Acta Agraria Kaposváriensis (2008) Vol 12 No 2, 53-58
Kaposvári Egyetem, Állattudományi Kar, Kaposvár
University of Kaposvár, Faculty of Animal Science, Kaposvár

Számítógépes szimulációs példák néhány keresési módszer oktatásához a mesterséges intelligencia területén

Kaczur S.
Gábor Dénes Főiskola, Informatikai Intézet, 1115 Budapest, Etele út 68.

ÖSSZEFOGLALÁS

A műszaki tárgyak, különösen az informatika oktatása területén számos esetben előtérbe kerül a számítógépes szimuláció, mint a szemléltetés egyik látványos módszere. A Gábor Dénes Főiskolán a Mesterséges intelligencia tantárgy keretében több szakon is folyik többféle számítógépes szimulációval támogatható feladat, probléma, módszer ismertetése, oktatása. Egy mesterséges intelligencia probléma megértése, a megoldáshoz vezető állapottér reprezentációja, döntéshozatal az aktuális állapottól függően, összetett probléma a hallgatók számára. Elkészítésre kerül egy grafikus felhasználói felülettel rendelkező, felhasználóbarát módon működő szoftvert, amely többféle szimulációs algoritmus bemutatására alkalmas. A szoftver előadáson és gyakorlaton történő alkalmazásán túl – megvalósulása után – szabadon letölthető a főiskola ILIAS nevű tananyagfejlesztő és távoktató e-learning keretrendszeréből, így a hallgatók egyénileg kipróbálhatják, tesztelhetik, megérthetik annak működését. A cikk ismerteti a szoftver tervezésének, megvalósításának lépéseit.
(Kulcsszavak: Programozás, oktatás, szimuláció, e-learning, mesterséges intelligencia)

ABSTRACT

Computer simulation examples for teaching of searching methods in the section of artifical intelligence

S. Kaczur
Dennis Gabor Applied University, Institute of Informatics, H-1115 Budapest, Etele út 68.

Solving problems by computer simulation is an important part of courses related to technology, especially in the case of information technology. Computer simulation also provides efficient possibilities for demonstration aims. I teached Artifical Intelligence in Dennis Gabor Applied University. In order to familiarize students more extensively with the methodology of subject the examples of the practical problems are illustrated and solved by computer simulation. Understanding AI problems, setting up the state space and making decision by the instantaneous states are difficult problems for the students. I develop a software with graphical user interface to simulate and demonstrate some algorithms. This software will be used in our seminars and trainings and will have been downloaded from ILIAS. ILIAS is our open source web-based learning management system. It is very useful for students to have possibility to learn themselves and to test their knowledge. Development steps of this software are shown in this paper.
(Keywords: programming, teaching, simulation, e-learning, artifical intelligence)

A SZIMULÁCIÓ

A szemléltetés legsokoldalúbb eszköze a szimuláció. Olyan esetekben célszerű használni, amikor:

- a valós kísérletek elvégzése rendkívüli költségekkel járna (pl.: űrkutatási kísérlet),
- a valós kísérletek produkálása felmérhetően/felmérhetetlenül nagy veszélyt jelentene (pl.: különböző robbantási kísérlet),
- a vizsgálandó/bemutatandó esemény nagyon ritkán fordul elő a természetben (pl.: szivárvány) stb.

A szimuláció elvitathatatlan előnye, hogy megállítható, képenként/mozzanatonként mutatható be a jelenség. Az egyes képi megjelenítéseket pedig szöveges magyarázattal is el lehet látni.

A szimuláció megvalósításának elengedhetetlen feltétele, hogy a folyamatot a szimulációnak megfelelő mélységben ismerjük (pl.: a szivárvány szimulációjához elég a fénytörés fizikáját ismerni, nincs szükség a fény részecsketulajdonságainak ismeretére) (*Horváth és mtsai*, 1995).

