Neuronhálók fejlesztése genetikus algoritmusokkal

Rövid áttekintő

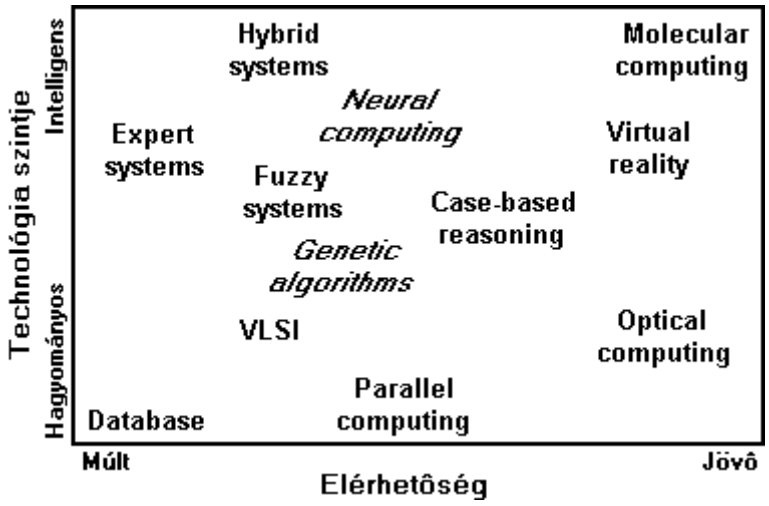
Bódis Lóránt
Babeş-Bolyai Tudományegyetem
Matematika és Informatika Kar
http://linux.scs.ubbcluj.ro/~bl50050
bl50050@scs.ubbcluj.ro

2003 június 4

1. Bevezetés

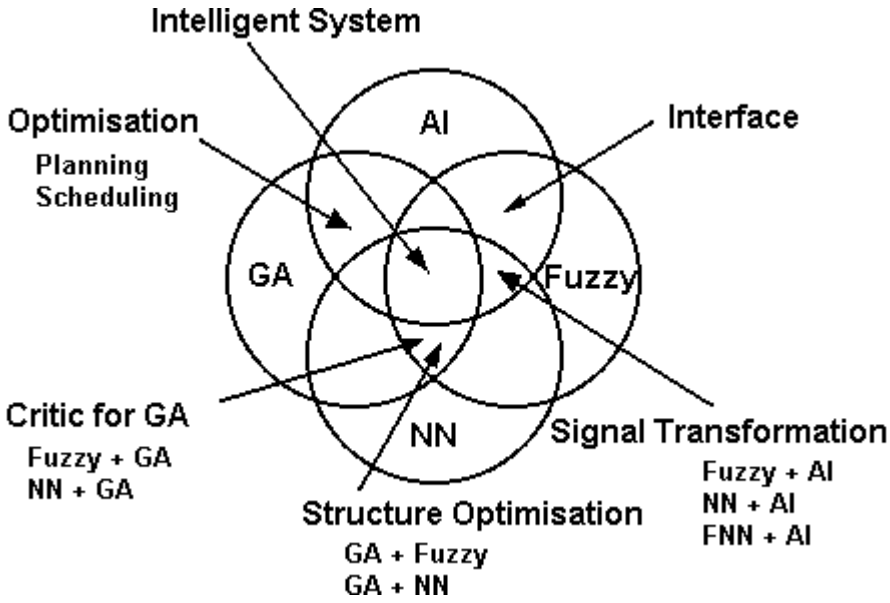
Ebben a dolgozatban a mesterséges intelligencia – röviden MI (angolul ***A**rtificial **I**ntelligence – AI*) – két nagyon népszerű területébe – evolúciós programozás és mesterséges neuronhálók – nyerünk betekintést, sajátos módon ötvözve a kettőt, felhasználva az eddigi kutatások eredményeit. Az utóbbi évtizedben a számítógépek fejlődésével nagy érdeklődés mutatkozott e területek iránt, hiszen ezek roppant számításigényes feladatok.

Miután a tudósok kiábrándultak a klasszikus és neoklasszikus módszerekből, amelyeket a mesterséges intelligencia modellezésére használtak, más irány felé fordultak. Két kiemelkedő terület jelent meg: a kapcsolati modell (*connectionism*), amelyik a neuronhálókkal és párhuzamos feldolgozással foglalkozik illetve az evolúciós számítások (*evolutionary computing*). Az 1. ábra szemlélteti az idők során kialakult MI technológiákat, kutatási területeket. Jól látszik, hogy a genetikus algoritmusok és a neuronhálók a közelmúltban fejlődtek ki és viszonylag intelligens rendszereket képviselnek. A genetikus algoritmusok központi szerepet töltenek be, hiszen nagyon sok technológia erre a módszerre épít. Az MI területén folytatott kutatások azt bizonyítják, hogy a jövő intelligens rendszerei a molekuláris számítások irányába mozdulnak el.



1. ábra. A neuronhálók és genetikus algoritmusok által elfoglalt hely az MI technológiák között.

Nehéz, azaz komplex feladatok optimalizálásához sokszor szükséges különböző módszerek összekapcsolása. A neuronhálók napjaink leghatékonyabb és legelterjedtebb gépi tanulási módszere, a genetikus algoritmusok pedig robusztuságukkal jól bevált globális optimalizálóknak bizonyultak az elmúlt évtizedekben. A 2. ábra a legfontosabb MI technológiák közötti kölcsönhatást szemlélteti, megjelölve a lehetséges alkalmazási területeket is.



2. ábra. A különböző MI technológiák ötvözése. Az eredmény az intelligens rendszerek.

2. Mesterséges neuronhálók

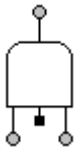
A mesterséges neuronhálók (***A**rtificial **N**eural **N**etworks – ANN*) a mesterséges intelligencia egyik legaktívabb és legkiterjedtebb kutatási területét képezik. Akárcsak a genetikus algoritmusok esetében, már a 60-as években voltak olyan kísérletek, amelyek az emberi gondolkodást, a neuront próbálták modellezni. Azóta számos modell jelent meg, amelyek más és más problémák megoldására specializálódtak. Ebben a dolgozatban az egyik legegyszerűbb és legalapvetőbb modellt ismertetjük, ugyanis a továbbiakban erre próbáljuk meg a genetikus algoritmusokat alkalmazni.

Minden neuronnak egy bizonyos számú bemenete van, amelyekhez **súly**okat (*weight*) rendelünk hozzá. Ezek határozzák meg, hogy az adott bemenet milyen fontossággal bír. Minden neuronnak van egy egyedi **küszöbérték**e (*threshold*), és ha a **neuron belső állapot**a (*net value*) nagyobb mint ez a küszöbérték akkor aktiválódik és a kimenet egy 1-es lesz, ha viszont az érték kisebb mint a küszöbérték akkor a neuron inaktív marad, tehát a kimenet egy 0-ás lesz. A kimeneti érték az adott neuronhoz kapcsolt összes neuron bemeneti értékét fogja képviselni.

