cs ∈ nyíltak ∧ ∀cs'(cs' ∈ nyíltak ⊃ Mélység[cs] ≥
Mélység[cs']})
    if Állapot[csomópont] ∈ C then
15  break
    end if
    Kiterjeszt(A, kezdő, C, O, csomópont, nyíltak, zártak)
    end while
    if not nyíltak = ∅ then
20  Megoldás-Kiír(csomópont)
    else
        print „Nincs megoldás”
    end if
end procedure
25

```

A nyílt csomópontokat gyakran

- a szélességi kereső *sorban*,
- a mélységi kereső *veremben*

tartja nyilván, melyből mélységi szám szerint ezek épp megfelelő sorrendben kerülnek ki.

2.5.1. A szélességi kereső értékelése

- *Teljesség*: A vezérlő,
 - ha van megoldás, tetszőleges reprezentációs gráfban véges sok keresőlépés után előállít egy megoldást,
 - ha nincs az adott reprezentációban megoldás, akkor (véges gráf esetén) azt a nyílt csomópontok elfogyásával felismeri.
- *Optimalitás*: Ha van megoldás, tetszőleges reprezentációs gráfban a vezérlő a legrövidebb megoldást állítja elő.
- *Tárigény*: Nagy az adatbázis. Legyen a reprezentációs fa minden csúcsának d gyermeke, és l hosszúságú a legrövidebb megoldás. Ekkor a keresőgráf csomópontjainak száma a keresés végére (a legrosszabb esetben):

$$1 + d + d^2 + d^3 + \dots + d^{l+1} - d \approx O(d^{l+1}). \quad (2.7)$$

2.5.2. A mélységi kereső értékelése

- *Teljesség*: A vezérlő véges reprezentációs gráfban,
 - ha van megoldás, véges sok keresőlépés után előállít egy megoldást,
 - ha nincs a problémának az adott reprezentációban megoldása, akkor azt a nyílt csomópontok elfogyásával felismeri.

2.6. Optimális kereső

1. Minden csomópontnál számon tartjuk az odavezető keresőfabeli út költségét:

$$\text{útköltség}(m) = \begin{cases} 0 & \text{ha } m = s \\ \text{útköltség}(n) + \text{költség}(n, m) & (n, m) \in E \end{cases} \quad (2.8)$$

$\text{útköltség}^*(n)$ jelölje a startcsúcsból n -be jutás optimális költségét. Ekkor

$$\text{útköltség}(n) \geq \text{útköltség}^*(n), \text{ mindenn } n \in N\text{-re.} \quad (2.9)$$

Kiterjesztésre az optimális kereső vezérlője a *legkisebb költségű* nyílt csomópontot választja ki.

2. Ha a vezérlő a keresőgráf egy csúcsához a keresés során *újabb odavezető utat* tár fel, azaz az n csomópont kiterjesztéskor előállt állapot szerepel már a keresőgráf m csomópontjában

- nyíltként: és ha az

$$\text{útköltség}(n) + \text{költség}(n, m) < \text{útköltség}(m), \quad (2.10)$$

akkor az új kisebb költségű utat tároljuk, a régit „elfelejtjük”.

- zártként: az m csomópontot már n kiterjesztése előtt kiterjesztette a vezérlő, azaz $\text{útköltség}(m) \leq \text{útköltség}(n)$ volt, így

$$\text{útköltség}(n) + \text{költség}(n, m) > \text{útköltség}(m). \quad (2.11)$$

Az m -hez feltárt új út költségesebb.

3. A célfeltételek vizsgálatát *nem hozhatjuk előre.*

```

1
procedure Kiterjeszt(A, kezdő, C, O, költség, csomópont, nyíltak, zártak)
  for all o ∈ O do
    if Előfeltétel(Állapot[csomópont ], o) then
5      állapot ← Alkalmaz(Állapot[csomópont],o)
      ny ← Keres(nyíltak, állapot)
      z ← Keres(zártak, állapot)
      if ny = Nil and z = Nil then
        Állapot[új-csomópont] ← állapot
10      Szülő[új-csomópont] ← csomópont
        Operátor[új-csomópont] ← o
        Útköltség[új-csomópont] ← Útköltség[csomópont] +
költség(o,Állapot[csomópont])
        nyíltak ← nyíltak U {új-csomópont}
      else if not ny = Nil then
15      új-út-költség ← Útköltség[csomópont] + költség(o,Állapot[csomópont])
      if új-út-költség ≤ Útköltség[ny] then
        Szülő[ny] ← csomópont
        Operátor[ny] ← o
        Útköltség[ny] ← új-út-költség
20      end if
      end if
    end for
    nyíltak ← nyíltak \ {csomópont}
25    zártak ← zártak U {csomópont}
  end procedure

1
procedure Optimális-Kereső(A, kezdő, C, O, költség)
  Állapot[új-csomópont] ← kezdő
  Szülő[új-csomópont] ← Nil
5  Operátor[új-csomópont] ← *
  Útköltség[új-csomópont] ← 0
  nyíltak ← {új-csomópont}
  zártak ← ∅
  while Igaz do
10    if nyíltak = ∅ then
      break
    end if
    csomópont ← Választ({cs | cs ∈ nyíltak ∧ ∀cs'(cs' ∈ nyíltak ⊃ Útköltség[cs] ≤
Útköltség[cs']}))
    if Állapot[csomópont] ∈ C then
15      break
    end if
    Kiterjeszt(A, kezdő, C, O, költség, csomópont, nyíltak, zártak)
  end while
  if not nyíltak = ∅ then
20    Megoldás-Kiír(csomópont)
  else
    print „Nincs megoldás”
  end if
end procedure
25

```

2.6.1. Az optimális kereső értékelése

- *Teljesség:* A vezérlő,
 - ha van megoldás, tetszőleges reprezentációs gráfban véges sok keresőlépés után előállít egy megoldást,
 - ha nincs a problémának az adott reprezentációban megoldása, akkor véges gráf esetén azt a nyílt csomópontok elfogyásával felismeri.
- *Optimalitás:* Ha van megoldás, tetszőleges reprezentációs gráfban a vezérlő véges sok keresőlépés után az optimális megoldást állítja elő.

2.7. Best-first algoritmus

1. A keresőfa minden csomópontjánál nyilvántartjuk, hogy a csomópontbeli állapot a h heurisztikus függvény szerint milyen „távol” van a hozzá legközelebbi céltól. Kiterjesztésre a best-first vezérlője a *legkisebb heurisztikájú* nyílt csomópontot választja ki (legjobb irányban haladó keresés).
2. Ha a vezérlő a keresőgráf egy csúcsához a keresés során *újabb odavezető utat* tár fel, ezt *nem tárolja*, „elfelejti”.
3. A célfeltételek vizsgálatát *előre hozhatjuk*.

```

1
  procedure Kiterjeszt(A, kezdő, C, O, h, csomópont, nyíltak, zártak)
    for all o ∈ O do
      if Előfeltétel(Állapot[csomópont], o) then
        állapot ← Alkalmaz(Állapot[csomópont], o)
        ny ← Keres(nyíltak, állapot)
        z ← Keres(zártak, állapot)
        if ny = Nil and z = Nil then
          Állapot[új-csomópont] ← állapot
          Szülő[új-csomópont] ← csomópont
          Operátor[új-csomópont] ← o
          Heurisztika[új-csomópont] ← h(állapot)
          nyíltak ← nyíltak ∪ {új-csomópont}
        end if
      end if
    end for
    nyíltak ← nyíltak \ {csomópont}
    zártak ← zártak ∪ {csomópont}
  end procedure
20