A JÁTÉK

A vizsgált témában a „Tili-toli" nevű logikai játék, vagy más néven „8-as játék" állapotterét kell ismerni ahhoz, hogy a játék menetét a hozzá kapcsoló *Mesterséges intelligencia* című tárgy oktatási anyagaként be lehessen mutatni.

A „8-as játék" állapottér-reprezentációja

A 8-as játék probléma közismert.

A feladat állapottere a 9 négyzetből adódó permutációk alapján 9!=362880. A startállapotban az egytől nyolcig számozott és az üres négyzet tetszőlegesen helyezkedhet el egy 3×3-as táblán.

A többféleképpen megadható célállapot csak a tetszőleges kiinduló állapotok feléből érhető el – mindez a startállapotra jellemző inverziószám alapján eldönthető. Legyen egy startállapot sorfolytonosan leírva az alábbiak szerint (az üres négyzetet nullával jelölve): 308456172 (*Szalay*, 2004).

E 9 elemű halmaz permutációjában két elem inverzióban áll, ha közülük a nagyobbik megelőzi a kisebbiket: 3-0, 3-1, 3-2, 8-4, 8-5, 8-6, 8-1, 8-7, 8-2, 4-1, 4-2, 5-1, 5-2, 6-1, 6-2, 7-2. Ez a permutáció páros, inverziószáma 16, azaz páros. Ebből az következik, hogy az 123804765 rögzített célállapot elérhető, azaz a feladatnak van megoldása. A megoldáson az egymást követő véges sok lépés (művelet) sorozatát értjük, azaz az állapottér elemeit. Egy lépés azt jelenti, hogy az üres négyzet helyet cserél egyik szomszédjával (*Russel és Norvig*, 2000).

Más megközelítésben: ha a játékot valós formájában játsszuk, akkor a négyzetek csak tologathatók, nem eshetnek ki a táblából. Ha az általunk tekintett célállapotból kiindulva bármilyen állapotot előállítottunk, akkor azt startállapotnak tekintve, az eredeti célállapot elérhető. Ha a táblából kiesnek a négyzetek, akkor véletlenszerűen előállított startállapotok közül csak az esetek felében kaphatjuk meg tologatással a célállapotot.

A művelet jelentse az üres hely mozgatását a négy irány közül (balra, felfelé, jobbra, lefelé) mindig csak a lehetséges irányokba, előre rögzített sorrendben. Mátrix-reprezentációban a művelet során a nullát felcseréljük egy másik számmal. A műveletek

költségét egységnek tekintjük, a megoldhatóságra és a lépésszámra helyezzük a hangsúlyt (*Sántáné-Tóth*, 1998).

Két – a 8-as játék megoldására alkalmas – algoritmust kívánunk bemutatni, több változatban. Mindkettő a nemmódosítható vezérlési stratégiák közé tartozik. Mindkettő heurisztikusnak tekinthető, mivel a célállapot eléréséhez szükséges műveletek számát becsüli. Egyik sem biztosít optimális megoldást, de még a megoldást sem garantálják. A páros inverziószámú startállapotot mindkét esetben feltételezzük.

A W függvény minden állapotban a célállapothoz képest nem a helyükön lévő négyzetek számának ellentettje. Alkalmazása során arra kell figyelni, hogy a függvény értéke állapotról-állapotra növekedjen, nem feltétlenül monoton módon. Ha ad megoldást, akkor a célállapotban a függvény értéke 0. Több lehetséges művelet esetén azt választjuk, amely nagyobb növekedést okoz, azaz közelebb visz a célállapothoz, vagy egyenlő értékek esetén a maximum négy irány rögzített sorrendje dönt (*Fekete és mtsai*., 1999).

Két esetben nincs megoldás:

Ha a függvénynek helyi maximuma van, vagyis bármely további művelet csökkenti a függvényértéket, azaz tovább rontja az aktuális állapotot.

A függvényértékek ekvidisztáns felületet alkotnak, azaz egy adott állapotban bármely további művelet végrehajtása után megegyező függvényértékű állapotba kerülnénk.