A neuronhálóknak számos architektúrája ismeretes és ennek következtében más-más tanulási szabályt igényelnek. A felügyelt tanulásnál egy úgynevezett tanár van jelen ami korrigálja a hibákat, azaz visszajelez a neuron felé, a tanulási szabály pedig a súlyokat módosítja megfelelő mértékben, amíg a háló megtanulta az adott feladatot. A felügyelt tanulási módszerek közé tartozik a back-propagation (lásd a 2.2. szakaszt) és a delta szabály (*delta rule*). A nem felügyelt tanulásnál nincs jelen tanár, a neuron létrehozza a saját kimeneti értékét, amit később kiértékelünk.

2.1. A perceptron modell

A perceptron modellt Rosenblatt vezette be 1958-ban és a cél a neuronok lineáris összekapcsolása volt. Egy egyedülálló perceptron felépítése nagyon egyszerű. Két bementje (*input*), egy súlyeltolódása (*bias*) és egy kimenetje (*output*) van. A perceptron egy sematikus diagramja a 3. ábrán látható. A perceptron be- és kimenete bináris értékű, azaz 0 illetve 1 lehet. A bemenetek és a súlyeltolódás súlyokon keresztül csatlakozik a perceptronhoz. Ezeknek az értéke általában egy 0 és 1 közötti valós szám.

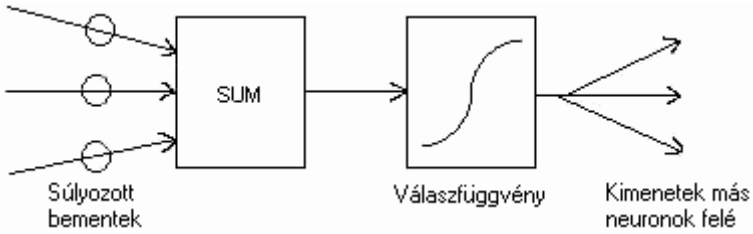


3. ábra. Egy egyszerű perceptron. A bemenet a két alsó kör, a súlyeltolódás a fekete négyzet és a kimenet a fent levő kör.

A bemeneti súlyokat összegezve, transzformáljuk egy **válaszfüggvény** (*hard-limiter function*) segítségével, ami meghatározza a neuron aktiváltságához szükséges küszöbértéket. A legelterjedtebb válaszfüggvény az úgynevezett sigmoid függvény:

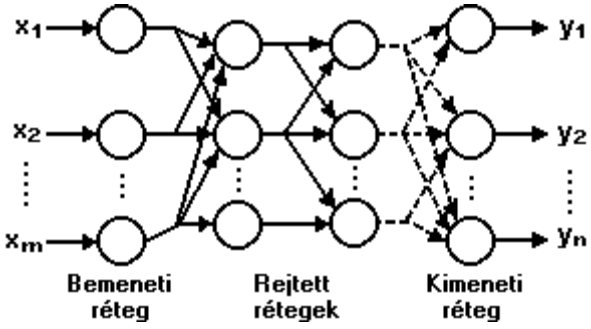
$$f(x) = \frac{1}{1+e^{-x}}.$$

Az így előállított kimeneti értékeket továbbítjuk a hálózat szomszédos neuronjai felé. A 4. ábra jól szemlélteti az eddig leirt perceptron modellt.



4. ábra. A perceptron modell folyamatábrája.

A perceptron esetében a legegyszerűbb tanulási szabály az amikor a súlyokat módosítjuk az elvárt kimenet és aktuális kimenet közötti különbségek függvényében. A perceptron csak lineárisan szétválasztható problémákat képes megoldani, komplexebb feladatok esetben többrétegű perceptronokra van szükség. A többrétegű neuronháló sematikus diagramja a 5. ábrán látható.

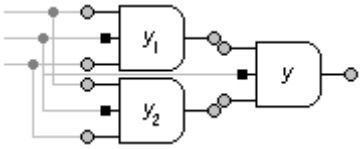


5. ábra. Többrétegű neuronháló.

Az XOR probléma megoldásához minimálisan 3 darab perceptronra van szükség (lásd a 6. ábrát), amelyek közül 2 a rejtett rétegben helyezkedik el. Ez azért van így, mert a kizáró vagy operátort lebontottuk részfeladatokra. A 1. táblázatból látszik, hogy az XOR az $y = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2) = (x_1 \vee x_2) \wedge \neg(x_1 \wedge x_2)$ összefüggéssel írható le. Ha a $y_1 = x_1 \vee x_2$ és $y_2 = \neg(x_1 \wedge x_2)$ jelöléseket hatjuk végre, akkor következik, hogy az előbbi egyenlőség a következőképpen néz ki: $y = y_1 \wedge y_2$.

1. táblázat. Az XOR igazságtáblázata.

x_1	x_2	XOR	$x_1 \wedge \neg x_2$	$\neg x_1 \wedge x_2$	$y_1 = x_1 \vee x_2$
0	0	0	0	0	0
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	0	0	0



6. ábra. Az XOR problémát megoldó perceptronok összekapcsolása.

2.2. A back-propagation módszer

Az egyrétegű neuronháló esetében minden neuron úgy módosítja a saját súlyát, hogy az elvárt és aktuális kimenet közötti különbség minél kisebb legyen. Matematikailag ezt a perceptron delta szabállyal írjuk le: $\Delta w_i = x_i \delta$, ahol δ = *elvárt kimenet – aktuális kimenet*, *w* pedig a súlyvektor, *x* pedig a bemeneti vektor. A perceptron tanulási szabálynak viszont nincs értelme többrétegű neuronhálóknál. A back-propagation módszer éppen ezt problémát hivatott megoldani, a fenti képlet pedig a következőképpen módosul: $\Delta w_i = n x_i \delta$, ahol $\delta = y(1-y)(d-y)$, y az aktuális kimenet, d pedig az elvárt kimenet. Ezzel a képlettel megadtuk a súlyok módosításának szabályát minden neuron számára és a delta kiszámítását a kimeneti réteg számára. Bizonyított, hogy a *p*-ik rejtett réteg, *q*-ik neuronjának a delta értékét a következő módon lehet kiszámítani: $\delta_p(q) = x_p(q)[1-x_p(q)]\sum w_{p+1}(q,i)\delta_{p+1}(i)$. A képletből jól látszik a módszer neve is: a hiba hátraterjed. Minden rejtett réteg delta értékének kiszámolásakor szükség van a rákövetkező rejtett réteg delta értékére. Tehát, egy 3-rétegű neuronháló esetében először kiszámítjuk a kimeneti réteg deltáját felhasználva az első képletet, majd az imént megadott képlet segítségével kiszámoljuk a megmaradt rejtett réteg delta értékeit. A kimeneti rétegtől a hiba lassan visszaterjed a háló többi neuronjai felé.

2.3. Alkalmazás: az XOR feladat

Az XOR (kizáró vagy) operátor fejlesztésére minimálisan egy 3 rétegű, 5 neuronból álló hálóra (2 bementi, 2 rejtett és 1 kimeneti neuron) van szükség, mivel a feladat nem lineárisan szétválasztható. A program implementálásánál azonban csak 3 neuronnal (2 rejtett és 1 kimeneti) kell dolgozni és egy mátrixban (a sor a neuront jelöli, az oszlop viszont a súlyeltolást és a 2 bemenetet) tároljuk a bemeneti súlyokat. A mátrix elemeit –1 és 1 közötti véletlen, valós számokkal inicializáljuk. A 2.2. fejezetben

bemutatott módszert alkalmazva megkapjuk az a neuronhálót, ami megtanulja az XOR feladatot.