1
  procedure Best-First( A, kezdő, C, O, h)
    Állapot[új-csomópont] ← kezdő
    Szülő[új-csomópont] ← Nil
    Operátor[új-csomópont] ← *
    Heurisztika[új-csomópont] ← h(kezdő)
    nyíltak ← {új-csomópont}
    zártak ← ∅
    while Igaz do
      if nyíltak = ∅ then
        break
      end if
      csomópont ← Választ({cs | cs ∈ nyíltak ∧ ∀cs' (cs' ∈ nyíltak ⊃ Heurisztika[cs] ≤
Heurisztika[cs']})
      if Állapot[csomópont] ∈ C then
        break
      end if
      Kiterjeszt(A, kezdő, C, O, h, csomópont, nyíltak, zártak)
    end while
    if not nyíltak = ∅ then
      Megoldás-Kiír(csomópont)
    else
      print „Nincs megoldás”
    end if
  end procedure
25

```

2.7.1. A best-first algoritmus értékelése

- *Teljesség*: A vezérlő tetszőleges véges reprezentációs gráfban,
 - ha van megoldás, véges sok keresőlépés után előállít egy megoldást,
 - ha nincs a problémának az adott reprezentációban megoldása, akkor azt a nyílt csomópontok elfogyásával felismeri.

- *Tárigény*: Perfekt heurisztika esetén, ha a reprezentációs fa minden csúcsának d gyermeke van, és l hosszú a legrövidebb megoldás, a keresőgráf csomópontjainak száma a keresés végére:

$$1 + d + d + \dots + d = O(l \cdot d). \quad (2.12)$$

2.8. Az A algoritmus

1. A keresőgráf minden csomópontjában megbecsüljük a rajta keresztülhaladó megoldás költségét. Ez egyrészt a csomópontig vezető nyilvántartott út költsége, amihez hozzászámítjuk a célig hátralevő út becsült költségét:

$$\text{összköltség}(m) = \text{útköltség}(m) + \text{heurisztika}(m), \quad (2.13)$$

azaz

$$\text{összköltség}(m) = \text{útköltség}(n) + \text{költség}(n, m) + \text{heurisztika}(m), \quad (2.14)$$

ha $(n, m) \in E$. Ha $\text{összköltség}^*(m)$ -mel jelöljük az m csúcson keresztül célba jutás optimális költségét, akkor minden m csúcsra

$$\text{összköltség}^*(m) \approx \text{összköltség}(m) \quad (2.15)$$

Kiterjesztésre az A algoritmus vezérlője a *legkisebb összköltségű* nyílt csomópontot választja ki.

2. Ha a vezérlő a keresőgráf egy csúcsához a keresés során *újabb odavezető utat* tár fel, azaz az n csomópont kiterjesztéskor előállt állapot szerepel már a keresőgráf m csomópontjában, és az

$$\text{útköltség}(n) + \text{költség}(n, m) < \text{útköltség}(m), \quad (2.16)$$

akkor az új kisebb költségű utat tároljuk, a régit „elfelejtjük”.

- Ha m nyílt volt, más teendő nincs.
- Ha m zárt volt, a keresőfa m -ből induló részének csomópontjaiban az *útköltség*-et frissíteni kell, ami problémát okoz:
 - külön eljárást írunk a frissítésre;
 - az A algoritmussal frissítetjük a részgráfot;
 - megelőzzük a probléma kialakulását.

3. A célfeltételek vizsgálatát *nem hozhatjuk előre*.

```

1  procedure Kiterjeszt(A, kezdő, C, O, költség, h, csomópont, nyíltak, zártak)
   for all o ∈ O do
     if Előfeltétel(Állapot[csomópont ], o) then
5      állapot ← Alkalmaz(Állapot[csomópont], o)
       ny ← Keres(nyíltak, állapot)
       z ← Keres(zártak, állapot)
       if ny = Nil and z = Nil then
10        Állapot[új-csomópont] ← állapot
          Szülő[új-csomópont] ← csomópont
          Operátor[új-csomópont] ← o
          Útköltség[új-csomópont] ← Útköltség[csomópont] +
költség(o, Állapot[csomópont])

```

```

    Heurisztika[új-csomópont] ← h(állapot)
    nyíltak ← nyíltak ∪ {új-csomópont}
15   else
        új-út-költség ← Útköltség[csomópont] + költség(o, Állapot[csomópont])
        if not ny = Nil then
            if új-út-költség ≤ Útköltség[ny] then
                Szülő[ny] ← csomópont
                Operátor[ny] ← o
                Útköltség[ny] ← új-út-költség
            end if
        else
            if új-út-költség ≤ Útköltség[z] then
                Szülő[z] ← csomópont
                Operátor[z] ← o
                Útköltség[z] ← új-út-költség
                zártak ← zártak \ {z}
                nyíltak ← nyíltak ∪ {z}
            end if
        end if
    end if
end for
35   nyíltak ← nyíltak \ {csomópont}
    zártak ← zártak ∪ {csomópont}
end procedure

