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

List of Tables

Chapter 1. A problémareprezentáció

Table of Contents

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

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 $h_1, \ldots, h_m$ értékekkel rendelkeznek azt mondjuk, hogy

Equation 1.1.

$$p$$

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

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

Equation 1.2.

$$H_1 \times \cdots \times H_m$$

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:

Equation 1.3.

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

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

Általában $\mathcal{C} \subset \mathcal{A}$, hiszen $kezdő \notin \mathcal{C}$, 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* O véges *halmazából* o egy operátort. Ekkor

Equation 1.4.

$$\begin{aligned} \mathrm{Dom}(o) &= \{a \mid a \in \mathcal{A} \text{ és } o\text{-alkalmazásának-előfeltétele}(a)\} \\ &\text{és} \\ \mathrm{Rng}(o) &= \{o(a) \mid a \in \mathrm{Dom}(o) \text{ és } o(a) \in \mathcal{A}\} \end{aligned}$$

Definíció

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

Equation 1.5.

$$\langle \mathcal{A}, kezdő, \mathcal{C}, O \rangle$$

négyest, azaz

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

Jelölése: $p = \langle \mathcal{A}, kezdő, \mathcal{C}, O \rangle$.

Legyen p egy probléma, $\langle \mathcal{A}, kezdő, \mathcal{C}, O \rangle$ p egy állapottér-reprezentációja és legyenek $a, a' \in \mathcal{A}$.

Definíció

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

Equation 1.6.

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

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 $a_1, \ldots, a_k \in \mathcal{A}$ $(k \geq 2)$ (véges) állapotsorozat, hogy

Equation 1.7.

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

Jelölése: $a \Rightarrow^* a'$.

Ha $a_i \Rightarrow a_{i+1}$ minden $1 \leq i \leq k-1$ esetén, akkor van olyan $o_1, o_2, \ldots, o_{k-1}$ operátorsorozat, hogy $a_i \Rightarrow_{o_i} a_{i+1}$ $(1 \leq i \leq k-1)$. Ilyenkor azt mondjuk, hogy az a állapotból az a' állapot az $o_1, o_2, \ldots, o_{k-1}$ operátorsorozat segítségével elérhető. Jelölve: $a \xRightarrow[o_1, \ldots, o_{k-1}]{*} a'$.

Definíció

Legyen $p = \langle \mathcal{A}, \text{kezdő}, \mathcal{C}, O \rangle$. A p probléma *megoldható* ebben az állapottér-reprezentációban, ha van olyan $c \in \mathcal{C}$ célállapot, hogy $\text{kezdő} \Rightarrow^* c$. Ha $\text{kezdő} \xRightarrow[o_1, \ldots, o_{k-1}]{*} c$ $(r \geq 1)$, akkor az $o_1, \ldots, o_r$ operátorsorozat a probléma egy *megoldása.*

A feladatunk lehet

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

Jelölje $\text{költség}(o, a)$ az o operátor a *állapotra alkalmazásának a költségét.*

Definíció

Legyen $p = \langle \mathcal{A}, \text{kezdő}, \mathcal{C}, O \rangle$ és $o_1, \ldots, o_r \in O$ a probléma egy megoldása, azaz

Equation 1.8.

$$\text{kezdő} \xRightarrow[o_1, \ldots, o_{k-1}]{*} c \quad (c \in \mathcal{C}).$$

.

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*:

Equation 1.9.

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

Ha $\text{költség}(o, a) \equiv 1$, 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. Alkalmas függvénnyel ki lehet számolni (vagy táblázattal meg lehet adni) az operátorok alkalmazási költségeit is.

Példák állapottér-reprezentációra

A nyolcas kirakó játék

Adott nyolc számozott négyzetalakú lapocska, 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 tolhatjuk. 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	$(1, 1)$	$(1, 2)$	$\cdots$	$(3, 3)$
lapocska	$l_{1,1}$	$l_{1,2}$	$\cdots$	$l_{3,3}$

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

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

A probléma világának egy-egy állapotát egy-egy olyan

Equation 1.10.

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

érték 9-es ($^{(3\times3)}$-as mátrix) határozza meg, melyben az értékek H-beli elemek, és az érték 9-esben minden egyes érték H-ból pontosan egyszer fordul elő:

Equation 1.11.

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

A kezdőállapot és a célállapot:

Equation 1.12.

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

Equation 1.13.

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

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

Equation 1.14.

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

operátort akkor alkalmazhatjuk, ha

Equation 1.15.

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

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

Equation 1.16.

$$l_{i,j}' \rightleftharpoons \begin{cases} 0 & \text{ha } i = s-1,\ j = o \\ l_{s-1,o} & \text{ha } i = s,\ j = o \\ l_{i,j} & \text{egyébként.} \end{cases}$$

Az állapottérgráf

Legyen a p probléma az $\langle \mathcal{A}, kezdő, \mathcal{C}, O \rangle$ 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

  Equation 1.17.

  $$N = \{n_a \mid a \in \mathcal{A}\}.$$

- 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:

  Equation 1.18.

  $$T = \{n_c \mid c \in \mathcal{C}\}.$$

- 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:

  Equation 1.19.

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

Az

Equation 1.20.

$$\langle N, s, T, E \rangle$$

irányított gráfot a p probléma $\langle \mathcal{A}, kezdő, \mathcal{C}, O \rangle$ állapottér-reprezentációjához tartozó *állapottérgráfjának* vagy *reprezentációs gráfjának* nevezzük.