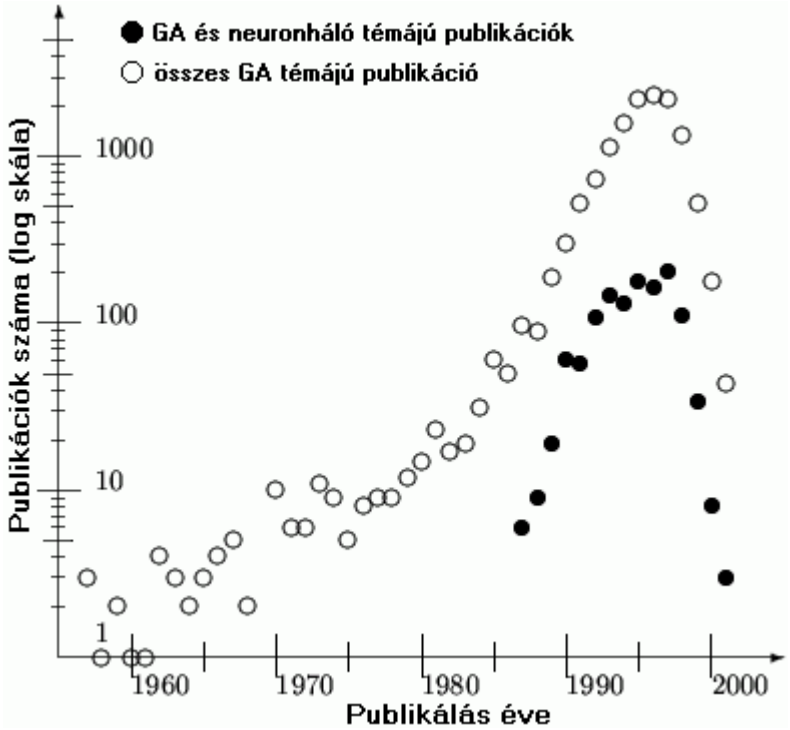
3. Genetikus algoritmusok

A genetikus algoritmusok (a továbbiakban GA) globális optimalizáló, sztochasztikus algoritmusok. Robusztus struktúrájukból kifolyólag képesek olyan nehéz feladatokat is megoldani, ahol nem rendelkezünk területspecifikus leírással, ezért az algoritmus problémafüggetlen. Mivel sztochasztikus algoritmus, ezért a keresési tér csak reprezentatív képviselőit vizsgálja meg, így jól alkalmazható NP-teljes feladatokra és nagy keresési térrel rendelkező problémákra. A GA az adaptív keresési technikák osztályát képviselik, a hagyományos „gyenge módszerekhez" képest egy sor hatékony, terület-független keresési heurisztikát kínál fel, anélkül, hogy erősen terület-specifikus ismeretet igényelne [De Jong 88].

A természetes szelekció mechanizmusát szimulálják egy valószínűsített adatcsere segítségével, alkalmazva a darwini elvet: a legrátermettebb túlélése (*survival of the fittest*). A GA az evolúciós számítások egy részhalmaza, amelyek evolúciós technikákat alkalmaznak a számítási algoritmusokra.

A genetikus algoritmusok fogalmát először Holland (az alapreferenciává vált [Holland 75]-ös monográfiában) és tanítványai (például [De Jong 75]) vezették be.

Az utóbbi évtizedben jelentősen megnőtt az érdeklődés a genetikus algoritmusok és általában az evolúciós számítások iránt. A 7. ábrán jól látszik, hogy a leközölt tudományos munkák száma évről-évre gyarapodott.



7. ábra. A genetikus algoritmusok iránti érdeklődés az utóbbi évtizedben exponenciálisan megnőtt. Leggyakrabban a neuronhálók fejlesztésére alkalmazzák. A publikációk száma 1997-ben volt a legmagasabb.

A genetikus algoritmusok legnépszerűbb alkalmazási területei a következők: mesterséges neuronhálózatok, mérnöki tudományok, vezérlés, képfeldolgozás, robotika,

gépi tanulás (*Machine Learning*), mintafelismerés stb. Ha a szerzők földrajzi eloszlását nézzük, akkor a publikációk több mint ¼-e az Egyesült Államokban tevékenykedő tudósoknak köszönhető. De komoly kutatások folynak e területen Japánban, az Egyesült Királyságban és Németországban, ahol szintén sok szerző van [Alander 01].

3.1. A genetikus algoritmus alkotóelemei

A genetikus algoritmus populációk sorozatát állítja elő operátorok segítségével úgy, hogy egyszerre több megoldással (egyeddel) dolgozik. Ahhoz, hogy a megoldásokból egyszerű operátorok segítségével könnyen újabb megoldásokat lehessen szerkeszteni, a megoldásokat kódolni kell. Minden megoldáshoz, hozzárendelünk egy szintaktikailag jól és könnyen kezelhető betű- vagy számsorozatot egy **kódoló függvény** segítségével. A 8. ábra egy 0-1-esekkel kódolt kromoszómát mutat be. Ezt nevezzük a megoldás **genotípus**ának, míg maga a megoldás a **fenotípus**. A genotípus pozíciói (indexei) a **gén**ek, az ott lévő értékek (számok vagy betűk) az **allél**ok.

0	1	1	0	0	0	1	0	1	1

8. ábra. A megoldást egy bitsorozatból álló kromoszóma kódolja. A harmadik gén allél értéke az 1.

A megoldások értékeit egy értékelő függvény, vagy **rátermettségi függvény** (*fitness function*) segítségével adjuk meg. Ennek a függvénynek kell megkeresni a globális optimumát. A rátermettségi függvény megválasztása lehet a legnehezebb feladat és egyben a legfontosabb is, hiszen ennek segítségével mérjük az egyedek teljesítményét, alkalmasságát, rátermettségét. Általában a rátermettség kiszámolása sok időt vesz igénybe, ezért sokat dob az algoritmuson, ha ezt jól választottuk meg. A rátermettségi függvény a keresési téren **rátermettségi tájkép**et (*fitness landscape*) határoz meg.

3.2. Genetikus operátorok

A kódokon alkalmazott operátorokat három nagy csoportba lehet sorolni. Az első csoportba tartoznak a **szelekció**k, amelyek valamilyen módszer segítségével bizonyos számú egyedet (általában a legrátermettebbeket) választanak ki a populációból. Az így kiválasztott egyedekre alkalmazzuk a **rekombináció**t, amely útján két szülő tulajdonságait ötvözve rátermettebb utódokat hozunk létre. A leszármazottak bizonyos százaléka mutálódhat is, alkalmazva a **mutáció**s operátort.

3.2.1. Szelekció, reprodukció

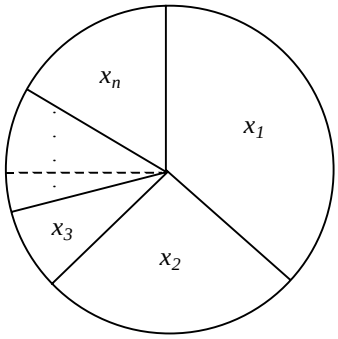
A szelekciós operátor problémafüggetlen és a rátermettséget veszi figyelembe. Ez kiválaszt a populációból egy egyedet (megoldást), és ezt annyiszor kell elvégezni, ahány szülőre szükség van az új populáció előállításához.