1   procedure A-algoritmus(A, kezdő, C, O, költség, h)
    Állapot[új-csomópont] ← kezdő
    Szülő[új-csomópont] ← Nil
5   Operátor[új-csomópont] ← *
    Útköltség[új-csomópont] ← 0
    Heurisztika[új-csomópont] ← h(kezdő)
    nyíltak ← {új-csomópont}
    zártak ← ∅
10  while Igaz do
        if nyíltak = ∅ then
            break
        end if
        csomópont ← Választ({cs | cs ∈ nyíltak ∧ ∀cs' (cs' ∈ nyíltak ⊃
(Útköltség[cs]+Heurisztika[cs]) ≤ (Útköltség[cs']+Heurisztika[cs'])))}
15     if Állapot[csomópont] ∈ C then
            break
        end if
        Kiterjeszt(A, kezdő, C, O, költség, h, csomópont, nyíltak, zártak)
    end while
20  if not nyíltak = ∅ then
        Megoldás-Kiír(csomópont)
    else
        print „Nincs megoldás”
    end if
25  end procedure

```

2.8.1. Az A algoritmus értékelése

- *Teljesség:* A vezérlő,
 - ha van megoldás, tetszőleges reprezentációs gráfban véges sok keresőlépés után előállít egy megoldást,
 - ha nincs a problémának az adott reprezentációban megoldása, akkor (véges gráf esetén) azt a nyílt csomópontok elfogyásával felismeri.
- *Optimalitás:* Nincs garancia az optimális megoldás előállítására. De ha minden $a \in \mathcal{A}$ esetén

$$h(a) \leq h^*(a), \quad (2.17)$$

ahol $h^*(a)$ az a állapotból célba A^* -ás optimális költsége, akkor az A algoritmus az optimális megoldást állítja elő, ha van megoldás. Ez az *algoritmus*.

Lemma:

Az A algoritmus a működése során egy csomópontot legfeljebb véges sokszor terjeszt ki.

Bizonyítás:

Egy csomópontot csak akkor terjeszthetünk ki, ha nyílt. A nyílt csomópontok közé legfeljebb annyiszor kerülhet, ahányszor egy minden addiginál olcsóbb utat találunk hozzá. Belátjuk, hogy véges sok ilyen út van.

Jelölje δ az élek költségének pozitív alsó korlátját, vagyis minden (n, m) esetén

$$\text{költség}(n, m) \geq \delta > 0. \quad (2.18)$$

Tegyük föl, hogy a p csúcsba először egy $\text{útköltség}(p)$ költségű úton jutunk el. Ez az út legfeljebb $\text{útköltség}(p)/\delta$ hosszú lehet. Az ennél olcsóbb p -be vezető utak $\text{útköltség}(p)/\delta$ -nál biztosan rövidebbek. A $\text{útköltség}(p)/\delta$ korlátnál rövidebb p -be vezető utak száma viszont véges.

Lemma:

Az A algoritmus, hacsak közben nem fejezi be sikeresen a keresést, minden a nyíltak halmazba bekerülő csomópontot véges sok lépés után kiterjeszt.

Bizonyítás:

Legyen $p \in \text{nyíltak}$. Megmutatjuk, hogy p kiválasztása előtt kiterjesztendő (nála kisebb összköltséggel rendelkező) csomópontok száma véges, és egy ilyen csak véges sokszor kerülhet vissza a nyílt csomópontok közé.

- Először belátjuk, hogy egy csomópont összköltsége arányos a csomópont mélységével. Amikor egy n csomópont bekerül a *nyíltak* halmazba, akkor

$$\begin{aligned} \text{összköltség}(n) &= \text{útköltség}(n) + \text{heurisztika}(n) \geq \text{útköltség}(n) \geq \\ &\geq \text{útköltség}^*(n) \geq \text{úthossz}^*(n) \cdot \delta, \end{aligned} \quad (2.19)$$

ahol $\text{úthossz}^*(n)$ az s -ből n -be vezető optimális költségű út hossza.

- Most egy mélységi korlátot adunk meg. Legyen

$$K = \left\lceil \frac{\text{összköltség}(p)}{\delta} \right\rceil + 1. \quad (\text{Nyilván } K \geq \text{összköltség}(p)/\delta). \quad (2.20)$$

- Ennél a korlátnál mélyebben fekvő csomópontokra az összköltség nagyobb, mint $\text{összköltség}(p)$. Ugyanis ha egy k csomópontra $\text{úthossz}^*(k) > K$, akkor

$$\text{összköltség}(k) \geq \text{úthossz}^*(k) \cdot \delta > K \cdot \delta \geq \text{összköltség}(p). \quad (2.21)$$

Tehát $\text{összköltség}(k) > \text{összköltség}(p)$, azaz az $\text{összköltség}(p)$ -nél nem nagyobb összköltséggel rendelkező csomópontok a mélységi korlátnál magasabban helyezkednek el.

- Mivel az egyes csomópontokból kiinduló élek száma fölülről korlátos, így egy adott mélységi korlátnál magasabban fekvő csomópontok száma véges.

Tétel:

Az A algoritmus véges reprezentációs gráfban véges sok lépés után befejezi a keresést.

Bizonyítás:

A korábbi lemma értelmében az A algoritmus a véges sok lehetséges csomópont mindegyikét legfeljebb véges sokszor terjesztheti ki. Ez azt jelenti, hogy véges sok lépésen belül az összes csomópontot végleg ki is terjeszti, ha előbb nem áll le sikeresen a keresés. Az algoritmus tehát

- vagy talál célállapotot tartalmazó csomópontot,
- vagy pedig elfogynak a nyílt csomópontok,

és befejeződik a keresés.

Lemma:

Ha van megoldás, az A algoritmus adatbázisában a nyílt csomópontok között mindig van az optimális úton fekvő csúcs.

Bizonyítás:

Legyen $s = n_0, n_1, \dots, n_k = t$ optimális út.

- 1. kiválasztás előtt: $s \in \text{nyíltak}$.
- k . kiválasztás előtt: indukciós feltevésünk szerint $\exists j (n_j \in \text{nyíltak})$. Legyen i a legkisebb ilyen index.
- $k+1$. kiválasztás előtt:
 - Ha nem n_i -t terjesztjük ki, akkor n_i nyílt marad.
 - Ha n_i -t terjesztjük ki, akkor akár először állítottuk elő, akár már szerepelt a keresőfában: n_{i+1} nyílt lesz.

Tétel:

Tetszőleges reprezentációs gráf esetén, ha van megoldás, az A algoritmus véges sok lépésben megoldással fejezi be a keresést.

Lemma:

Az A^* algoritmus által kiterjesztésre kiválasztott tetszőleges n csomópontra

$$\text{összköltség}(n) \leq \text{összköltség}^*(s). \quad (2.22)$$

Bizonyítás:

$$\overset{\text{összköltség}^*(s)}{\text{összköltség}^*(s)} = \overset{\text{útköltség}^*(s)}{\text{útköltség}^*(s)} + \overset{\text{heurisztika}^*(s)}{\text{heurisztika}^*(s)} < \infty$$
 , egyébként az optimális megoldás költsége n korábbi lemma szerint: $\overset{m}{\text{útköltség}(m)} = \overset{n}{\text{útköltség}^*(m)}$ optimális úton fekvő csomópont. Legyen az első ilyen. Ekkor $\overset{m}{\text{összköltség}(n)} \leq \overset{n}{\text{összköltség}(m)}$. Az algoritmus az csúcsot választotta kiterjesztésre, tehát n . De

$$\begin{aligned} \text{összköltség}(m) &= \text{útköltség}^*(m) + \text{heurisztika}(m) \leq \\ &\leq \text{útköltség}^*(m) + \text{heurisztika}^*(m) = \text{összköltség}^*(m) = \text{összköltség}^*(s), \end{aligned} \quad (2.23)$$

amiből $\text{összköltség}(n) \leq \text{összköltség}^*(s)$ következik.

A lemmát a következőképpen is megfogalmazhatjuk: Annak szükséges feltétele, hogy az A^* algoritmus egy n csomópontot kiterjesztésre kiválasszon:

$$\text{összköltség}(n) \leq \text{összköltség}^*(s). \quad (2.24)$$

Tehát az

$$S = \{n \mid \text{összköltség}(n) > \text{összköltség}^*(s)\} \quad (2.25)$$

csomópontok nem kerülnek soha kiterjesztésre, nem is kell őket a keresőfában őrizni. Nem ismerjük ugyanakkor az $\text{összköltség}^*(s)$ értéket. Becsüljük meg: legyen $K \geq \text{összköltség}^*(s)$. Ekkor az

$$S' = \{n \mid \text{összköltség}(n) > K\} \subseteq S \quad (2.26)$$

csomópontokat a keresőfából elhagyhatjuk. Annak elegendő feltétele, hogy az A^* algoritmus egy n csomópontot kiterjesztésre kiválasszon:

$$\text{összköltség}(n) < \text{összköltség}^*(s). \quad (2.27)$$

Definíció:

Legyen P és Q két A^* algoritmus. Azt mondjuk, hogy P jobban informált, mint Q , ha célállapotot tartalmazó csomópontok kivételével bármely n csomópontra

$$\text{heurisztika}_P(n) > \text{heurisztika}_Q(n) \quad (2.28)$$

teljesül, ahol heurisztika_P és heurisztika_Q a P és Q algoritmusok heurisztikus függvényei. (Más szóval: a P algoritmus alulról pontosabban becsli a hátralévő út költségét bármely csúcsban.)

Tétel:

Ha P jobban informált A^* algoritmus Q -nál, akkor minden olyan csomópontot, amelyet P kiterjeszt, kiterjeszt Q is.

Bizonyítás:

Legyen n egy olyan nyílt csomópont, melyet P éppen kiválaszt kiterjesztésre! Ekkor

$$\text{összköltség}_P(n) \leq \text{összköltség}^*(s). \quad (2.29)$$

A P keresőgrájában az s -ből n -be vezető út tetszőleges n' csúcsára szintén teljesül az

$$\text{összköltség}_P(n') \leq \text{összköltség}^*(s) \quad (2.30)$$

összefüggés. Mit csinál ezzel az n' csúccsal a Q algoritmus? Mivel

$$\begin{aligned} \text{összköltség}_Q(n') &= \text{útköltség}(n') + \text{heurisztika}_Q(n') < \\ &< \text{útköltség}(n') + \text{heurisztika}_P(n') = \text{összköltség}_P(n') \leq \text{összköltség}^*(s), \end{aligned} \quad (2.31)$$

azaz $\text{összköltség}_Q(n') < \text{összköltség}^*(s)$, ezért n' -t a Q algoritmus is kiterjeszti. Az n' tetszőleges volt, így a Q algoritmus az s -ből n -be vezető út minden csúcsát kiterjeszti, beleértve n -et is.

2.9. A monoton A algoritmus

Definíció:

Azt mondjuk, hogy egy h heurisztikus függvény kielégíti a *monoton* megszorítás feltételét, ha értéke bármely él mentén legföljebb az illető él költségével csökken, azaz minden $(n, m) \in E$ él esetén

$$h(n) - h(m) \leq \text{költség}(n, m). \quad (2.32)$$

Tétel:

Ha egy heurisztikus függvény kielégíti a monoton megszorítás feltételét, akkor

$$h(n) \leq h^*(n) \quad (2.33)$$

teljesül minden $n \in N$ -re.

Bizonyítás:

A bizonyítás két részből áll:

1. Ha az n csúcsból nem vezet út terminálisba, akkor $h^*(n) = \infty$.
2. Ha van ilyen út, akkor legyen $n = n_0, n_1, n_2, \dots, n_l = t \in T$ optimális út. Ennek éleire

$$\begin{aligned} h(n) - h(n_1) &\leq \text{költség}(n, n_1) \\ h(n_1) - h(n_2) &\leq \text{költség}(n_1, n_2) \\ &\vdots \\ h(n_{l-1}) - h(t) &\leq \text{költség}(n_{l-1}, t) \end{aligned} \quad (2.34)$$

teljesül. Az egyenlőtlenségeket összeadva

$$h(n) - h(t) \leq \sum_{i=1}^l \text{költség}(n_{i-1}, n_i) = h^*(n) \quad (2.35)$$

adódik, ahol a bal oldal $h(n)$, mivel $h(t) = 0$, lévén t terminális csúcs. Így $h(n) \leq h^*(n)$.

Definíció:

Monoton A algoritmusnak nevezzük azt az *A* algoritmust, amelynek heurisztikus függvénye monoton megszorításos.

Tétel:

Amikor a monoton *A* algoritmus egy nyílt n csomópontot kiterjesztésre kiválaszt, akkor n -be már optimális utat talált, azaz $\text{útköltség}(n) = \text{útköltség}^*(n)$.

Bizonyítás:

Tegyük föl indirekt módon, hogy amikor az n csúcsot kiterjesztésre kiválasztja az algoritmus, $\text{útköltség}(n) > \text{útköltség}^*(n)$. Legyen m egy olyan nyílt csúcs, amely egy S -ből n -be vezető optimális úton van és amelyre $\text{útköltség}(m) = \text{útköltség}^*(m)$ teljesül. Az indirekt föltevés miatt n és m nem lehet ugyanaz a csúcs. Mivel azonban az algoritmus n -et választotta m helyett, ez azt jelenti, hogy $\text{összköltség}(n) \leq \text{összköltség}(m)$. Ugyanakkor az m -ből n -be vezető $m = m_0, m_1, m_2, \dots, m_{l-1}, m_l = n$ optimális útvonalra felírható a következő összefüggés:

$$\begin{aligned} \text{összköltség}(m) &= \text{útköltség}(m) + \text{heurisztika}(m) = \\ &= \text{útköltség}^*(m) + \text{heurisztika}(m) \leq \\ &\leq \text{útköltség}^*(m) + \text{költség}(m, m_1) + \text{heurisztika}(m_1) = \\ &= \text{útköltség}^*(m_1) + \text{heurisztika}(m_1) \leq \\ &\leq \text{útköltség}^*(m_1) + \text{költség}(m_1, m_2) + \text{heurisztika}(m_2) = \\ &= \text{útköltség}^*(m_2) + \text{heurisztika}(m_2) \leq \\ &\vdots \\ &\leq \text{útköltség}^*(m_{l-1}) + \text{költség}(m_{l-1}, n) + \text{heurisztika}(n) = \\ &= \text{útköltség}^*(n) + \text{heurisztika}(n) < \\ &< \text{útköltség}(n) + \text{heurisztika}(n) = \text{összköltség}(n) \end{aligned} \quad (2.36)$$

A levezetésből azt kaptuk, hogy $\text{összköltség}(m) < \text{összköltség}(n)$, ami ellentmond annak, hogy az n csomópontot választjuk ki.

Chapter 3. Kétszemélyes stratégiai játékok és lépésajánló algoritmusok

Stratégiai játékok azok a játékok, melyekben játékosoknak a játék kimenetelére (ellenőrizhető módon) van befolyásuk. Ilyen játékok pl. a sakk, a bridzs, a póker, az üzleti „játékok” mint két vállalat konkurrenca harca, harci „játékok”.

Néhány a játékelméleti kutatásokban fontos név:

1921 E. Borel

1928 Neumann János

1944 Neumann János és O. Morgenstein

1994 Harsányi János (közgazdasági Nobel-díj)

Egy játék leírásához meg kell adni

- a játék lehetséges állásait (helyzeteit),
- a játékosok számát,
- hogyan következnek lépni az egyes játékosok (pl. egy időben vagy felváltva egymás után),
- egy-egy állásban a játékosoknak milyen lehetséges lépései (lehetőségei) vannak,
- a játékosok milyen -- a játékkal kapcsolatos -- információval rendelkeznek a játék folyamán,
- van-e a véletlennek szerepe a játékban és hol,
- milyen állásban kezdődik és mikor ér véget a játék,
- és az egyes játékosok mikor, mennyit nyernek, illetve veszítenek.

Osztályozás

- a játékosok száma szerint: pl. *egy-, két-, n-személyes játékok*;
- ha a játszma állásból állásba vivő lépések sorozata *diszkrét* a játék;
- ha az állásokban véges sok lehetséges lépése van minden játékosnak és a játszmák véges sok lépés után véget érnek *véges* a játék;