Lemma:

Legyen $\langle N, s, T, E \rangle$ a p probléma $\langle \mathcal{A}, kezdő, \mathcal{C}, O \rangle$ állapottér-reprezentációjához tartozó

állapottérgráfja. Pontosan akkor vezet az állapottérgráf n_a csúcsából az ettől különböző $n_{a'}$ csúcsba irányított út, ha az a állapotból az a' állapot elérhető.

Bizonyítás:

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

   Equation 1.21.

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

   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

   Equation 1.22.

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

   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 állapottérgráf n_a csúcsából az ettől különböző $n_{a'}$ csúcsba irányított út vezet. Ez azt jelenti, hogy van olyan $(n_{a_1}, n_{a_2}), (n_{a_2}, n_{a_3}), \ldots, (n_{a_{k-1}}, n_{a_k}) \in E$ $(k \geq 2)$ irányított élsorozat az állapottérgráfban, 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 a állapotból az a' állapot elérhető.

Tétel:

Legyen $\langle N, s, T, E \rangle$ a p probléma $\langle \mathcal{A}, kezdő, c, o \rangle$ állapottér-reprezentációjához tartozó állapottérgráfja. Pontosan akkor megoldható p, ha van az állapottérgráfban a startcsúcsból valamelyik terminális csúcsba vezető irányított út.

Bizonyítás:

1. Egyrészt ha p megoldható, van olyan $c \in C$ célállapot, hogy $kezdő \Rightarrow^* c$. De ekkor az állapottérgráfban irányított út vezet az $n_{kezdő}$ startcsúcsból az $n_c \in T$ terminális csúcsba.
2. Másrészt ha van az állapottérgáfban 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 $\langle \mathcal{A}, kezdő, c, o \rangle$ állapottér-reprezentációjában vegyük figyelembe az

operátorok alkalmazási költségeit. Rendeljünk ekkor minden $(n_a, n_{a'})$ élhez költséget: ha $a \Rightarrow_o a'$, akkor $\text{költség}(o, a)$ legyen ezen *él költsége* (jelölve: $\text{költség}(n_a, n_{a'})$). Egy

Equation 1.23.

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

irányított út költsége a benne szereplő élek költségösszege

Equation 1.24.

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

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

Egy állapottér-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 állapottér 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

Table of Contents

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:

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

A megoldást kereső rendszerek különböző szempontok alapján *osztályozhatók*:

- Módosítható-e valahogy egy már alkalmazott művelet hatása?
  - nem: nemmódosítható megoldáskeresők
  - igen: módosítható megoldáskeresők
    - visszalépéses (*backtracking*) keresők
    - keresőfával keresők
- Használunk-e valamiféle speciális tudást a keresés során?
  - nem: irányítatlan (vak, szisztematikus) megoldáskeresők
  - igen: heurisztikus megoldáskeresők

    „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 keresünk 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?

Nemmódosítható 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 nemmódosítható keresést az MI területén.

Legyen $p = \langle \mathcal{A}, kezdő, C, O \rangle$. Egy nemmódosítható megoldáskereső rendszer

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

```
procedure Nemmódosítható-Kereső(A, kezdő, C, O)
  aktuális ← kezdő
  while Igaz do
    if aktuális ∈ C then
      break
    end if
    O′ ← {o | o ∈ O ∧ Előfeltétel(aktuális, o)}
    if not O′ = ∅ then
      operátor ← Választ(O′)
      aktuális ← Alkalmaz(aktuális, operátor)
    else
      break
    end if
  end while
  if aktuális ∈ C then
    print aktuális
  else
    print „Sikertelen keresés”
  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 $h : \mathcal{A} \to R^+$ ún. *heurisztikus függvénnyel*, hogy az egyes állapotokból legkevesebb hány operátor alkalmazásával érhetünk célállapotba:

  Equation 2.1.

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

  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

  Equation 2.2.

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

  Azt az $\mathcal{O}'$-beli $operátor$t fogjuk alkalmazni a-ra, amelyik a becslésünk szerint a legközelebb visz valamelyik terminálishoz:

  Equation 2.3.

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

```
procedure Hegymászó-Algoritmus(A, kezdő, C, O )
  aktuális ← kezdő
  while Igaz do
    if aktuális ∈ C then
      break
    end if
    O´ ← {o | o ∈ O ∧ Előfeltétel(aktuális, o) ∧ h(Alkalmaz(aktuális, o)) ≤
h(aktuális)}
    if not O´ = ∅ then
      operátor ← Választ({o | o ∈ O´ ∧ ∀o´(o´ ∈ O´ ⊃ h(Alkalmaz(aktuális, o)) ≤
h(Alkalmaz(aktuális, o´)))})
      aktuális ← Alkalmaz(aktuális, operátor)
    else
      break
    end if
  end while
```

```
  if aktuális ∈ C then
    print aktuális
  else
    print „Sikertelen keresés”
  end if
end procedure
```

A nemmódosítható megoldáskereső rendszerek értékelése:

- *Teljesség*: Nem teljesek.
  - Próba-hiba módszer: Ha olyan kört nem tartalmazó véges állapottérgráfokban 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.

Módosítható megoldáskereső rendszerek

A visszalépéses megoldáskeresés algoritmusa

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