A P függvény minden állapotban a négyzetek célhelyétől mért távolság összegének ellentettje. Alkalmazása során szintén nem csökkenhet a függvény értéke.

A PROGRAM

A program adatszerkezetei közül az `AllapotTomb` -2-től 255-ig indexelt `TAllapot` típusú rekordokból álló tömb. A `TAllapot` rekord `Tabla` tömbje a 3x3-as kétdimenziós tömb, amely az összes állapotot (játékállást) tárolja a start- és célállapot között. Sor- és oszlopindexei 1-től 3-ig futnak. Az `AllapotTomb` néhány indexe: `csere`=-2, `cel`=-1, `start`=0, végül a `lepes` az aktuális index 1-től 255-ig.

Mindkét függvény esetén a program legfeljebb 255 lépésben próbálkozik megoldást találni, és észreveszi, ha a startállapot éppen a célállapottal egyezik meg.

A `Wfuggveny` nevű függvény az üres négyzet helyzetét figyelmen kívül hagyva, a `db` változó segítségével megszámlálja, hogy a `Tabla[i,j]`-edik elemei hány helyen térnek el a `lepes` helyzetben a célállapothoz képest. Végül visszaadja a kapott érték ellentettjét.

```
function Wfuggveny (lepes: Integer): ShortInt; //állapotkiértékelő
függvény
  var
    i, j,
    db: Byte;  //az aktuális állapotban (lepes) hány négyzet nincs a
helyén
  begin
    db:=0;
    for i:=1 to 3 do
      for j:=1 to 3 do
        if (AllapotTomb[lepes].Tabla[i,j]<>AllapotTomb[cel].Tabla[i,j])
            and (AllapotTomb[lepes].Tabla[i,j]<>0) then
          inc(db);                                  //az üres nem számít
    Wfuggveny:=(-1)*db;
  end;
```

A W függvény képlete:

$$\forall (i,j) \in [1;3], \quad AllapotTomb[lepes].Tabla[i,j] <> 0 \quad \Rightarrow \quad W : N \to R_0^-,$$
$$W(lepes) := \sum_{i=1}^{3}\sum_{j=1}^{3} db(AllapotTomb[lepes].Tabla[i,j] <> AllapotTomb[cel].Tabla[i,j]) \tag{1}$$

A `Pfuggveny` nevű függvény az üres négyzet helyzetét figyelmen kívül hagyva, a hanyhely változót felhasználva az osszeg változóba összegyűjti, hogy a `Tabla[i,j]`-edik elemei közül az egyes négyzetek mekkora távolságra (mozgatásnyira) vannak a lepes helyzetben a célállapottól. Végül visszaadja a kapott érték ellentettjét. Ez a koordináta-rendszerben két pont Manhattan-távolságát jelenti (háztömb). A két külső ciklus az aktuális `Tabla[i,j]`-edik elemét vizsgálja. A két belső ciklus megkeresi az aktuális pozícióban lévő elem célállapotban lévő helyét: `celi`, `celj`. Az (`i,j`) és a (`celi, celj`) koordináták Manhattan-távolságát a hanyhely változó tartalmazza.

```
function Pfuggveny (lepes: Integer): ShortInt;  //állapotkiértékelő
függvény
  var  //Manhattan-távolság
    i, j, k, l,
    osszeg, //az akt. áll.-ban a négyzetek hány helynyire vannak a céltól
    hanyhely, celi, celj, ertek: Byte;
  begin
    osszeg:=0;
    for i:=1 to 3 do
      for j:=1 to 3 do
        begin
          ertek:=AllapotTomb[lepes].Tabla[i,j];
          if ertek<>0 then  //az üres nem számít
            begin
              for k:=1 to 3 do //keresi a négyzet céláll.-ban lévő helyét
                for l:=1 to 3 do
                  if ertek=AllapotTomb[cel].Tabla[k,l] then
                    begin
                      celi:=k;
                      celj:=l;
                    end;
              hanyhely:=Abs(i-celi)+Abs(j-celj);
              //ennyit kell mozgatni a négyzetet, hogy a helyére kerüljön
              inc(osszeg,hanyhely);
            end;
        end;
    Pfuggveny:=(-1)*osszeg;
  end;
```