- ha a játékosok a játékkal kapcsolatos összes információval rendelkeznek a játék folyamán, *teljes információjú* a játék;
- ha nincs a véletlennek szerepe a játékban, *determinisztikus* a játék;
- a játékosok nyereségeinek és veszteségeinek összege 0 , *zérusösszegű* a játék.

A továbbiakban játék alatt kétszemélyes, diszkrét, véges, teljes információjú, determinisztikus, zérusösszegű stratégiai játékot fogunk érteni.

1. A játékok reprezentációja

Jelölje a két játékost A és B , a játékállások halmazát H . A játékot az $a_0 \in H$ kezdőállásban kezdje $J_0 \in \{A, B\}$. Tegyük fel, hogy a játékosok a játék során felváltva lépnek, és ismerjük az egyes állásokban megtehető lépéseket: $\{l \mid l : H \rightarrow H\}$. Az l lépés egy a állásban akkor tehető meg, ha

l -megtételeinek-előfeltétele(a). A játék az a állásban véget ér, ha $végállás(a)$. A szabályok leírják, itt ki a nyerő játékos: $nyer : \{a \mid végállás(a)\} \rightarrow \{jön_a, volt_a\}$, ahol $jön_a$ lenne az a állásban soron következő játékos ($jön_a \neq volt_a$). E játék állapotter-reprezentációja az az $(\mathcal{A}, kezdő, \nu, o)$ négyes, ahol

- $\mathcal{A} = \{(a, J) \mid a \in H, J \in \{A, B\}, J \text{ következik lépni}\}$,
- $kezdő = (a_0, J_0)$
- $\nu = \{(a, J) \mid végállás(a), J \text{ nyer, ha } nyer(a) = jön_a\}$
- $o = \{o_i \mid o_i(a, J) = (l(a), I), I, J \in \{A, B\}, I \neq J\}$

A játék állapotter-reprezentációját szemléltető gráf a *játékgráf*. „Egyenesítsük ki” a játékgráfot fává. A *játékfaban*

- páros szinteken lévő állásokban a kezdő játékos, páratlan szinteken lévőekben pedig az ellenfele léphet;
- egy állást annyi különböző csúcs szemléltet, ahány különböző módon a játék során a kezdőállásból eljuthatunk hozzá;
- véges hosszúságúak az utak, hisz véges játékokkal foglalkozunk.

Ha a játék során $kezdő$ állapotból a játékosok valamelyik $v \in \mathcal{V}$ állapotba érnek, azaz $kezdő \xrightarrow{o_1, \dots, o_r}^* v$ ($r \geq 1$), azt mondjuk lejátszottak egy *játszmát*. A játszmákat a játékfaban a startcsúcsból a levelelembe vezető utak szemléltetik. Egy játék játékfabja a játék összes lehetséges játszmáját szemlélteti a startcsúcsból induló, a különböző levelekben végződő útjaival.

Definíció

Az (N, HE) párt *ÉS/VAGY gráfnak* nevezzük:

- N nemüres halmaz, a gráf csúcsainak halmaza,
- $HE \subseteq \{(n, M) \in N \times 2^N \mid 0 \neq |M| < \infty\}$ pedig az irányított *hiperélek* halmaza.

Definíció

$$(n_1, \{n_{11}, n_{12}, \dots, n_{1k_1}\}),$$

$$(n_2, \{n_{21}, n_{22}, \dots, n_{2k_2}\}),$$

⋮

$$(n_r, \{n_{r1}, n_{r2}, \dots, n_{rk_r}\})$$

Az *ÉS/VAGY* gráfban a gráf hiperéleinek egy olyan $(n_r, \{n_{r1}, n_{r2}, \dots, n_{rk_r}\})$ sorozata, ahol

$$\forall i \forall j ((i \neq j) \supset (n_i \cap n_j = \emptyset)) \wedge$$

$$\wedge \forall i ((i > 1) \supset \exists j ((i > j) \wedge (n_i \in \{n_{j1}, n_{j2}, \dots, n_{jk_j}\}))),$$

a gráf egy *hiperútja*.

2. A stratégia

Definíció

A J játékos *stratégiája* egy olyan

$$s_J : \{(a, J) \mid (a, J) \in \mathcal{A}\} \rightarrow \mathcal{O} \quad (3.1)$$

döntési terv, amely J számára előírja, hogy a játék során előforduló azon állásokban, melyekben J következik lépni, a megtehető lépései közül melyiket lépje meg.

A J játékos stratégiáinak szemléltetése a játékfában:

Alakítsuk át a játékfát ÉS/VAGY fává J játékos szempontjából: J lépéseit szemléltető élek mindegyike egy élből álló hiperél marad (VAGY élek), ellenfelének egy-egy állásból megtehető lépéseit szemléltető élköteg egy-egy hiperél lesz (ÉS élek). Ebben az ÉS/VAGY gráfban J stratégiáit a startcsúcsból induló olyan hiperutak szemléltethetik, melyek levelei az eredeti játékgráfnak is levelei.

Definíció

Tegyük fel, hogy a J játékos az s_J stratégiájával játszik. Ekkor csak az s_J -t szemléltető hiperutat alkotó közös utak által szemléltetett játszmák játszhatók le.

Lemma

Tegyük fel, hogy az A játékos az s_A , a B játékos pedig az s_B stratégiájával játszik. A két stratégia egyértelműen meghatározza a lejátszható játszmát.

Definíció

A J játékos stratégiáját J *nyerő stratégiájának* nevezzük, ha (az ellenfelének stratégia-választásától függetlenül) minden a stratégia alkalmazása mellett lejátszható játszmában J nyer.

Megjegyzés

A J szempontjából átalakított ÉS/VAGY fában a J nyerő stratégiát szemléltető hiperút levelelemei mind J -nyerő állások.

Tétel

(Az általunk vizsgált) minden játék esetén valamelyik (de nyilván csak az egyik) játékos számára van nyerő stratégia.

Bizonyítás

Rögzítsünk tetszőlegesen egy játékot, és tegyük fel, hogy adott a teljes játékfa.

- Ekkor a fa leveleit címkézzük A -val, ha a levél olyan végállást szemléltet, ahol A nyer, illetve B -vel, ha a levél olyan végállást szemléltet, ahol B nyer.
- Címkézzük szintenként csökkenő sorrendben a nem levél csúcsokat is: ha a csúcsban $J \in \{A, B\}$ következik lépni és van J címkéjű gyermeke, J címkét kap, egyébként az ellenfél címkéjét.

Mivel a játékfá véges, végül a startcsúcs is címkét kap. A teljes indukció elve alapján ez a címke egyértelmű. A startcsúcs címkéje pedig megmutatja, melyik játékosnak van nyerő stratégiája, és magát a nyerő stratégiá(ka)t is le lehet olvasni a felcímkézett játékfáról.

3. Minimax algoritmus

Cél: a támogatott játékosnak, J -nek, egy adott állásban „elég jó” lépést ajánlani. Az algoritmus számára át kell adni

- a játék $(A, kezdő, v, o)$ reprezentációját,
- J azon a állását, ahol lépni következik,
- az állások „jóságát” J szempontjából becslő $h_J : A \rightarrow R$ heurisztikát
- és egy mélységi *korlát* -ot.

3.1. Az algoritmus fő lépései

1. A játékfá (a, J) állapotot szemléltető csúcsából kiinduló részének előállítás *korlát* mélységig.
2. A részfa leveleiben található állások jóságainak becslése a heurisztika segítségével:
 $jóság(n_b) = h_J(b)$.
3. Szintenként csökkenő sorrendben a részfa nem levél csúcsai jóságainak számítása: ha az n csúcs gyermekei rendre n_1, \dots, n_k , akkor

$$jóság(n) = \begin{cases} \max\{jóság(n_1), \dots, jóság(n_k)\} & \text{ha } n \text{ szintje páros,} \\ \min\{jóság(n_1), \dots, jóság(n_k)\} & \text{ha } n \text{ szintje páratlan.} \end{cases} \quad (3.2)$$

Javaslat: az a állásból egy olyan lépést tegyen meg J , amelyik az n_a csúcs „jóság” értékével megegyező értékű gyermekébe vezet.