```
procedure Alap-Backtrack-1(A, kezdő, C,O)
  Állapot[aktuális-csomópont] ← kezdő
  Szülő[aktuális-csomópont] ← Nil
  Operátor[aktuális-csomópont] ← *
  Kipróbált[aktuális-csomópont] ← ∅
  while Igaz do
    if aktuális-csomópont = Nil then
      break
    end if
    if Állapot[aktuális-csomópont] ∈ C then
      break
    end if
    O´ ← {o | o ∈ O ∧ Előfeltétel(Állapot[aktuális-csomópont], o) ∧ o ∉
Kipróbált[aktuális-csomópont]}
    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
      Operátor[új] ← operátor
      Kipróbált[új] ← ∅
      aktuális-csomópont ← új
    else
      aktuális-csomópont ← Szülő[aktuális-csomópont]
    end if
  end while
  if notaktuális-csomópont = Nil then
    Megoldás-Kiír(aktuális-csomópont )
  else
    print „Nincs megoldás”
  end if
end procedure

procedure Alap-Backtrack-2(A, kezdő, C,O)
  Állapot[aktuális-csomópont] ← kezdő
  Szülő[aktuális-csomópont] ← Nil
  Operátor[aktuális-csomópont] ← *
  Alkalmazható[aktuális-csomópont] ← {o | o ∈ O ∧ Előfeltétel(Állapot[aktuális-
csomópont], o)}
  while Igaz do
    if aktuális-csomópont = Nil then
      break
    end if
    if Állapot[aktuális-csomópont] ∈ C then
      break
    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)
      Szülő[új] ← aktuális-csomópont
      Operátor[új] ← operátor
      Alkalmazható[új] ← {o | o ∈ O ∧ Előfeltétel(Állapot[új], o)}
      aktuális-csomópont ← új
    else
      aktuális-csomópont ← Szülő[aktuális-csomópont]
    end if
  end while
  if not aktuális-csomópont = Nil then
    Megoldás-Kiír(aktuális-csomópont )
  else
```

```
    print „Nincs megoldás”
  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 $h : \mathcal{A} \to R^+$ heurisztikával, hogy az egyes csúcsok milyen távol vannak a hozzájuk legközelebbi terminális csúcstól. Legyen

  Equation 2.4.

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

  Azt az O'-beli $operátor$t fogjuk alkalmazni a-ra, amelyik a becslésünk szerint a legközelebb visz valamelyik terminálishoz:

  Equation 2.5.

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

Az alap visszalépéses megoldáskeresők 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.

Visszalépéses megoldáskeresés köröket is tartalmazó gráfokban

- *Visszalépéses megoldáskeresés körfigyeléssel:* Ha van megoldás, akkor van körmentes megoldás is. A vezérlő a visszalépést választja, ha az aktuális csúcs szerepelt már az aktuális úton.
- *Visszalépéses megoldáskeresés 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.

```
procedure Körfigyeléses-Backtrack(A, kezdő, C,O)
  Állapot[aktuális-csomópont] ← kezdő
  Szülő[aktuális-csomópont] ← Nil
  Operátor[aktuális-csomópont] ← *
  Alkalmazható[aktuális-csomópont] ← {o | o ∈ O ∧ Előfeltétel(Állapot[aktuális-
csomópont], o)}
  while Igaz do
    if aktuális-csomópont = Nil then
      break
```

```
    end if
    if Állapot[aktuális-csomópont] ∈ C then
      break
    end if
    if VoltMár(Állapot[aktuális-csomópont ]) then
      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}
      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
    else
      aktuális-csomópont ← Szülő[aktuális-csomópont]
    end if
  end while
  if not aktuális-csomópont = Nil then
    Megoldás-Kiír(aktuális-csomópont )
  else
    print „Nincs megoldás”
  end if
end procedure

procedure Úthosszkorlátos-Backtrack(A, kezdő, C,O, korlát )
  Állapot[aktuális-csomópont] ← kezdő
  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
    if aktuális-csomópont = Nil then
      break
    end if
    if Állapot[aktuális-csomópont] ∈ C then
      break
    end if
    if Mélység[aktuális-csomópont] = korlát then
      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}
      Állapot[új] ← Alkalmaz(Állapot[aktuális-csomópont], operátor)
      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
    else
      aktuális-csomópont ← Szülő[aktuális-csomópont]
    end if
  end while
  if not aktuális-csomópont = Nil then
    Megoldás-Kiír(aktuális-csomópont )
  else
    print „Sikertelen keresés”
```

```
  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.

Ág és korlát algoritmus

A backtrack alkalmas optimális (legrövidebb) megoldás keresésére is.

- Egy induló úthosszkorlátnál nem hosszabb megoldást keresünk.
- Ha találunk ilyet, 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 előállítása lehet a cél.

```
procedure Ág-és-Korlát(A, kezdő, C,O, korlát )
  talált ← Hamis
  Állapot[aktuális-csomópont] ← kezdő
  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
    if aktuális-csomópont = Nil then
      break
    end if
    if Állapot[aktuális-csomópont] ∈ C then
      talált ← Igaz
      Megoldás-Feljegyez(aktuális-csomópont )
      korlát ← Mélység[aktuális-csomópont]
```

```
    end if
    if Mélység[aktuális-csomópont] = korlát then
      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}
      Állapot[új] ← Alkalmaz(Állapot[aktuális-csomópont], operátor)
      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
    else
      aktuális-csomópont ← Szülő[aktuális-csomópont]
    end if
  end while
  if talált then
    Megoldás-Kiír
  else
    print „Sikertelen keresés”
  end if
end procedure
```

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.

Keresőfával keresők

Legyen $p = \langle \mathcal{A}, kezdő, c, O \rangle$. 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 $a \in \mathcal{A}$ 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 tartamazó 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.