A P függvény képlete:

$$\forall (i,j) \in [1;3], \quad AllapotTomb[lepes].Tabla[i,j] <> 0 \quad \Rightarrow \quad P : N \to R_0^-,$$
$$P(lepes) := \sum_{i=1}^{3}\sum_{j=1}^{3} hanyhely((i,j),(celi,celj)), \quad ahol\ hanyhely = |i-celi| + |j-celj| \tag{2}$$

AZ EREDMÉNY ÉS HELYE AZ OKTATÁSBAN

A programot elindítva, majd algoritmust és lehetőséget választva az *1. ábrát* kapjuk eredményül. Az *Indít* gombra kattintva legenerálódik az állapottér, és a fülek segítségével bármelyik állapot megtekinthető. A *Start* fül a startállapotot tartalmazza. A kiválasztott függvény aktuális értéke mellett az aktuális állapotban lehetséges lépéseket is láthatjuk. Továbblépéshez szükséges döntést akkor lehet hozni, ha választhatunk a számunkra lokálisan vagy globálisan kedvező lehetőségek közül. Ha a célállapot nem érhető el, akkor nem jelenik meg a *Cél* fül, és az utolsó fülön kiderül, miért állt le az algoritmus.

1. ábra

Egy lehetséges startállapot a W függvényhez

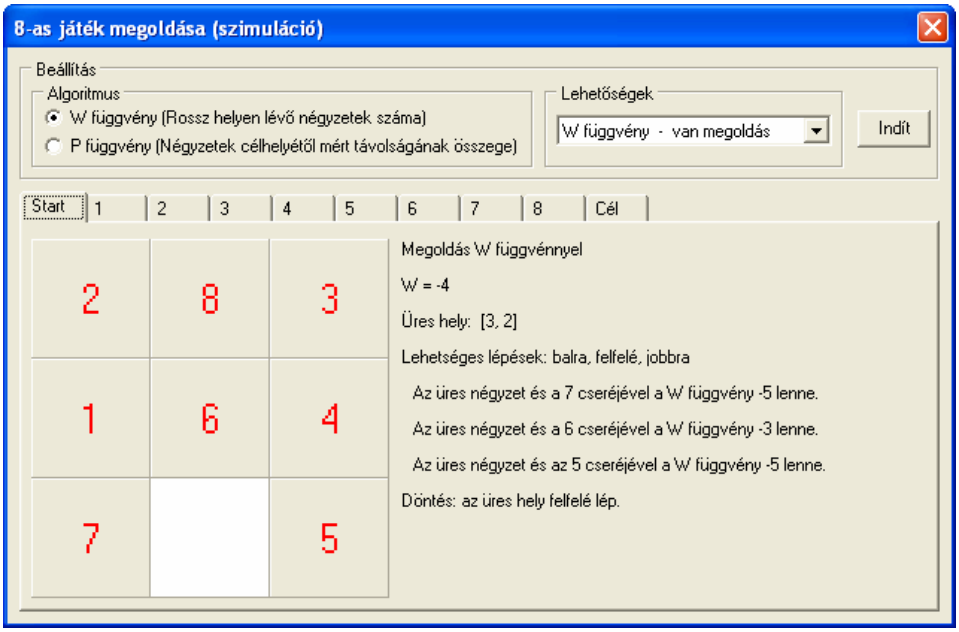

Figure 1. Optional start state for function W

A 8-as játék megoldására többféle heurisztika alkalmazható. A heurisztika olyan ötlet, tanács, amely gyakran hatékonyan alkalmazható, ám nem mindig érvényes. A feladatok megoldása során alkalmazható heurisztikus kiértékelő függvények a probléma egy-egy állapotához egy adott számot rendel.