Az egyik ilyen szelekciós operátor a **rátermettség-arányos szelekció** (*fitness proportionate selection*), amely szerint egy megoldás kiválasztásának a valószínűsége annál nagyobb, minél nagyobb a rátermettsége a populáció rátermettségi átlagához képest. A populáció minden e elemére a kiválasztódás valószínűsége a következő:

$$P(e) = \frac{f(e)}{nf(Pop)},$$

ahol $f(e)$ a rátermettség értéke, n a populáció elemszáma, és $f(Pop)$ a populáció elemeinek átlagos rátermettsége. Ezt a feladatot a gyakorlatban az úgynevezett

rulettkerék módszer segítségével oldjuk meg [Jelasity 99]. Ennél az algoritmusnál a populáció minden egyedét a rulettkerék egy-egy mélyedése képviseli, amely egyenesen arányos az egyed rátermettségével (lásd a 9. ábrát). A rulettkerék módszer e változatát még **költség szelekció**nak (*cost szelection*) is nevezzük. Az egyedeket véletlenszerűen választjuk ki figyelembe véve a rátermettségi értékeket, általában a rátermettebb egyedek lesznek kiválasztva.



9. ábra. A rulettkerék módszer. Az x_1, x_2, x_3, …, x_n egyed által lefoglalt körcikk egyenesen arányos az egyed rátermettségével.

A költség szelekciónál felmerülhet az a probléma, hogy egy megoldás túlságosan magas fitnesz-értékkel rendelkezik a többihez képest, így mindig ő lesz a kiválasztott szülő, ami a változatosság csökkenéséhez vezet, az eredmény egy lokális optimumhoz fog konvergálni, az algoritmus "beragad". Ennek elkerülése végett a kromoszómákat a rátermettségük szerint rangsoroljuk, majd ennek következtében osztjuk ki az új fitnesz-értékeket. Ezt az eljárást használva a legrosszabb egyednek 1 lesz a fitnesz-értéke, a második legrosszabbnak 2 és így tovább, míg a legjobb kromoszóma fitnesz-értéke N lesz, ahol N a populáció mérete. Így a rulettkeréken minden egyed sokkal jobb arányban lesz képviselve mint az előbbi módszernél. Az új módszer neve a **rang szelekció** (*rank selection*).

Az előző problémát úgy is kiküszöbölhetjük, hogy az új populációt csak akkor fogadjuk el, ha ez rátermettebb az előzőnél, ellenkező esetben addig alkalmazzuk a szelekciós, rekombinációs és mutációs operátorokat, amíg az új populáció jobb nem lesz a réginél.

Egy másik szelekciós operátor a **verseny szelekció** (*tournament selection*), amikor is egy csoportot (általában 2-7 egyedet tartalmaz) választunk ki véletlenszerűen a populációból és ezek közül a rátermettebb lesz kiválasztva az új populációba.

Az **elitista** (*elitist*) genetikus algoritmus mindig áthelyezi az új populációba az eddig kapott legjobb egyedet, akár ki lett választva akár nem, feltéve ha az új populációban minden elem egyébként rosszabb lenne. A verseny szelekció természetétől fogva elitista. Ha a verseny szelekció esetében kiválasztott csoport csak két egyedet tartalmaz akkor az operátort **pár-verseny szelekció**nak (*binary tournament selection*) nevezzük [Jelasity 99].

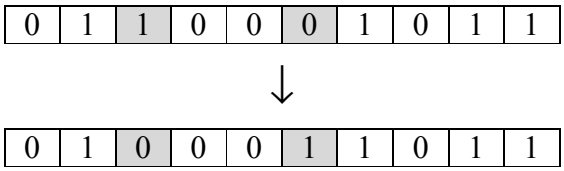
A fent leirt szelekciós operátorokat tetszőleges számszor alkalmazhatjuk, annak függvényében, hogy hány egyedre van szükségünk az új populációban. Ha a régi populáció összes tagját le akarjuk cserélni, akkor az algoritmusokat annyiszor hajtjuk végre amennyi egyedet tartalmaz a populáció. Ilyenkor **nem átfedő populációk**ról beszélünk [Pécsy 98]. Ha viszont a populációnak csak egy bizonyos százalékát cseréljük le akkor **átfedő populációk** jönnek létre, ami a **generációs rés** (*generation gap*) effektusban nyilvánul meg [De Jong 75]. Ha G a generációs rés paramétere egy genetikus algoritmusnak, akkor G=1 esetén nem átfedő populációk jönnek létre, míg

$0<G<1$ esetén átfedő populációkról beszélünk, amikor is GN darab egyed cserélődik, ahol N a populáció mérete. A lecserélendő egyedeket általában a fitnesz-értékük által meghatározott leggyengébb kromoszómák közül választják ki, a mi esetünkben pontosan GN darabot.

3.2.2. Mutáció

A mutáció unáris operátor, azaz egy operandust igényel, ami egy megoldás és ebből állít elő egy újabbat. A célja az egyedek, a populáció frissítése, változatosság bevitele a populációba. Ez biztosítja a lokális optimumba való beragadástól a védelmet. Ekkor véletlenszerűen kiválasztott pozíciókon lévő allél értékeket változtatunk meg. A 10. ábrán egy 10 hosszúságú kromoszóma mutálódik. Általában ezt úgy oldják meg, hogy minden pozíció egy rögzített valószínűséggel mutálódhat, és ezt úgy állítják be, hogy átlagosan egy pozíció változzon meg egy kódban.

pl. (a_1, a_2, a_3, a_4, a_5) egy egyed a populációból, a mutáció pozíciója 3. Ekkor az (a_1, a_2, a_3', a_4, a_5) leszármazottat kapjuk.

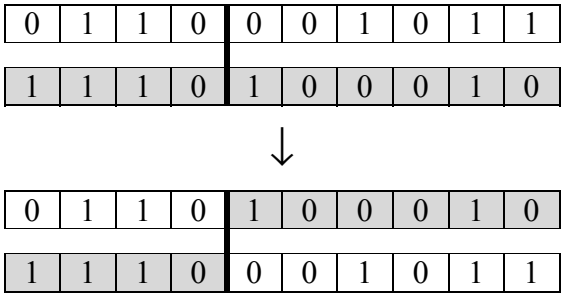


10. ábra. A kromoszóma harmadik és hatodik génje mutálódik.

3.2.3. Rekombináció, kereszteződés

A rekombinációk (keresztezés) több kiindulási megoldásból (szülőből) állítanak elő újabb megoldást, tehát egy bináris operátor. A célja az, hogy különböző egyedek tulajdonságait ötvözve újabb, esetleg rátermettebb egyedeket hozzunk létre. Az egyik ilyen rekombinációs operátor az **egypontos keresztezés** (*1-point crossover*). Véletlenszerűen választunk egy **kereszteződési pont**ot (*crossover point*) és a kapott két fél kódból új megoldást rakunk össze. A 11. ábra ezt a fajta keresztezést szemlélteti két különböző kromoszómán. Jól látszik, hogy a leszármazottak mind a két szülőtől örökölnek tulajdonságokat.

pl. (a_1, a_2, a_3, a_4, a_5) és (b_1, b_2, b_3, b_4, b_5) két egyed a populációból, a kereszteződési pont 2. Ekkor a keletkezett két leszármazott (b_1, b_2, a_3, a_4, a_5) és (a_1, a_2, b_3, b_4, b_5) lesz.