```
procedure Keresőfával-Kereső(A, kezdő, C,O)
  Állapot[csomópont] ← kezdő
  Szülő[csomópont] ← Nil
  Operátor[csomópont] ← *
  nyíltak ← {csomópont}; zártak ← ∅
  while Igaz do
    if nyíltak = ∅ then
      break
    end if
    csomópont ← Választ(nyíltak)
    if Állapot[csomópont] ∈ C then
      break
    end if
    Kiterjeszt(csomópont, nyíltak, zártak)
  end while
  if not nyíltak = ∅ then
    Megoldás-Kiír(csomópont )
  else
    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.

Szélességi és mélységi keresők

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

Equation 2.6.

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

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.

```
procedure Kiterjeszt(A, kezdő, C,O, 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
        Mélység[új-csomópont] ← Mélység[csomópont] + 1
        nyíltak ← nyíltak ∪ {új-csomópont}
      end if
    end if
  end for
  nyíltak ← nyíltak \ {csomópont}
  zártak ← zártak ∪ {csomópont}
end procedure

procedure Szélességi-Kereső(A, kezdő, C,O)
  Állapot[új-csomópont] ← kezdő
  Szülő[új-csomópont] ← Nil
  Operátor[új-csomópont] ← *
  Mélység[új-csomópont] ← 0
  nyíltak ← {új-csomópont}
  zártak ← ∅
  while Igaz do
    if nyíltak = ∅ then
      break
```

```
    end if
    csomópont ← Választ({cs | cs ∈ nyíltak ∧ ∀cs´(cs´ ∈ nyíltak ⊃ Mélység[cs] ≤
Mélység[cs´])})
    if Állapot[csomópont] ∈ C then
      break
    end if
    Kiterjeszt(A, kezdő, C, O, csomópont, nyíltak, zártak)
  end while
  if not nyíltak = ∅ then
    Megoldás-Kiír(csomópont )
  else
    print „Nincs megoldás”
  end if
end procedure

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

A nyílt csomópontokat gyakran

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

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

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  gyermeke, és  hosszúságú a legrövidebb megoldás. Ekkor a keresőgráf csomópontjainak száma a keresés végére (a legrosszabb esetben):

  Equation 2.7.

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

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.

Optimális kereső

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

Equation 2.8.

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

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

Equation 2.9.

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

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

  Equation 2.10.

  $$útköltség(n) + költség(n, m) < útköltség(m),$$

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

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

  Equation 2.11.

$$útköltség(n) + költség(n, m) > útköltség(m).$$

Az m-hez feltárt új út költségesebb.

3. A célfeltételek vizsgálatát *nem hozhatjuk előre.*

```
procedure Kiterjeszt(A, kezdő, C, O, költség, 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
        Útköltség[új-csomópont]         ←         Útköltség[csomópont]         +
költség(o,Állapot[csomópont])
        nyíltak ← nyíltak ∪ {új-csomópont}
      else if not ny = Nil then
        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
        end if
      end if
    end if
  end for
  nyíltak ← nyíltak \ {csomópont}
  zártak ← zártak ∪ {csomópont}
end procedure

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

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.

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*.

```
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

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
```

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:

  Equation 2.12.

  $$1 + d + d + \ldots + d = O(l \cdot d).$$

Az *A* algoritmus

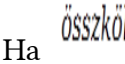
1. A keresőgráf minden csomópontjában megbecsüljük a rajta keresztülhaladó megoldás költségét. Ez egyrészt a csomópontig vezető nyilvántartott út költsége, amihez hozzászámítjuk a célig hátralevő út becsült költségét:

Equation 2.13.

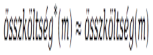
$$összköltség(m) = útköltség(m) + heurisztika(m),$$

azaz

Equation 2.14.

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

Equation 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  csomópont kiterjesztéskor előállt állapot szerepel már a keresőgráf m csomópontjában, és az

Equation 2.16.

$$útköltség(n) + költség(n, m) < útköltség(m),$$

ekkor 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 frissíttetjük a részgráfot;
  - megelőzzük a probléma kialakulását.

3. A célfeltételek vizsgálatát *nem hozhatjuk előre.*

```
procedure Kiterjeszt(A, kezdő, C, O, költség, 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
        Útköltség[új-csomópont]         ←         Útköltség[csomópont]         +
költség(o,Állapot[csomópont])
        Heurisztika[új-csomópont] ← h(állapot)
        nyíltak ← nyíltak ∪ {új-csomópont}
      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 if
  end for
  nyíltak ← nyíltak \ {csomópont}
  zártak ← zártak ∪ {csomópont}
```