```

1  function Minimax-lépés(A, kezdő, V, O, állapot, korlát, hJ)
    max ← -∞
    operátor ← Nil
5   for all o ∈ O do
        if Előfeltétel(állapot, o) then
            új-állapot ← Alkalmaz(állapot, o)
            v ← Minimax-Érték(A, kezdő, V, O, új-állapot, korlát - 1, hJ)
            if v ≥ max then
10                max ← v
                    operátor ← o
            end if
        end if
    end for
15  return operátor
end function

1  function Minimax-Érték(A, kezdő, V, O, állapot, mélység, hJ)
    if állapot ∈ V or mélység = 0 then
        return hJ(állapot)
5   else if Játékos[állapot] = J then

```

```

max ← -∞
for all o ∈ O do
  if Előfeltétel(állapot, o) then
    új-állapot ← Alkalmaz(állapot, o)
10    v ← Minimax-Érték(A, kezdő, V, O, új-állapot, mélység - 1, hJ)
    if v ≥ max then
      max ← v
    end if
  end if
15 end for
return max
else
min ← ∞
for all o ∈ O do
20   if Előfeltétel(állapot, o) then
    új-állapot ← Alkalmaz(állapot, o)
    v ← Minimax-Érték(A, kezdő, V, O, új-állapot, mélység - 1, hJ)
    if v ≤ min then
      min ← v
25   end if
  end if
end for
return min
end if
30 end function

```

4. Negamax algoritmus

Cél: a támogatott játékosnak, J -nek, egy adott állásban „élég jó” lépést ajánlani. Az algoritmus számára át kell adni

- a játék $(\mathcal{A}, \text{kezdő}, v, o)$ reprezentációját,
- J azon a állását, ahol lépni következik,
- az állások „jóságát” a soron következő játékos szempontjából becslő $h : \mathcal{A} \rightarrow R$ heurisztikát
- és egy mélységi *korlát*-ot.

4.1. Az algoritmus fő lépései:

1. A játéka (a, J) állapotot szemléltető csúcsából kiinduló részének előállítás *korlát* mélységgig.
2. A részfa leveleiben található állások jóságainak becslése a heurisztika segítségével: $\text{jóság}(n_b) = h(b)$.
3. Szintenként csökkenő sorrendben a részfa nem levél csúcsai jóságainak számítása: ha az n csúcs gyermekei rendre n_1, \dots, n_k , akkor

$$\text{jóság}(n) \doteq \max\{-\text{jóság}(n_1), \dots, -\text{jóság}(n_k)\} \quad (3.3)$$

Javaslat: az a állásból egy olyan lépést tegyen meg J , amelyik az n_a csúcs „jóság” értékének -1 -szeresével megegyező értékű gyermekébe vezet.