11. ábra. Az egypontos keresztezés. A keresztezési pont értéke 4, amit a vastag vonal jelöl. A leszármazottak mindkét szülő tulajdonságát öröklik.

A keresztezési operátoroknak számos változat ismeretes, attól függően, hogy milyen feladatra alkalmazzuk a genetikus algoritmust. Az itt bemutatott operátorok a

bináris ábrázolásra vonatkoznak, de a kromoszómákat lehet lebegőpontosan is ábrázolni (függvényoptimalizálási feladatoknál sokkal gyorsabb, hiszen nem kell átalakítani bináris formátumból) és ekkor más operátorokra van szükség [Michalewicz 96]. Ugyancsak speciális keresztezési és mutációs operátort alkalmaznak az utazóügynök feladat (***T**raveling **S**alesman **P**roblem – TSP*) esetében, ahol szintén számos változata létezik ezeknek az operátoroknak [Michalewicz 96].

3.3. Algoritmus

A genetikus algoritmus struktúrája nagyon egyszerű. A következőkben egy vázlatos és általános formában megadott algoritmust ismertetünk.

```
Hozzuk létre a P0 kezdeti populációt
t := 0
while not Kilépés(Pt)
  Pt' := UjElemek(Pt)
  Pt+1 := UjPopuláció(Pt',Pt)
  t := t + 1
```

A **kezdeti populáció**t (*initial population*) általában véletlen elemekkel töltik fel, de érdemes valamilyen egyszerű heurisztika által szolgáltatott viszonylag jó teljesítményű megoldásokból kiindulni. A populáció elemszáma a futás során változatlan (általában 50-100 egyed).

A `Kilépés(`P_t`)` függvény a már kiértékelt magoldások számát vizsgálja meg. Ha elértük a megengedett kiértékelések számát (tipikusan 1000 és 500000 között van), akkor az algoritmus leáll.

Az `UjElemek(`P_t`)` függvény feltölti a P_t' populációt új elemekkel. A szelekció segítségével kiválasztjuk a szülőket, a rekombináció ezekből egy utódot hoz létre és erre alkalmazható a mutáció, majd ennek az új elemnek a rátermettsége kerül kiszámolásra. A P_t' elemszáma nem feltétlenül azonos a P_t elemszámával. Ha mégis azonos akkor az `UjPopuláció(`P_t'`,`P_t`)` függvény a P_t' populációt adja. Ekkor a genetikus algoritmus **generációs** (*generational*). Ha a P_t' elemszáma kisebb, a megfelelő számú elemet törli a P_t-ből és helyükre a P_t' elemei kerülnek. Ha a P_t' csak egy elemből áll, akkor az algoritmus **helybeni** (*steady state*). Az elemek törlésére a fordított előjelű szelekcióhoz hasonló operátorok használhatók [Jelasity 99].

4. Genetikus algoritmusok és neuronhálók alkalmazása különböző optimalizálási problémákra

A mesterséges neuronhálók elsősorban arra voltak tervezve, hogy mintákkal dolgozzanak. Jól alkalmazhatóak mintafelismerésre, minták osztályozására, minták asszociációjához. A gyakorlatban elterjedt az optikai karakterfelismerés (***O**ptical **C**haracter **R**ecognition – OCR*), az amerikai hadsereg pedig a radarok által visszaadott jelek elemzésére használják. A minta asszociációk egy gyakorlati alkalmazása lehet az aláírás, arc és újlenyomat felismerése, ugyanis ezekben az esetekben egy nem egészen tiszta képből kell előállítani a teljes képet.

A genetikus algoritmusokat általában olyan problémákra érdemes használni, amelynek a keresési területe nagy, nem ismert a struktúrája, nem áll rendelkezésre területspecifikus tudás.

Sok alkalmazása van a gépi tanulás területén, mivel minden tanulási probléma megadható optimalizálási feladatként [Goldberg 89].

Egy másik alkalmazási terület az evolúciós mesterséges neuronhálók (***E**volutionary **A**rtificial **N**eural **N**etworks – EANN*) tanítása, fejlesztése. A genetikus algoritmusokat nagy sikerrel alkalmazták a neuronháló súlyainak a tanítására, hiszen bizonyos esetekben sokkal gyorsabban konvergál mint a hiba hátraterjedésének (*back-propagation*) a módszere. Neuronháló felépítésének a generálása, fejlesztése (*evolve*) sokkal bonyolultabb folyamat. Kis méretű hálóknál egy mátrix reprezentációt használunk, amiben megvan, hogy melyik neuron melyikkel van összekapcsolva, majd ezt a mátrixot kromoszómává konvertálva, ezeknek a különböző kombinációt fejlesztjük ki. Válaszfüggvényt (*learning function*) is lehet genetikus algoritmussal tanítani. Habár a genetikus programozás éppen erre hivatott (függvények, programok fejlesztése) mégsem ajánlatos a használata ebben az esetben, mivel neuronhálókkal kombinálva hihetetlenül lassú lehet.

Kombinatorikus, NP-teljes problémák (pl. gráfszínezés, utazó ügynök, ládapakolás, hátizsák, SAT) megoldására is széles körben alkalmazhatók. Felhasználják szabályozó rendszerekben, illetve fuzzy függvények keresésére is. Az amerikai hadsereg olyan egyenletek fejlesztésére használják, amelyek képesek többféle radar jelek megkülönböztetésére [Military 01], a brókercégek pedig GA által vezérelt programokat használnak a tőzsdei események előrejelzésére.

4.1. Játékstratégiák fejlesztése

Már a 80-as években voltak olyan kísérletek, ahol a genetikus algoritmusokat megpróbálták alkalmazni a gépi tanulásra. Ezek a próbálkozások a GA megalapítóihoz fűződnek: [De Jong 88], [Goldberg et al 88]. Később a GA-t a szimbolikus tanulás módszereivel ötvözték. [Janikow 93] a felügyelt tanulásra (induktív tanulási módszer) használta a genetikus algoritmusokat. A kifejlesztett módszer egy magas szintű leíró nyelvet használ akárcsak a szabály-alapú rendszereknél és ami lehetővé teszi a következtetések egyszerűbb kezelését. Az elkészített applikáció bebizonyította, hogy a GA képes magas szintű fogalmak feldolgozására feladat-specifikus tudást biztosítva. [Whitley et al 93] megpróbálták a GA-t a megerősítésen alapuló tanulási (*Reinforcement Learning*) problémákra alkalmazni. A kísérleti eredmények azt mutatták, hogy ez a módszer versenyképes volt más neuronhálókra használt megerősítéses tanulási paradigmával (időbeli különbség – ***T**emporal **D**ifference* – módszer).