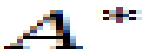
```
end procedure

procedure A-algoritmus(A, kezdő, C, O, költség, h)
  Állapot[új-csomópont] ← kezdő
  Szülő[új-csomópont] ← Nil
  Operátor[új-csomópont] ← *
  Útköltség[új-csomópont] ← 0
  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 ⊃ (Útköltség[cs]
+Heurisztika[cs]) ≤ (Útköltség[cs´]+Heurisztika[cs´]))})
    if Állapot[csomópont] ∈ C then
      break
    end if
    Kiterjeszt(A, kezdő, C, O, költség, h, csomópont, nyíltak, zártak)
  end while
  if not nyíltak = ∅ then
    Megoldás-Kiír(csomópont )
  else
    print „Nincs megoldás”
  end if
end procedure
```

Az *A* algoritmus értékelése

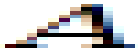
- *Teljesség*: A vezérlő,
  - ha van megoldás, tetszőleges reprezentációs gráfban véges sok keresőlépés után előállít egy megoldást,
  - ha nincs a problémának az adott reprezentációban megoldása, akkor (véges gráf esetén) azt a nyílt csomópontok elfogyásával felismeri.
- *Optimalitás*: Nincs garancia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

  Equation 2.17.

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

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

Lemma:

Az  algoritmus a működése során egy csomópontot legföljebb 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

Equation 2.18.

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

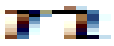
Tegyük föl, hogy a p csúcsba először egy $\text{útköltség}(p)$ költségű úton jutunk el. Ez az út legfeljebb $\text{útköltség}(p)/\delta$ hosszú lehet. Az ennél olcsóbb p-be vezető utak $\text{útköltség}(p)/\delta$-nál biztosan rövidebbek. A $\text{útköltség}(p)/\delta$ korlátnál rövidebb p-be vezető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

Lemma:

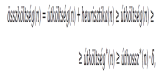
Az A algoritmus, hacsak közben nem fejezi be sikeresen a keresést, minden a nyíltak halmazba bekerülő csomópontot véges sok lépés után kiterjeszt.

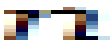
Bizonyítás:

Legyen $p \in \text{nyíltak}$. Megmutatjuk, hogy  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  csomópont bekerül a nyíltak halmazba, akkor

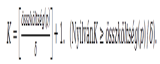
  Equation 2.19.

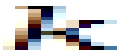
  

  ahol $\text{úthossz}^*(n)$ az  -ből  -be vezető optimális költségű út hossza.

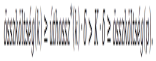
- Most egy mélységi korlátot adunk meg. Legyen

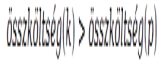
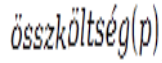
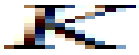
  Equation 2.20.

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

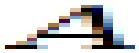
  Equation 2.21.



Tehát  , azaz az  -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  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 $s = n_0, n_1, \ldots, n_k = t$ optimális út.

- *1. kiválasztás előtt*: $s \in nyíltak$.
- *k. kiválasztás előtt*: indukciós feltevésünk szerint $\exists\, j(n_j \in nyíltak)$. 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

Equation 2.22.

$$összköltség(n) \leq összköltség^*(s).$$

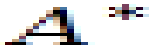
Bizonyítás:

Ha a gráfban nincs megoldás, $összköltség^*(s) = \infty$, egyébként $összköltség^*(s) = útköltség^*(s) + heurisztika^*(s) < \infty$ az optimális megoldás költsége. A korábbi lemma szerint van n kiterjesztése előtt a nyíltak között az optimális úton fekvő csomópont. Legyen m az első ilyen. Ekkor $útköltség(m) = útköltség^*(m)$. Az algoritmus az n csúcsot választotta kiterjesztésre, tehát $összköltség(n) \leq összköltség(m)$. De

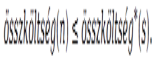
Equation 2.23.

$$összköltség(m) = útköltség^*(m) + heurisztika(m) \leq$$
$$\leq útköltség^*(m) + heurisztika^*(m) = összköltség^*(m) = összköltség^*(s),$$

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

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

Equation 2.24.



Tehát az

Equation 2.25.

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

Equation 2.26.

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

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

Equation 2.27.

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

Definíció:

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

Equation 2.28.

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

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

Equation 2.29.

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

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

Equation 2.30.

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

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

Equation 2.31.

$$\text{összköltség}_Q(n') = \text{időköltség}(n') + \text{heurisztika}_Q(n') <$$

$$< \text{időköltség}(n') + \text{heurisztika}_P(n') = \text{összköltség}_P(n') \leq \text{összköltség}^*(s)$$

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.

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

Equation 2.32.

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

Tétel:

Ha egy heurisztikus függvény kielégíti a monoton megszorítás feltételét, akkor

Equation 2.33.

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

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 $n = n_0, n_1, n_2, \ldots, n_\ell = t \in T$ optimális út. Ennek éleire

   Equation 2.34.

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

   teljesül. Az egyenlőtlenségeket összeadva

   Equation 2.35.

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

   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 $útköltség(n) = útköltség^*(n)$.

Bizonyítás:

Tegyük föl indirekt módon, hogy amikor az n csúcsot kiterjesztésre kiválasztja az algoritmus, $útköltség(n) > útköltség^*(n)$. Legyen m egy olyan nyílt csúcs, amely egy s-ből n-be vezető optimális úton van és amelyre $útköltség(m) = útköltség^*(m)$ 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 $összköltség(n) \leq összköltség(m)$. Ugyanakkor az m-ből n-be vezető $m = m_0, m_1, m_2, \ldots, m_{\ell-1}, m_\ell = n$ optimális útvonalra felírható a következő összefüggés:

Equation 2.36.