```

1  function Negamax-lépés(A, kezdő, V, O, állapot, korlát, h)
    max ← -∞
    operátor ← Nil
5  for all o ∈ O do

```

```
    if Előfeltétel(állapot, o) then
      új-állapot ← Alkalmaz(állapot, o)
      v ← -Negamax-Érték(A, kezdő, V, O, új-állapot, korlát - 1, h)
      if v ≥ max then
10      max ← v
        operátor ← o
      end if
    end if
  end for
15 return operátor
end function

1
function Negamax-Érték(A, kezdő, V, O, állapot, mélység, h)
  if állapot ∈ V or mélység = 0 then
    return h(állapot)
  5 else
    max ← -∞
    for all o ∈ O do
      if Előfeltétel(állapot, o) then
10      új-állapot ← Alkalmaz(állapot, o)
        v ← -Negamax-Érték(A, kezdő, V, O, új-állapot, mélység - 1, h)
        if v ≥ max then
          max ← v
        end if
      end if
    end for
15 return max
  end if
end function
```

Chapter 4. Problémamegoldás redukcióval

Gyakran előfordul, hogy egy problémát úgy próbálunk megoldani, hogy több külön-külön megoldandó részproblémára bontjuk. Ha a részproblémákat megoldjuk, az eredeti probléma megoldását is megkapjuk. A részproblémák megoldását további részek megoldására vezetjük vissza, egészen addig, amíg csupa olyan problémához nem jutunk, amelyeket egyszerűségükönél fogva már könnyedén meg tudunk oldani. A probléma megoldásnak ezt a módját *problémaredukciónak* nevezzük.

1. Problémaredukciós reprezentáció

- Először is le kell írni az eredeti problémát, jelöljük ezt most \mathcal{P} -vel.
- Egy probléma részproblémákra bontása során a nyert részek az eredeti problémához hasonló, de annál egyszerűbb problémák.

Jelöljük az így nyert problémahalmazt \mathcal{P} -vel. Természetesen $\mathcal{P} \in \mathcal{P}$.

- \mathcal{P} problémáinak összegyűjtése során törekszünk arra, hogy legyenek közöttük olyanok, melyeket meg tudunk oldani, vagy ismerjük a megoldásukat. Ezek a problémák az ún. *egyszerű problémák*.

Az egyszerű problémák halmazát \mathcal{E} -vel jelöljük.

$\mathcal{E} \subset \mathcal{P}$, hiszen $\mathcal{P} \notin \mathcal{E}$, különben nincs megoldandó feladat.

- Meg kell még adni a problémákat egyszerűsítő, illetve részekre bontó *redukciós operátorokat*. Egy redukciós operátor egy problémához azokat a (rész)problémákat rendeli hozzá, melyek egyenkénti megoldásával a probléma megoldása is előáll. Jelöljön a *redukciós operátorok* \mathcal{R} véges halmazából r egy operátort. Ekkor

$$\begin{aligned} \text{Dom}(r) &= \{q \mid q \in \mathcal{P} \setminus \mathcal{E} \text{ és } r\text{-alkalmazásának-előfeltétele}(q)\} \\ &\text{és} \\ \text{Rng}(r) &= \{r(q) = \{q_1, \dots, q_m\} \mid q \in \text{Dom}(r) \text{ és } q_1, \dots, q_m \in \mathcal{P}\} \end{aligned} \tag{4.1}$$

Tehát egy redukciós operátor egy-egy problémához \mathcal{P} egy-egy részhalmazát rendeli, így értékészlete \mathcal{P} hatványhalmazának valamely részhalmaza.

Definíció:

Legyen \mathcal{P} egy probléma. Azt mondjuk, hogy a \mathcal{P} problémát *problémaredukciós reprezentációval* írtuk le, ha megadtuk a $(\mathcal{P}, \mathcal{P}, \mathcal{E}, \mathcal{R})$ négyest, azaz

- a megoldandó $\mathcal{P} \in \mathcal{P}$ problémát,
- a $\mathcal{P} \neq \emptyset$ halmazt, a \mathcal{P} problémához hasonló problémák halmazát,
- az egyszerű problémák $\mathcal{E} \subset \mathcal{P}$ halmazát és
- a redukciós operátorok $\mathcal{R} \neq \emptyset$ véges halmazát.

Jelölése: $\langle \mathcal{P}, p, \mathcal{E}, \mathcal{R} \rangle$.

Definíció:

Legyen a p probléma a $\langle \mathcal{P}, p, \mathcal{E}, \mathcal{R} \rangle$ reprezentációval leírva és legyenek

$$Q = \{p_1, p_2, \dots, p_i, \dots, p_n\} \subseteq \mathcal{P} \quad (4.2)$$

$$Q' = \{p_1, p_2, \dots, p_{i-1}, q_1, q_2, \dots, q_m, p_{i+1}, \dots, p_n\} \subseteq \mathcal{P} \quad (4.3)$$

egy-egy problémahalmaz ($n \geq 1, m \geq 1$). Azt mondjuk, hogy a Q problémahalmaz *egy lépésben* vagy *közvetlenül redukálható* a Q' problémahalmazzá, ha van olyan $r \in \mathcal{R}$ redukciós operátor, melyre $p_i \in \text{Dom}(r)$, és

$$r(p_i) = \{q_1, q_2, \dots, q_m\}. \quad (4.4)$$

Ennek jelölése: $Q \triangleleft Q'$, illetve ha fontos, hogy az r redukciós operátor segítségével állítottuk elő Q -ből a Q' -t, akkor $Q \triangleleft_r Q'$.

Definíció:

Legyen a p probléma reprezentációja $\langle \mathcal{P}, p, \mathcal{E}, \mathcal{R} \rangle$, és $Q, Q' \subseteq \mathcal{P}$. A Q -ből a Q' *redukálható*, ha van olyan $P_1, \dots, P_k \subseteq \mathcal{P}$ ($k \geq 2$) véges problémahalmaz-sorozat, hogy

$$P_1 = Q, \quad P_k = Q' \quad (4.5)$$

és $P_i \triangleleft P_{i+1}$ minden $1 \leq i \leq k-1$ esetén. Jelölése: $Q \triangleleft^* Q'$.

Definíció:

Nyilvánvaló, hogy ha $P_i \triangleleft P_{i+1}$ minden $1 \leq i \leq k-1$ esetén, akkor van olyan r_1, r_2, \dots, r_{k-1} redukciós operátorsorozat, hogy $P_i \triangleleft_{r_i} P_{i+1}$ ($1 \leq i \leq k-1$). Ilyenkor azt mondjuk, hogy a Q problémahalmazt a Q' problémahalmazzá az r_1, r_2, \dots, r_{k-1} redukciós operátorsorozat segítségével redukáltuk. Jelölve: $Q \triangleleft_{r_1, \dots, r_{k-1}}^* Q'$.

Definíció:

Legyen a p probléma problémaredukciós reprezentációja $\langle \mathcal{P}, p, \mathcal{E}, \mathcal{R} \rangle$. A p probléma *megoldható* ebben a reprezentációban, ha $\{p\}$ csupa egyszerű problémából álló problémahalmazzá redukálható, azaz

$$\{p\} \triangleleft_{r_1, \dots, r_l}^* Q \subseteq \mathcal{E}. \quad (4.6)$$

Ekkor az r_1, \dots, r_l redukciós operátorsorozatot tekinthetjük a *probléma megoldásának*.

A feladatunk lehet

- annak eldöntése, hogy megoldható-e a probléma az adott problémaredukciós reprezentációban,
- egy (esetleg az összes) megoldás előállítás,
- valamilyen minősítés alapján jó megoldás előállítás (a megoldások között különbséget tehetünk, pl. a megoldás költsége alapján).

Jelölje $\text{költség}(r, q) \geq 0$ az r redukciós operátor q problémára való alkalmazásának a költségét, és ha $q \in \mathcal{E}$, akkor $c(q) \geq 0$ pedig a q egyszerű probléma közvetlen megoldásának költségét.

Definíció:

A $(\mathcal{P}, p, \mathcal{E}, \mathcal{R})$ problémaredukciós reprezentációban $g(q)$ a $q \in \mathcal{P}$ probléma megoldásának minimális költsége, ha

$$g(q) = \begin{cases} c(q) & \text{ha } q \in \mathcal{E}, \\ \infty & \text{ha } q \notin \mathcal{E}, \text{ de } \neg \exists r (r \in \mathcal{R} \wedge q \in \text{Dom}(r)), \\ \min\{\text{költség}(r, q) + \sum_{q_i \in r(q)} g(q_i)\} & \text{egyébként.} \end{cases} \quad (4.7)$$

A részproblémák párhuzamos megoldása esetén lehetőségünk van a legrövidebb idő alatt előállítható megoldás megkeresésére. Ekkor $\text{költség}(r, q)$ az r redukciós operátor q problémára való alkalmazásának a végrehajtási idejét, $c(q)$ pedig a q egyszerű probléma közvetlen megoldásának idejét jelenti.

Definíció:

A $(\mathcal{P}, p, \mathcal{E}, \mathcal{R})$ problémaredukciós reprezentációban $t(q)$ a $q \in \mathcal{P}$ probléma megoldásának minimális ideje, ha

$$t(q) = \begin{cases} c(q) & \text{ha } q \in \mathcal{E}, \\ \infty & \text{ha } q \notin \mathcal{E}, \text{ de } \neg \exists r (r \in \mathcal{R} \wedge q \in \text{Dom}(r)), \\ \min\{\text{költség}(r, q) + \max_{q_i \in r(q)} t(q_i)\} & \text{egyébként.} \end{cases} \quad (4.8)$$

2. Példák problémaredukciós reprezentációra

2.1. Hanoi tornyai

A legenda szerint egy szerzetesek lakta távol-keleti kolostor udvarán áll három rúd, amelyeken 64 különböző átmérőjű aranykorong található. Eredetileg mind a 64 korong egyetlen rúdra volt ráakva úgy, hogy minden korong alatt egy nála nagyobb volt. A szerzeteseknek az a feladatuk, hogy a korongokat helyezték át az első rúdról a harmadik rúdra, egyszerre mindig csak egyet mozgatva úgy, hogy sohase rakjanak nagyobb korongot kisebbre. Amint mind a 64 korongot átpakolják a harmadik rúdra, eljön majd a világvége.

A legenda szerint a szerzetesek a munka elvégzésével a legidősebb társukat bízták meg. Sokat törte a fejét, gondolkodott, meditált, majd hirtelen világosság töltötte el: a feladatot három lépésben meg tudja oldani!

1. lépés: Át kell vinni az első rúdon lévő felső 63 korongból álló tornyot a második rúdra.
2. lépés: Át kell vinni az első rúdon lévő utolsó, legnagyobb korongot a harmadik rúdra.
3. lépés: Át kell vinni a második rúdon lévő 63 korongból álló tornyot a harmadik rúdra.

A szerzetes másnap kiszögezte a templom kapujára az algoritmus leírását:

Módszer és út arra vonatkozóan hogy hogyan vigyünk át egy n korongból álló tornyot az X rúdról az Y -ra a Z felhasználásával:

1. Abban az esetben, ha a torony egynél több korongból áll, bízd meg a legöregebb tanítványodat, hogy a szobán forgó torony felső $n - 1$ korongját vigye át az X rúdról a Z -re, miközben az Y -t használhatja.
2. Vidd át magad az X rúdon maradt egyetlen korongot az Y -ra.
3. Abban az esetben, ha a torony egynél több korongból állt, bízd meg a legöregebb tanítványodat, hogy a szobán forgó torony felső $n - 1$ korongját vigye át a Z rúdról az Y -ra, miközben az X -t használhatja.

A legenda szerint tehát a hanoi szerzetesek problémaredukcióval próbálták megoldani az előttük álló feladatot. Adjuk meg most az elképzelésüknek megfelelő reprezentációt a módosított feladatra. A megoldandó P feladat tehát: mindhárom az A rúdon levő korong átvitele a C rúdra (B felhasználásával). Ezt jelölhetjük a következőképpen:

$$p = (3; A \rightarrow C) \quad (4.9)$$

A megoldandó feladathoz hasonló feladatok a következők: az X rúdon levő felső valahány korong átvitele a Y rúdra (Z felhasználásával):

$$\mathcal{P} = \{(n; x \rightarrow y) \mid n \geq 1, x, y \in \{A, B, C\}, x \neq y\} \quad (4.10)$$

Ezek közül egyszerűen megoldhatók, ha a felső korongot kell áthelyezni az X rúdról egy olyan rúdra, amelyiken nincs ennél kisebb átmérőjű: s Minden nem egyszerű problémát három, az eredetnél egyszerűbb részre bonthatunk:

3. A problémaredukciós reprezentációt szemléltető gráf

Legyen a P probléma a $(\mathcal{P}, p, \mathcal{E}, \mathcal{R})$ reprezentációval megadva. Ez a reprezentáció is egy irányított gráfot, ún. *ÉS/VAGY* gráfot határoz meg.

- A \mathcal{P} problémahalmaz elemei (a problémák) a gráf csúcsai. Vezessük be az $q \in \mathcal{P}$ probléma által definiált csúcsra az n_q jelölést. Ekkor a gráf csúcsainak halmaza

$$N = \{n_q \mid q \in \mathcal{P}\}. \quad (4.11)$$

- A gráf csúcsai közül kitéüntetett szerepet játszanak a P problémát szemléltető ún. *startcsúcs* (jele: n_p vagy s),
- és az egyszerű problémákat szemléltető *terminális csúcsok*. A terminális csúcsok halmaza tehát:

$$T = \{n_e \mid e \in \mathcal{E}\}. \quad (4.12)$$

- Egy $q \in \mathcal{P}_{n_{q_1}, \dots, n_{q_m}}$ éltető csúcsból irányít $\{q\} \prec \{q_1, q_2, \dots, q_m\} \in \mathcal{P}$ problémákat szemléltető csúcsokba, amikor . Ezek az élek összetartozónak tekinthetők: egy *ÉS élköteget* vagy *hiperélt* alkotnak. A gráf hiperéleinek halmaza tehát a következő:

Azt mondjuk, hogy az (N, s, T, E) irányított ÉS/VAGY gráf a \mathcal{P} probléma $(\mathcal{P}, p, \mathcal{E}, \mathcal{R})$ problémaredukciós reprezentációjához tartozó *reprezentációs gráfja*.

Lemma

Legyen (N, s, T, E) a \mathcal{P} probléma $(\mathcal{P}, p, \mathcal{E}, \mathcal{R})$ problémaredukciós reprezentációjához tartozó reprezentációs gráfja. Pontosán akkor áll fenn a $\{q\} \prec^* \{q_1, q_2, \dots, q_m\}$ reláció, ha a reprezentációs gráfban van az n_q csúcsából induló olyan hiperút, melynek levelei éppen az $\{n_{q_1}, n_{q_2}, \dots, n_{q_m}\}$ csúcsok.

Bizonyítás

1. Tegyük fel, hogy $\{q\} \prec^* \{q_1, q_2, \dots, q_m\}$. Ez a redukálhatóság definíció miatt azt jelenti, hogy létezik olyan $P_1, P_2, \dots, P_k \subseteq \mathcal{P}$ ($k \geq 2$) (véges) problémahalmaz-sorozat úgy, hogy

$$P_1 = \{q\}, \quad P_k = \{q_1, q_2, \dots, q_m\} \quad (4.13)$$

és

$$P_i \prec P_{i+1} \quad (4.14)$$

minden $1 \leq i \leq k - 1$ esetén.

- Tehát minden $1 \leq i \leq k - 1$ -re P_i valamelyik problémája, mondjuk $P_{ij} \in P_{i+1}$ -ben már részekre van bontva, azaz van olyan redukciós operátor, amelyik P_{ij} -t épp ezekre a részproblémákra bontja, így a reprezentációs gráfban a P_{ij} -t szemléltető csúcsból ÉS élköteg indul a részproblémákat szemléltető csúcsokba.

- Továbbá $P_{ij} \in P_{i+1}$ -ben már nem szerepel, tehát újabb hiperél már nem indul belőle.

Azaz a reprezentációs gráfunkban egy $k - 1$ hiperélből álló sorozatunk van, melyben az első hiperél a q -t szemléltető n_q csúcsból indul, minden következő hiperél kezdőcsúcsa valamely előző hiperél végcsúcsa, és minden csúcsból legfeljebb egy hiperél indul. Tehát a szemléltető részgráf egy hiperút.

Továbbá a sorozat utolsó halmazának, P_k -nak a problémái azok, amiket nem bontottunk tovább, tehát az ezeket szemléltető csúcsok a a hiperút levelei.

2. Most tegyük fel azt, hogy a reprezentációs gráf n_q csúcsából ind $(n_1, \{n_{11}, n_{12}, \dots, n_{1m_1}\})$,
 $(n_2, \{n_{21}, n_{22}, \dots, n_{2m_2}\})$,
 \vdots
 $\{n_{q1}, n_{q2}, \dots, n_{qm}\}$ $(n_k, \{n_{k1}, n_{k2}, \dots, n_{km_k}\})$

csúcsok. Ez azt jelenti, hogy van olyan hiperélsorozat a reprezentációs gráfban, hogy

$$n_q = n_1, \tag{4.15}$$

továbbá

$$\forall i \forall j ((1 \leq i, j \leq k) \wedge (i \neq j) \supset (n_i \neq n_j)) \tag{4.16}$$

és

$$\forall i ((1 < i \leq k) \supset \exists j ((i > j \leq k) \wedge (n_i \in \{n_{j1}, n_{j2}, \dots, n_{jm_j}\}))). \tag{4.17}$$

A sorozat minden $(n_i, \{n_{i1}, n_{i2}, \dots, n_{im_i}\})$ hiperéle egy redukciós operátoralkalmazást szemléltet: az n_i által szepléltetett problémát bontja a redukciós operátor az $\{n_{i1}, n_{i2}, \dots, n_{im_i}\}$ csúcsok által szemléltetett problémákká. Tehát a hiperélsorozat egy redukciós operátorsorozat, mely első operátorát Q -ra alkalmaztuk, az összes többit pedig, valamely megelőző operátor eredményeképpen előállt problémára.

Tétel

Legyen (N, s, T, E) a P probléma $(\mathcal{P}, p, \mathcal{E}, \mathcal{R})$ problémaredukciós reprezentációjához tartozó reprezentációs gráfja. Pontosán akkor oldható meg P , ha van a reprezentációs gráfban a startcsúcsból induló olyan hiperút, melynek levelei terminális csúcsok.

4. Problémaredukcióval reprezentált feladatok megoldáskereső módszerei

4.1. Visszalépéses megoldáskereső ÉS/VAGY fák esetén

Legyen $(\mathcal{P}, p, \mathcal{E}, \mathcal{R})$ a P probléma problémaredukciós reprezentációja. Egy visszalépéses megoldáskereső

- *adatbázisa* a reprezentációs gráf egy a startcsúcsból induló hiperútját tartalmazza. Ezt az utat *aktuális hiperútnak* nevezzük. Az adatbázis az aktuális hiperút csúcsait és e csúcsokból kiinduló bizonyos hiperéleket (explicit vagy implicit módon) nyilvántartó csomópontokból épül fel.

A keresés megkezdésekor az adatbázis egyetlen egy - a P kezdőproblémát tartalmazó - csomópontból áll. Egy csomópont az alábbi információkat tartalmazza:

- egy $q \in \mathcal{P}$ problémát;
- arra a csomópontra mutatót, mely a szülő problémát (azt a problémát, melyre redukciós operátort alkalmazva előállt q) tartalmazza;

- Q első részproblémáját (Q első ÉS gyermekét) nyilvántartó csomópontra mutatót;
- Q szülőjének Q -t követő részproblémáját (Q következő ÉS testvérét) nyilvántartó csomópontra mutatót;
- azt a redukciós operátort, amelyet Q -ra aktuálisan alkalmaztunk;
- Q -ra a keresés során már alkalmazott (vagy még alkalmazható) redukciós operátorok halmazát.
- A visszalépéses megoldáskereső műveleteit egyrészt a redukciós operátorokból származtatjuk, továbbá alkalmazhatjuk a *visszalépést*.
 - Az $r \in \mathcal{R}$ redukciós operátorból nyert művelet
 - alkalmazási előfeltétele: a kiválasztott levél csomópontban található problémára alkalmazható r , de még sikertelenül nem alkalmaztuk rá.
 - hatása:
 - A visszalépés
 - alkalmazási előfeltétele: van csomópont az adatbázisban.
 - hatása:
- Az induló adatbázis létrehozása után kezdi el a vezérlő a keresést.
 - Ha elfogytak a csomópontok az adatbázisból \rightarrow az adott reprezentációban nincs megoldás.
 - Ha van nem egyszerű problémát tartalmazó levélcsomópont az adatbázisban, akkor a vezérlő választ egyet.
 - A kiválasztott problémára választ egy még sikertelenül ki nem próbált redukciós operátort, és alkalmazza.
 - Ha ilyen nincs, visszalép.
 - Ha a hiperút minden levél csomópontja egyszerű problémát tartalmaz \rightarrow előállt egy megoldás.

4.2. Keresőfával megoldáskereső ÉS/VAGY fák esetén

Legyen $(\mathcal{P}, \mathcal{P}, \mathcal{E}, \mathcal{R})$ a \mathcal{P} probléma problémaredukciós reprezentációja. A reprezentációs gráfot alakítsuk át olyan ÉS/VAGY gráffá, melyben minden csúcsból vagy csak VAGY élek, vagy csak egy ÉS élköteg indul ki.

Keresőfával megoldást keresés esetén az

- *adatbázis* a reprezentációs gráf startcsúcsból induló felderített hiperútjai. Az adatbázis a hiperutak csúcsait és e csúcsokból kiinduló hiperéleket (explicit vagy implicit módon) nyilvántartó csomópontokból épül fel. A keresés megkezdésekor az adatbázis egyetlen egy - a \mathcal{P} kezdőproblémát tartalmazó - csomópontból áll. Egy csomópont az alábbi információkat tartalmazza:
 - egy $q \in \mathcal{P}$ problémát;
 - ha q VAGY gyermek:
 - a szülő csomópontra mutatót;

- azt a redukciós operátort, mellyel Q -t redukáljuk;
- Q következő VAGY testvérét tartalmazó csúcsra mutatót;
- Q első ÉS gyermekét nyilvántartó csomópontra mutatót;
- ha Q ÉS gyermek
 - a szülő csomópontra mutatót;
 - Q következő ÉS testvérét nyilvántartó csomópontra mutatót;
 - Q első VAGY gyermekét nyilvántartó csomópontra mutatót;
- címkét: *megoldott* / *megoldhatatlan* / *folyamatban*
- *művelete a kiterjesztés*: a keresőfát annak egy *folyamatban* címkéjű levélcsoportján keresztül kibővíti.
 - alkalmazási előfeltétele: a keresőfában van *folyamatban* címkéjű levélcsoport.
 - hatása:
 - alkalmazzuk az összes alkalmazható redukciós operátort a *folyamatban* címkéjű levélcsoport problémájára,
 - az előálló problémákat új csomópontokként felfűzzük a keresőfába megfelelő címkékkal:
 - *megoldott*, ha az előállt probléma egyszerű;
 - *folyamatban*, ha az előállt probléma nem egyszerű;
 - módosítjuk a keresőfa csúcsainak címkéit.
- - Ha a gyökér csomópont címkéje *megoldott*, előállt az adatbázisban egy megoldás.
 - Ha a gyökér csomópont címkéje *megoldhatatlan*, nincs a reprezentáció mellett a problémának megoldása.
 - Ha a gyökér csomópont címkéje *folyamatban*, a *vezérlő* megmondja, hogy melyik *folyamatban* címkéjű levélcsoport legyen a következő lépésben kiterjesztve.

Colophon

A tananyag a TÁMOP-4.1.2-08/1/A-2009-0046 számú Kelet-magyarországi Informatika Tananyag Tárház projekt keretében készült. A tananyagfejlesztés az Európai Unió támogatásával és az Európai Szociális Alap társfinanszírozásával valósult meg.