Mivel a genetikus algoritmusok alkalmazhatóak neuronhálók (általában a bemeneti súlyok) fejlesztésére, így vannak olyan megoldások is ahol a stratégiát egy neuronháló reprezentálja és ezt tanítjuk az evolúciós algoritmusokkal. Ezt a módszert kevesen alkalmazták és csak egyszerű (a keresési tér kicsi) játékokra, hiszen a GA és neuronháló szimbiózisa hihetetlenül lassú lehet. Az egyik ilyen próbálkozás Fogel nevéhez fűződik és a 3x3-as amőbára (TTT) alkalmazta ([Fogel 93] és lásd a [Fogel 00] monográfiában az *5.2 General Problem Solving: Experiments with Tic-Tac-Toe* fejezetet). [Chellapilla et al 99a] több játékra is használták: fogolydilemma, TTT és dámajáték (*Checker*). A [Chellapilla et al 99b] és [Kim et al 02] cikkek azt írják le, hogy az evolúciós algoritmus talált egy olyan neuronhálót, ami segítségével a Checker majdnem szakértői, profi szinten játszik. [Pollack et al 97], [Pollack et al 98] pedig egy jobb Backgammon stratégia előállítására használták ezt a hibrid módszert. [Konidaris et al 02] genetikus algoritmust használ a neuronháló tanítására, ami a Go egy egyszerűbb változatát képes játszani.

[Freisleben et al 96] az amőba (*Go-Moku*) esetében egy megfelelő neuronhálót használ a szabad pozíciók kiértékelésére és ezt a hálót a megerősítésen alapuló tanulási módszerrel tanítja egy sor játékon keresztül.

[Chellapilla et al 99a] megpróbálták az evolúciós algoritmusok és neuronhálók segítségével egy jobb fogolydilemma (***I**terated **P**risoner's **D**ilemma – IPD*) stratégiát kapni. A véges állapotú automatákat többrétegű visszacsatolásos perceptronokra (***M**ultilayer **F**eedforward **P**erceptron – MLP*) cserélték le, azaz minden stratégia egy olyan MLP által volt képviselve, amelynek 6 bemente, bizonyos számú rejtett réteg és egy kimenete volt. Az implementált evolúciós algoritmus a következő lépéseket tartalmazta:

1. Kezdeti populáció inicializálása véletlenszerű MLP-kel. Minden háló neuronjának a súlya (*weight*) és súlyeltolódása (*bias*) egyenletes eloszlású a [-0,5, 0,5] intervallumon.
2. Minden szülőből egyetlen leszármazott jött létre úgy, hogy minden súlyhoz és súlyeltoláshoz egy standard Gauss eloszlású véletlen számot adtunk hozzá.
3. Minden háló játszott mindegyikkel egy round robin versenyben. Egy neuronháló-pár 151 darab lépest játszott, és minden háló rátermettsége a lépesek alatt elért átlagpontszám volt.
4. A hálók rangsorolása a rátermettségük alapján, a fele legjobb egyed kiválasztása, hogy a következő generációban a szülőket képviseljék.
5. Ha a program elért az előre megadott lépésszámig, akkor vége, különben folytasd a 2. lépéssel.

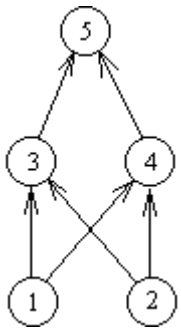
4.2. Az XOR probléma

Implementáljuk az XOR (kizáró vagy) operátort GA és ANN segítségével. A kizáró vagy csak annyiban különbőzik a vagytól, hogy ha mindkét operandus (legyen A és B) igaz, akkor az XOR hamisat eredményez.

A módszer nagyon hasonlít a genetikus programozáshoz (***G**enetic **P**rogramming – GP*), mivel ott is egy olyan struktúrát (az adott esetben egy elemző fa) kell optimalizálni, kell megkeresni, ami megoldja ezt a problémát. Míg a GP esetében egy Lisp programot fejlesztünk, addig itt egy neuronhálónak a struktúráját, szerkezetét kell megkeresni, vagy egy előre adott hálónak a súlyait kell maghatározni.

4.2.1. Neuronháló szerkezetének a fejlesztése

A komplexebb és időigényesebb feladat az amikor GA segítségével neuronháló architektúráját próbáljuk meg generálni, viszont ez sokkal hatékonyabb lehet. Ahhoz, hogy GA-val is valamilyen feladatot megoldjunk, szükséges egy megfelelő reprezentációt találni.



12. ábra. A legelterjedtebb neuronháló struktúra az XOR feladat számára.

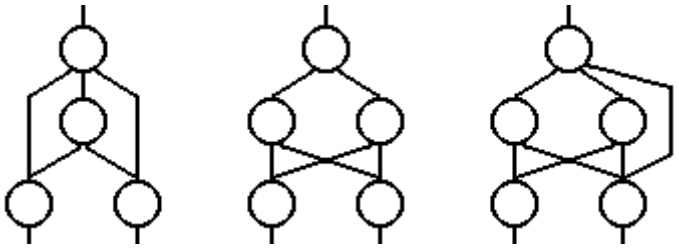
Egy neuronháló szerkezetét egy irányított szomszédossági mátrix segítségével tudjuk megadni, ahol leírható, hogy melyik neuron melyikkel van összekapcsolva. Példának okáért lássuk a 12. ábrát, ami egy irányított gráfhoz hasonlítható. Ehhez a neuronhoz tartozó szomszédossági mátrix (az irány az oszlop-sor) és a súlyeltolások:

	1	**2**	**3**	**4**	**5**	**Súlyeltolás**
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	1	1	0	0	0	1
4	1	1	0	0	0	1
5	0	0	1	1	0	1

A mátrix sorait és a megfelelő súlyeltolásokat egymás mellé írva, megkapjuk a kromoszóma reprezentációt, ami nem más mint a következő bitsorozat:

000000 000000 110001 110001 001101.

Az így megadott reprezentációt felhasználva, a genetikus algoritmus a 13. ábrán megadott neuronháló struktúrákat állította elő.



13. ábra. A GA által előállított neuronháló szerkezetek.

4.2.2. Neuronháló súlyainak a meghatározása

Egy másik módszer lehet az, amikor GA segítségével csak az előre megadott szerkezetű neuronháló bementi súlyait optimalizáljuk. Ez kevésbé bonyolultabb és időigényesebb folyamat.

4.2.2.1. Reprezentáció

Ekkor is a súlyokat egy mátrixban tároljuk a 2.3. részben megadott módon, a kromoszómák pedig a súlyok sorozatát fogják tartalmazni.

4.2.2.2. Rátermettségi függvény

A GA legfontosabb része és ugyanakkor a legnehezebb is. A rátermettségi függvény a 4 bemeneti mintára (00, 01, 10, 11) kiszámolja az elvárt (0, 1, 1, 0) és aktuális kimenetek közötti különbséget, majd ezeket összegezi. A négyes (ez a maximális hiba) számból kivonva az összhibát megkapjuk a fitnesz értéket, a GA pedig ezt az értéket maximalizálja, vagyis a hibát minimalizálja. A szelekció esetében a rulettkerék szelekció hatékonyabb mint a verseny szelekció, az egyelemű elitizmus, pedig jól működik, hiszen ily módon megőrizzük az eddigi legjobb neuronhálót.