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

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

Chapter 3. Kétszemélyes stratégiai játékok és lépésajánló algoritmusok

Table of Contents

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 konkurrencia 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

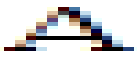
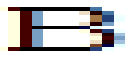
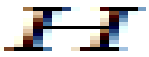
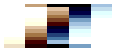
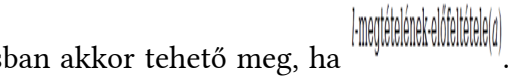
- a játék lehetséges állásait (helyzeteit),
- a játékosok számát,
- hogyan következnek lépni az egyes játékosok (pl. egy időben vagy felváltva egymás után),
- egy-egy állásban a játékosoknak milyen lehetséges lépései (lehetőségei) vannak,
- a játékosok milyen -- a játékkal kapcsolatos -- információval 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.

A játékok reprezentációja

Jelölje a két játékost  és  , a játékállások halmazát  . 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: $\{l \mid l : H \to H\}$. Az  lépés egy  állásban akkor tehető meg, ha  . A játék az  állásban véget

ér, ha $\text{végállás}(a)$. A szabályok leírják, itt ki a nyerő játékos: $nyer : \{a \mid \text{végállás}(a)\} \to \{\text{jön}_a, \text{volt}_a\}$, ahol jön_a lenne az [illegible] állásban soron következő játékos ($\text{jön}_a \neq \text{volt}_a$). E játék állapottér-reprezentációja az az $\langle \mathcal{A}, kezdő, \mathcal{V}, \mathcal{O} \rangle$ négyes, ahol

- $\mathcal{A} = \{(a, J) \mid a \in H, J \in \{A, B\}, J \text{ következik lépni}\}$,
- $kezdő = (a_0, J_0)$
- $\mathcal{V} = \{(a, J) \mid \text{végállás}(a), J \text{ nyer, ha } nyer(a) = \text{jön}_a\}$
- $\mathcal{O} = \{o_l \mid o_l(a, J) = (l(a), I), I, J \in \{A, B\}, I \neq J\}$

A játék állapottér-repezentációját szemléltető gráf a *játékgráf.* „Egyenesítsük ki” a játékgráfot fává. A *játékfában*

- páros szinteken lévő állásokban a kezdő játékos, páratlan szinteken lévőkben 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 $kezdő \xRightarrow[o_1, \ldots, o_r]{*} v \quad (r \geq 1)$, azt mondjuk lejátszottak egy *játszmát.* A játszmákat a játékfában a startcsúcsból a levélelemekbe vezető utak szemléltetik. Egy játék játékfája a játék összes lehetséges játszmáját szemlélteti a startcsúcsból induló, a különböző levelekben végződő útjaival.

Definíció

Az $\langle N, HE \rangle$ 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

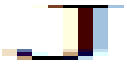
Az ÉS/VAGY gráfban a gráf hiperéleinek egy olyan
$$\begin{gathered}(n_1, \{n_{11}, n_{12}, \ldots, n_{1k_1}\}),\\ (n_2, \{n_{21}, n_{22}, \ldots, n_{2k_2}\}),\\ \vdots\\ (n_r, \{n_{r1}, n_{r2}, \ldots, n_{rk_r}\})\end{gathered}$$
sorozata, ahol

$$\forall i \forall j\left(\neg(i=j) \supset \neg(n_i = n_j)\right) \wedge$$

$$\wedge \forall i\left((i>1) \supset \exists j\left((i>j) \wedge (n_i \in \left\{n_{j1}, n_{j2}, \ldots, n_{jk_j}\right\})\right)\right),$$ a gráf egy *hiperútja.*

A stratégia

Definíció

A  játékos *stratégiája* egy olyan

Equation 3.1.

$$s_J : \{(a,j) \mid (a,j) \in A\} \to 0$$

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 kinduló 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 közönséges 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 \textbf{ 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 levélelemei mind J-nyerő állások.

Tétel

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

Bizpnyítás

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.

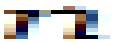
Mivel a játékfa véges, végül a startcsúcs is címkét kap. A teljes indukció elve alapján ez a címke egyértelmű. A startcsúcs címkéje pedig megmutatja, melyik játékosnak van nyerő stratégiája, és magát a nyerő stratégiá(ka)t is le lehet olvasni a felcímkézett játékfáról.

Minimax algoritmus

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

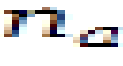
- a játék $\langle \mathcal{A}, kezdő, V, O \rangle$reprezentációját,
- J azon a állását, ahol lépni következik,
- az állások „jóságát” J szempontjából becslő $h_J : \mathcal{A} \to R$ heurisztikát
- és egy mélységi $korlát$ -ot.

Az algoritmus fő lépései

1. A játékfa (a, J) állapotot szemléltető csúcsából kiinduló részének előállítása $korlát$ 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  csúcs gyermekei rendre $n_1, \ldots, n_k$, akkor

   Equation 3.2.