Ha a W függvénynél figyelünk arra, hogy az egymást követő lépések ne legyenek egymás inverzei, akkor ezzel az állapottér gráfjában elkerültük a kettő hosszúságú köröket. Másképpen: ha nem engedjük meg az oda-vissza lépést, akkor így nem rekedhet meg az algoritmus. A P függvénynél erre – jellegéből adódóan – nincs szükség.

Ennél hosszabb körök is előfordulhatnak. Ezek figyelése összetett feltételeket kívánna. Ehelyett A* algoritmus (többféle függvénnyel), vagy visszalépéses algoritmus

alkalmazása célravezetőbb lenne. Mindez ötletet adhat az elkészült szoftver továbbfejlesztéséhez (*Cormen et al*., 2003).

A szoftver fejlesztése során lényeges szempont volt a felhasználóbarát grafikus felület, illetve az egyszerű kezelhetőség. A rendelkezésre álló fejlesztőeszközök közül a Delphi 7.0-ra esett a választás, többek között azért, mert képes olyan önállóan futtatható exe fájlt készíteni, amelyet telepíteni sem kell. Az elkészült oktatóprogram eleget tesz a követelményeknek (*Bernát*, 2003). Tantárgyhoz kapcsolódik, egy anyagrészének megértését segíti, megmutatja milyen alkalmazási lehetőségek adódnak, szimulációs lépéseken át a teljes probléma lépésről-lépésre megtekinthető, így következtetéseket is lehet levonni. Az előzetes tapasztalatok kedvezőek.

Jelenleg a Gábor Dénes Főiskola ILIAS keretrendszerében (http://ilias.gdf.hu) a *Mesterséges intelligencia* tantárgyhoz tartozó tantárgyi kezdőlapon megtalálható a tantárgyi útmutató, tantárgyleírás, MI fogalomtár, 42 kérdésből álló önellenőrző teszt. A tantárgyhoz tartozó fórum igen aktív. A tantárgyi csomaghoz tartozó tankönyv, példatár, előadásvázlat PDF formátumban letölthető. Néhány oktatóprogram is rendelkezésre áll, kipróbálható. A tantárgy kötelező a műszaki informatika és informatikus közgazdász szakos hallgatók számára. A nappali és távoktatás tagozaton jelentős eltérés mutatkozik a kontaktórák számában. Az említett tananyagok a tanévkezdéskor a hallgatók számára kiosztott *Hallgatói DVD*-n is elérhetők.

IRODALOM

Bernát L. (2003): Az oktatóprogram készítés egy hatékony alternatívája, In: Acta Agraria Debreceniensis, 12. 1-6. p.

Cormen T.H., Leiserson C.E., Rivest R.L. (2003): Algoritmusok, Műszaki Könyvkiadó : Budapest 884. p.

Fekete I., Gregorics T., Nagy S. (1999): Bevezetés a mesterséges intelligenciába, LSI Oktatóközpont : Budapest 293. p.

Horváth L., Szlávi P., Zsakó L. (1995): Modellezés és szimuláció, ELTE : Budapest 109. p.

Sántáné-Tóth E. (1998): Tudásalapú technológia, szakértő rendszerek, Miskolci Egyetem Dunaújvárosi Főiskolai Kar Kiadói Hivatala : Dunaújváros 298. p.

Russel S.J., Norvig P. (2000): Mesterséges intelligencia modern megközelítésben, Panem Könyvkiadó : Budapest 1093. p.

Szalay T. (2004): A mesterséges intelligencia alapjai példatár, LSI Informatikai Oktatóközpont : Budapest 74. p.

Levelezési cím (*Corresponding author*):

Kaczur Sándor
Gábor Dénes Főiskola, Informatikai Intézet
1115 Budapest, Etele út 68.
Dennis Gabor Applied University, Institute of Informatics
H-1115 Budapest, Etele út 68.
Tel.: 36-1-203-0304/5220
e-mail: kaczur@gdf.hu