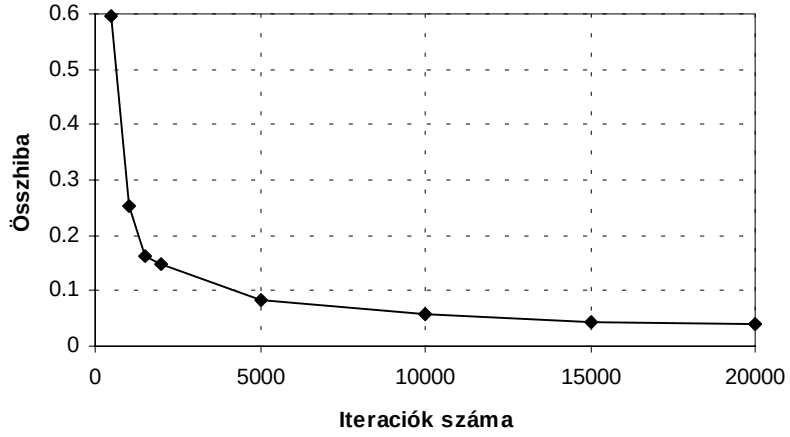
4.2.2.3. Genetikus operátorok

A keresztezés esetében az egypontos keresztezést használtuk a leszármazottak generálásához. Ez azért jó, mert a stringeknél létrejövő adatcsere következtében az egypontos keresztezés hajlamos megőrizni a szülők felsőbb részét.

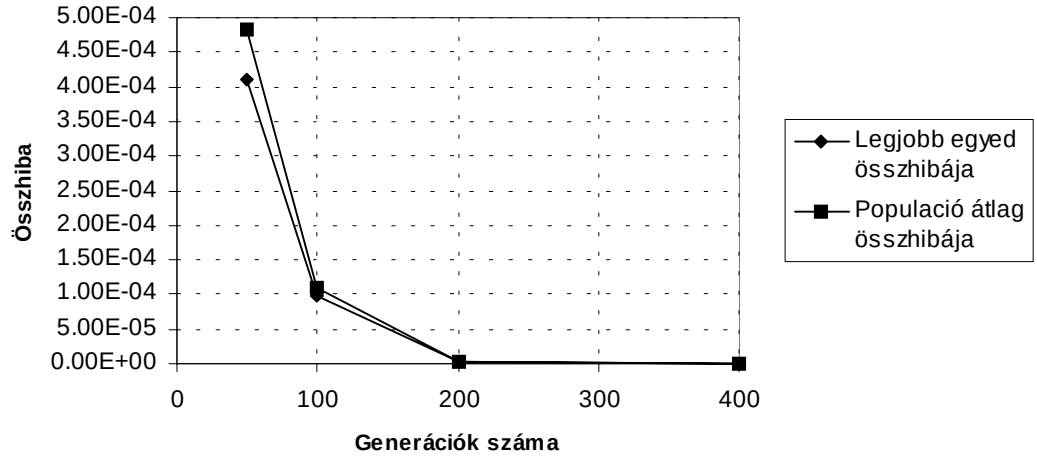
A mutáció véletlenül felcserél bizonyos elemeket egy kiválasztott egyedben, ami további genetikai változatossághoz vezet, elkerülve a lokális optimum csapdákat.

4.2.2.4. Kísérleti eredmények

Az eredmények azt mutatják, hogy ez a módszer sokkal pontosabb megoldásokat szolgáltat mint a hagyományos back-propagation algoritmus (lásd a 14. és 15. ábrákat), habár a futáshoz szükséges idő jóval több. A paramétereket (iteráció szám, populáció mérete, keresztezési – és mutációs valószínűségek) széles határok között változtattuk és a programmal több tesztet végeztünk. Az ábrák 10 különböző futásból származó összhibák átlagát jelenítik meg. Az összhiba mind a 4 bemeneti mintára (00, 01, 10 és 11) kapott hibák összessége, ami nem más mint az elvárt és aktuális értékek közötti különbségek.

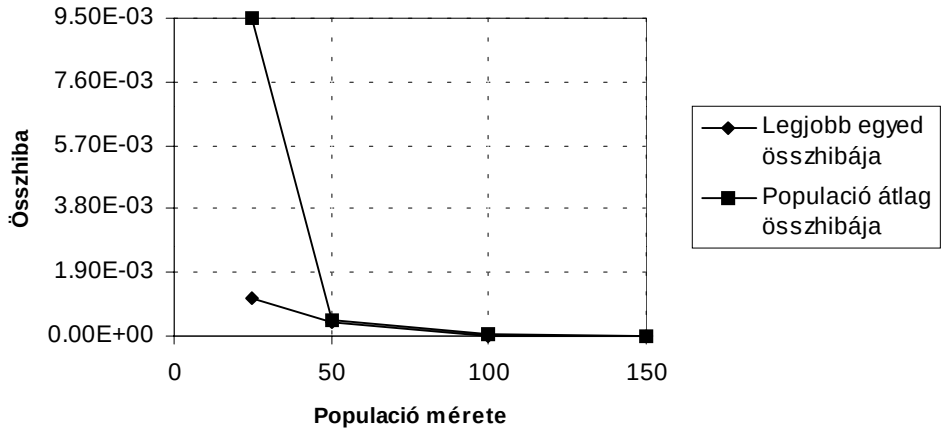


14. ábra. A BP által elért eredmények.



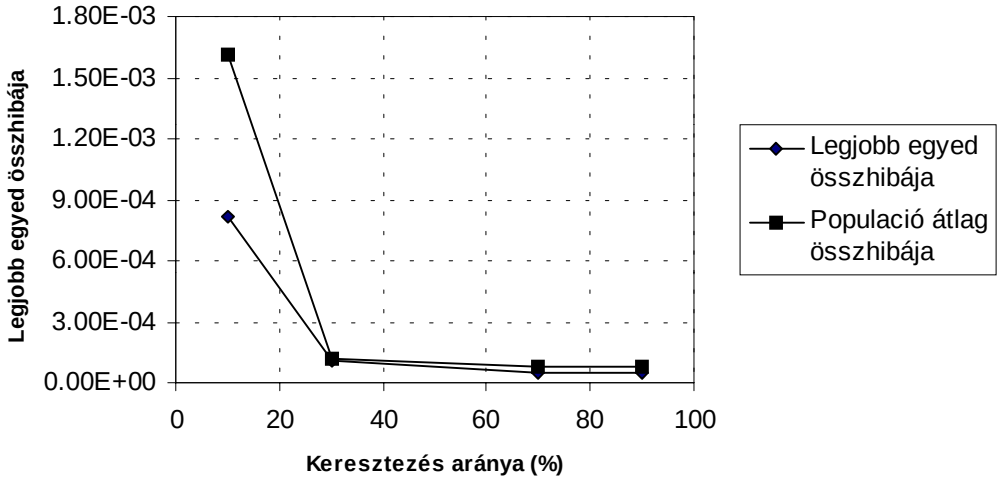
15. ábra. A GA-val elért eredmények. A populáció mérete 50, a keresztezési valószínűség 0,7, a mutációs valószínűség 0,3, elitista verseny szelekció.

A genetikus algoritmusnál a legfontosabb paraméter a generáció-szám mellett a populáció mérete. A nagyobb populáció genetikai változatosságot biztosít, fontos génanyag fejlődik ki, amely pont az optimális megoldást tartalmazza. Az összhiba nagy mértékben csökkenthető a populáció számának növelésével, viszont nagyon erőforrás-igényes. A kiértékelést a 16. ábra szemlélteti.