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

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

```
function Minimax-lépés(A, kezdő, V, O, állapot, korlát, hJ)
  max ← −∞
  operátor ← Nil
  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
        max ← v
        operátor ← o
      end if
    end if
  end for
  return operátor
end function

function Minimax-Érték(A, kezdő, V, O, állapot, mélység, hJ)
  if állapot ∈ V or mélység = 0 then
    return hJ(állapot)
  else if Játékos[állapot] = J then
    max ← −∞
    for all o ∈ O do
      if Előfeltétel(állapot, o) then
        új-állapot ← Alkalmaz(állapot, o)
        v ← Minimax-Érték(A, kezdő, V, O, új-állapot,mélység − 1, hJ)
        if v > max then
          max ← v
        end if
      end if
    end for
    return max
  else
    min ← ∞
    for all o ∈ O do
      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
        end if
      end if
    end for
    return min
  end if
end function
```

Negamax algoritmus

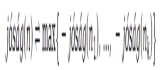
Cél: a támogatott játékosnak, J-nek, egy adott állásban „elég jó” lépést ajánlani. Az algoritmus számára át kell adni

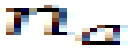
- a játék $\langle \mathcal{A}, kezdő, V, O \rangle$ reprezentációját,
- J azon a állását, ahol lépni következik,
- az állások „jóságát” a soron következő játékos szempontjából becslő $h : \mathcal{A} \to R$ heurisztikát
- és egy mélységi $korlát$-ot.

Az algoritmus fő lépései:

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

   Equation 3.3.

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

```
function Negamax-lépés(A, kezdő, V, O, állapot, korlát, h)
  max ← −∞
  operátor ← Nil
  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
        max ← v
        operátor ← o
      end if
    end if
  end for
  return operátor
end function

function Negamax-Érték(A, kezdő, V, O, állapot, mélység, h)
  if állapot ∈ V or mélység = 0 then
    return h(állapot)
  else
    max ← −∞
    for all o ∈ O do
      if Előfeltétel(állapot, o) then
        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
    return max
  end if
end function
```

Chapter 4. Problémamegoldás redukcióval

Table of Contents

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 egyszerűségüknél fogva már könnyedén meg tudunk oldani. A probléma megoldásnak ezt a módját *problémaredukciónak* nevezzük.

Problémaredukciós reprezentáció

- Először is le kell írni az eredeti problémát, jelöljük ezt most 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 $p \in \mathcal{P}$.
- $\mathcal{P}$ 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 $p \notin \mathcal{E}$, 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

  Equation 4.1.

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

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

Definíció:

Legyen p egy probléma. Azt mondjuk, hogy a p problémát *problémaredukciós reprezentációval*

írtuk le, ha megadtuk a $\langle \mathcal{P}, p, \mathcal{E}, \mathcal{R} \rangle$ négyest, azaz

- a megoldandó $p \in \mathcal{P}$ problémát,
- a $\mathcal{P} \neq \emptyset$ halmazt, a 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

Equation 4.2.

$$Q = \{p_1, p_2, \ldots, p_i, \ldots, p_n\} \subseteq \mathcal{P}$$

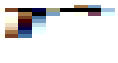
Equation 4.3.

$$Q' = \{p'_1, p'_2, \ldots, p'_j, \ldots, p'_m\} \subseteq \mathcal{P}$$

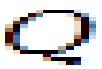
egy-egy problémahalmaz ($n \geq 1, m \geq 1$). Azt mondjuk, hogy a  problémahalmaz *egy lépésben* vagy *közvetlenül redukálható* a  problémahalmazzá, ha van olyan $r \in \mathcal{R}$ redukciós operátor, melyre $p_i \in \mathrm{Dom}(r)$, és

Equation 4.4.

$$r(p_i) = \{q_1, q_2, \ldots, q_m\}.$$

Ennek jelölése: $Q \lhd Q'$, illetve ha fontos, hogy az  redukciós operátor segítségével állítottul elő  -ból a  -t, akkor $Q \lhd_r Q'$.

Definíció:

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

Equation 4.5.

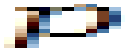
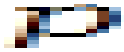
$$P_1 = Q, \quad P_k = Q'$$

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

Definíció:

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

Definíció:

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

Equation 4.6.

$$\{p\} \lessdot^*_{r_1,\ldots,r_l} Q \subseteq \mathcal{E}.$$

Ekkor az $r_1, \ldots, r_l$ 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 előállítása,
- valamilyen minősítés alapján jó megoldás előállítása (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 $\langle \mathcal{P}, p, \mathcal{E}, \mathcal{R} \rangle$ problémaredukciós reprezentációban $g(q)$ a $q \in \mathcal{P}$ probléma *megoldásának minimális költsége*, ha

Equation 4.7.

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

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 \textbf{alkalmazásának a végrehajtási idejét}, $c(q)$ pedig a q egyszerű probléma közvetlen megoldásának idejét jelenti.

Definíció:

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

Equation 4.8.

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

Példák problémaredukciós reprezentációra

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 rárakva úgy, hogy minden korong alatt egy nála nagyobb volt. A szerzeteseknek az a feladatuk, hogy a korongokat helyezzék 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!

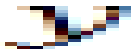
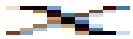
1. lépés: Át kell vinni az első rúdon lévő felső 63 korongból álló tornyot a második rúdra.
2. lépés: Át kell vinni az első rúdon lévő utolsó, legnagyobb korongot a harmadik rúdra.
3. lépés: Át kell vinni a második rúdon lévő 63 korongból álló tornyot a harmadik rúdra.