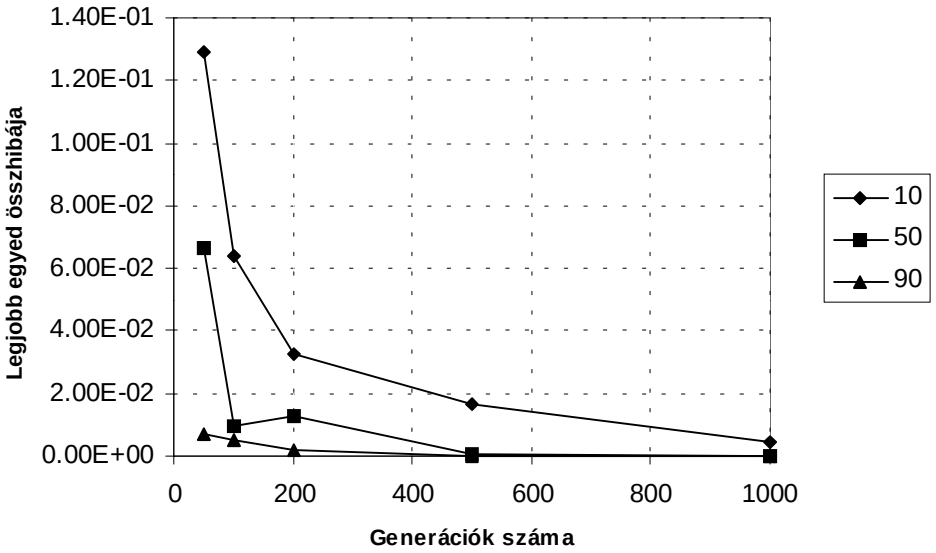
16. ábra. Az összhiba mértékének csökkentése a populáció méretének növelésével. Az iteráció szám 50, a keresztezési valószínűség 0,7 a mutációs valószínűség 0,3, elitista verseny szelekció.

Az evolúciós algoritmusok ereje abban rejlik, hogy képesek rátermettebb egyedeket előállítani, úgy, hogy öröklik a szülök jó tulajdonságait. Ebben nagy szerepe a keresztezésnek van és ezt bizonyítják a mérési eredmények is, amelyet a 17. grafikon ábrázol.



17. ábra. A keresztezés szerepe. Az iteráció szám 50, a populáció mérete 50, a mutációs valószínűség 0,3, elitista verseny szelekció.

Összegezésképpen tekintsük a 18. ábrát, ahol a GA egyik két legfontosabb paraméterét egyidejűleg dinamikusan változtattunk.



18. ábra. A különböző keresztezési arányok (százalékban) eredményei a generációk függvényében. A populáció mérete 50, a mutációs valószínűség 0,3, elitista rulettkerék szelekció.

5. Szakirodalom, hivatkozások

1. [Alander 01] Jarmo T. Alander: An Indexed Bibliography of Genetic Algorithms and Neural Networks, 2001, Report 94-1-NN, University of Vaasa, Department of Information Technology and Production Economics, Vaasa, Finland
2. [Chellapilla et al 99a] Kumar Chellapilla and David B. Fogel. Evolution, Neural Networks, Games, and Intelligence. In Proceedings of the IEEE, Vol. 87:9, pp. 1471-1496, 1999.
3. [Chellapilla et al 99b] Kumar Chellapilla and David B. Fogel. Evolving Neural Networks to Play Checkers without Expert Knowledge. IEEE Transactions on Neural Networks, Vol. 10:6, pp. 1382-1391, 1999.
4. [De Jong 75] Kenneth A. De Jong (1975) An Analysis of the Behavior of a Class of Genetic Adaptive Systems. PhD Dissertation. Dept. of Computer and Communication Sciences, Univ. of Michigan, Ann Arbor.
5. [De Jong 88] Kenneth de Jong: Learning with Genetic Algorithms: An Overview, Machine Learning 3 (2-3): 121-138, 1988
6. [Fogel 93] David B. Fogel. Using Evolutionary Programming to Create Neural Networks that are Capable of Playing Tic-Tac-Toe. 1993, In Proceedings of the IEEE International Conference on Neural Networks (ICNN-93), San Francisco, pages 875–880, IEEE Press, Piscataway, NJ.
7. [Fogel 00] David B. Fogel (2000). Evolutionary Computation: Toward a New Philosophy of Machine Intelligence, IEEE Press, New York
8. [Freisleben et al 94] Bernd Freisleben and Hartmut Luttermann. Learning to Play the Game of Go-Moku: A Neural Network Approach, Australian Journal of Intelligent Information Processing Systems (AJIIPS), 3(2), 52-60, 1996
9. [Goldberg 89] David E. Goldberg: Genetic Algorithms in Search, Optimization and Machine Learning, 1989, Addison-Wesley Pub Co
10. [Goldberg et al 88] David E. Goldberg and John H. Holland: Genetic Algorithms and Machine Learning, Machine Learning 3 (2-3): 95-99, 1988

11. [Holland 75] John H. Holland: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology Control, and Artificial Intelligence, First MIT Press Edition 1992, Cambridge, MA, second edition; First edition 1975, The University of Michigan Press
12. [Janikow 93] Cezary Z. Janikow: A Knowledge-Intensive Genetic Algorithm for Supervised Learning, Machine Learning 13 (2-3): 189-228, 1993
13. [Jelasity 99] Jelasity Márk: Genetikus algoritmusok, Mesterséges intelligencia-ban, szerkesztette Futó Iván, 1999, Aula Kiadó, Budapest, 549-568 old.
14. [Kim et al 02] Kyung-Joong Kim and Sung-Bae Cho: Checkers Strategy Evolution with Speciated Neural Networks, 2002, in M. Ishizuka and A. Sattar (Eds.): Proceedings of the 7th Pacific Rim International Conference on Artificial Intelligence (PRICAI 2002): Trends in Artificial Intelligence, Tokyo Japan, Lecture Note in Artificial Intelligence (LNAI) 2417, p. 599, Springer-Verlag, Berlin Heidelberg
15. [Konidaris et al 02] George Konidaris, Dylan Shell, Nir Oren: Evolving Neural Networks for the Capture Game, 2002, SAICSIT Postgraduate Research Symposium
16. [Michalewicz 96] Zbigniew Michalewicz: *Genetic Algorithms + Data Structures = Evolution Programs,* 1996, 3rd Revision edition, Springer-Verlag, Berlin-Heidelberg-New York
17. [Military 01] US Military GA Archives, 2001: www.aic.nrl.navy.mil/galist/
18. [Pécsy 98] Pécsy Gábor: Genetikus algoritmusok a képfeldolgozásban, 1998, Államvizsga-dolgozat, Eötvös Loránd Tudományegyetem, Természettudományi Kar, Programtervező-matematikus Szak, Budapest
19. [Pollack et al 97] Jordan B. Pollack, Alan D. Blair, Mark Land. Coevolution of a backgammon player. in Christopher G. Langton and Taksunori Shimohara, editors, Artificial Life V: Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems (Nara, Japan, 1996), MIT Press, Cambridge, MA, 1997
20. [Pollack et al 98] Jordan B. Pollack and Alan D. Blair. Co-Evolution in the Successful Learning of Backgammon Strategy. Machine Learning 32(3): 225-240, 1998.
21. [Whitley et al 93] Darrell Whitley, Stephen Dominic, Rajarshi Das, Charles W. Anderson: Genetic Reinforcement Learning for Neurocontrol Problems, Machine Learning 13 (2-3): 259-284, 1993