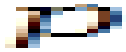
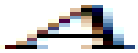
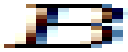
A szerzetes másnap kiszögezte a templom kapujára az algoritmus leírását:

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 szóban 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 szóban forgó torony felső $n-1$ korongját vigye át a z rúdról az*  *-ra, miközben az*  *-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  feladat tehát: mindhárom az  rúdon levő korong átvitele a C rúdra ( felhasználásával). Ezt jelölhetjük a következőképpen:

Equation 4.9.

$$p = (3;\ A \to C)$$

A megoldandó feladathoz hasonló feladatok a következők: az x rúdon levő felső valahány korong átvitele a y rúdra (z felhasználásával):

Equation 4.10.

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

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 eredetinél egyszerűbb részre bonthatunk:

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

  Equation 4.11.

  $$N = \{n_q \mid q \in \mathcal{P}\}.$$

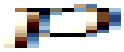
- A gráf csúcsai közül kitüntetett szerepet játszanak a p problémát szemléltető ún. *startcsúcs* (jele: n_p vagy s)

- és az egyszerű problémákat szemléltető *terminális csúcsokí*. A terminális csúcsok halmaza tehát:

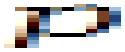
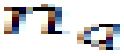
  Equation 4.12.

  $$T = \{n_e \mid e \in \mathcal{E}\}.$$

- Egy $q \in \mathcal{P}$ problémát szemléltető csúcsból *irányított éleket* húzunk az $q_1, \ldots, q_m \in \mathcal{P}$ problémákat szemléltető $n_{q1}, \ldots, n_{qm}$ csúcsokba, amikor $\{q\} \lhd \{q_1, q_2, \ldots, q_m\}$. Ezek az élek összetartozónak tekinthetők: egy *ÉS élköteget* vagy *hiperélt* alkotnak. A gráf hiperéleinek halmaza tehát a következő:

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

Lemma

Legyen $\langle N, s, T, E \rangle$ a  probléma $\langle \mathcal{P}, p, \mathcal{E}, \mathcal{R} \rangle$ problémaredukciós reprezentációjához tartozó reprezentációs gráfja. Pontosan akkor áll fenn a $\{q\} \lhd^* \{q_1, q_2, \ldots, q_m\}$ reláció, ha a reprezentációs gráfban van az  csúcsából induló olyan hiperút, melynek levelei éppen az $\{n_{q1}, n_{q2}, \ldots, n_{qm}\}$ csúcsok.

Bizonyítás

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

   Equation 4.13.

   $$P_1 = \{q\}, \quad P_k = \{q_1, q_2, \ldots, q_m\}$$

   és

   Equation 4.14.

   $$P_i \lhd P_{i+1}$$

   minden $1 \leq i \leq k-1$ esetén.

   - Tehát minden $1 \leq i \leq k-1$-re P_i valamelyik problémája, mondjuk p_{ij} 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} 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 indul olyan hiperút, melynek levelei az $\{n_{q_1}, n_{q_2}, \ldots, n_{q_m}\}$ csúcsok. Ez azt jelenti, hogy van olyan $(n_1, \{n_{11}, n_{12}, \ldots, n_{1m_1}\})$,

$(n_2, \{n_{21}, n_{22}, \ldots, n_{2m_2}\})$,

$\vdots$

$(n_k, \{n_{k1}, n_{k2}, \ldots, n_{km_k}\})$ hiperélsorozat a reprezentációs gráfban, hogy

Equation 4.15.

$$n_q = n_1,$$

továbbá

Equation 4.16.

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

és

Equation 4.17.

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

A sorozat minden $(n_i, \{n_{i1}, n_{i2}, \ldots, n_{im_i}\})$ hiperéle egy redukciós operátoralkalmazást szemléltet: az n_i által szepléltetett problémát bontja a redukciós operátor az $\{n_{i1}, n_{i2}, \ldots, n_{im_i}\}$ 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.

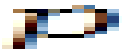
Tétel

Legyen $\langle N, s, T, E \rangle$ a p probléma $\langle \mathcal{P}, p, \mathcal{E}, \mathcal{R} \rangle$ problémaredukciós reprezentációjához tartozó reprezentációs gráfja. Pontosan akkor oldható meg p, ha van a reprezentációs gráfban a

startcsúcsból induló olyan hiperút, melynek levelei terminális csúcsok.

Problémaredukcióval reprezentált feladatok megoldáskereső módszerei

Visszalépéses megoldáskeresés ÉS/VAGY fák esetén

Legyen $(\mathcal{P}, p, \mathcal{E}, \mathcal{R})$ a  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, mellyet q-ra aktuálisan alkalmaztunk;
  - q-ra a keresés során már alkalmazott (vagy még alkalmazható) redukciós operátorok halmazát.
- A visszalépéses megoldáskeresők 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.

Keresőfával megoldáskeresés ÉS/VAGY fák esetén

Legyen $\langle \mathcal{P}, p, \mathcal{E}, \mathcal{R} \rangle$ a 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 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 levélcsomópontján keresztül kibővíti.
  - alkalmazási előfeltétele: a keresőfában van *folyamatban* címkéjű levélcsomópont.
  - hatása:
    - alkalmazzuk az összes alkalmazható redukciós operátort a *folyamatban* címkéjű levélcsomópont problémájára,
    - az előálló problémákat új csomópontokként felfűzzük a keresőfába megfelelő címkékkel:
      - *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 levélcsomópont 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.