Universitatea „Babeş-Bolyai”
Facultatea de Matematică şi Informatică
Specializarea Informatică

BÓDIS LÓRÁNT

ALGORITMI GENETICI ÎN TEORIA JOCURILOR

Lucrare de diplomă

Coordonator ştiinţific:
Dr. Soós Anna

2003

„Babeş-Bolyai" Tudományegyetem
Matematika és Informatika Kar
Informatika Szak

BÓDIS LÓRÁNT

GENETIKUS ALGORITMUSOK A JÁTÉKELMÉLETBEN

Diplomadolgozat

Szakmai irányító:
Dr. Soós Anna

Kolozsvár, 2003

"Babeş-Bolyai" University
Faculty of Mathematics and Computer Science
Department of Computer Science

LÓRÁNT BÓDIS

GENETIC ALGORITHMS IN GAME THEORY

Diploma Work

Supervisor:
Dr. Anna Soós

2003

Kivonat

A számítógépes játékok és játékelmélet fontos kutatási területei a mesterséges intelligenciának, ahol a cél minél hatékonyabb keresési algoritmusok fejlesztése. Ezért a kutatási eredmények általában felhasználhatók más hasonló területen is. Mivel a hagyományos keresési algoritmusok idő- és tárbonyolultsága nagy lehet, így szükségessé vált a fejlettebb keresési eljárások alkalmazása.

A genetikus algoritmusok az utóbbi évtizedben széles körben elterjedt optimalizálási módszereknek bizonyultak. Robusztus struktúrájukból kifolyólag képesek olyan nehéz feladatokat is megoldani, ahol nem rendelkezünk terület-specifikus leírással, ezért az algoritmus probléma-független. A GA az adaptív keresési technikák osztályát képviselik, a hagyományos „gyenge módszerekhez" képest egy sor hatékony, terület-független keresési heurisztikát kínál fel. A természetes szelekció elvén alapszik, számos biológiai folyamatot imitálva, szimulálva: kromoszóma reprezentáció, genetikus operátorok, egyed kiválasztás, rátermettségi függvény stb.

Mint sztochasztikus algoritmus jól alkalmazható NP-teljes feladatokra és nagy keresési térrel rendelkező problémákra. A játékok pont ilyen típusú feladatokat képviselnek: nagy a keresési tér és nincs konkrét terület-specifikus ismeret. A dolgozat folyamán áttekintjük a genetikus algoritmusokat, megismerünk néhány játékelméleti fogalmat és keresési módszert, valamint azt, hogy milyen evolúciós próbálkozások voltak a gépi tanulás területén. Felhasználva az eddigi kutatási eredményeket, stratégiákat fejlesztünk ki az iterált fogolydilemma és amőba számára. Mindkét játék esetében több módszert ismerünk meg, különböző modern technikát alkalmazva: az egyedek egy úgynevezett mérkőzés rátermettség keretén belül versenyeznek egymással és a használt értékelő függvény a versengő rátermettségi függvény; az együtt-fejlődés (koevolúció) a verseny rátermettségi függvényt és a megosztott mintázást használja; genetikus minimax eljárás stb.

Kulcsszavak: genetikus algoritmusok, evolúciós számítások, metaheurisztikák, együtt-fejlődés, játékok, játékelmélet, iterált fogolydilemma, amőba, kirakós játék.

Abstract

The computer games and game theory are important research fields of the artificial intelligence, where the aim is the development of more and more efficient searching algorithms. Therefore the experimental results usually can be applied in other related fields. Since the time- and space complexity of the traditional searching methods could be large, thus it became necessary the utilization of more advanced searching procedures.

In the last decade the genetic algorithms had become widely used optimization methods. Having a robust structure, they are able to solve such hard problems where we don't have domain-specific description, therefore the algorithm is domain-independent. Genetic algorithms represent a class of adaptive search techniques, providing over traditional "weak methods" a set of efficient, domain-independent heuristics. It is based upon the principles of natural selection, imitating, simulating several biological processes: chromosome representation, genetic operators, individual selection, fitness function etc.

As a stochastic algorithm it is useful for NP-hard tasks and for problems with large search-spaces. The games are representing exactly these kinds of tasks: the search-space is large and there is no domain-specific knowledge. In this paper, work we will overview the genetic algorithms, we will get acquainted with some game theory concepts and searching methods, as well as the evolutionary attempts used in machine learning. Putting the foregoing experimental results to use, we will evolve, develop strategies for the Iterated Prisoner's Dilemma and Tic-Tac-Toe. For both of the games we will present several methods, using different modern techniques: the individuals will compete in a so called tournament fitness and the adopted fitness function is the competitive fitness function; the co-evolution utilizes the competitive fitness sharing and shared sampling; genetic minimax procedure etc.

Keywords*: genetic algorithms, evolutionary computation, metaheuristics, co-evolution, games, game theory, Iterated Prisoner's Dilemma, Tic-Tac-Toe, Puzzle.*

Tartalomjegyzék

Table of Contents

1. Bevezetés

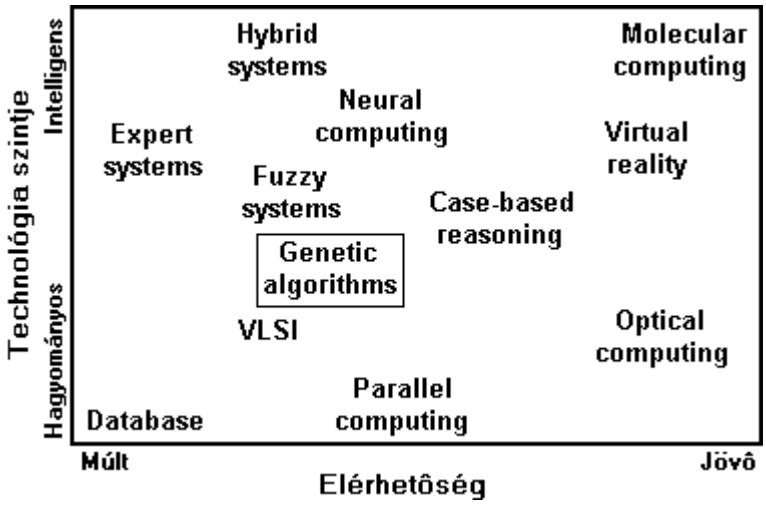
A mesterséges intelligencia – röviden MI (angolul ***A**rtificial **I**ntelligence – AI*) – az informatika egyik legfontosabb és legkorábban kialakult kutatási területe. Informatikáról 1945 óta beszélhetünk, amikor is az Egyesült Államokban megépítették az első modern számítógépet, amely a Neumann-elvre épült. A számítógépek fejlődésével lehetőség nyílt nagyobb mennyiségű adatok kezelésére és feldolgozására. Öt évvel később, 1950-ben Alan Turing megfogalmazta az első mesterséges intelligencia feladatot, amely a Turing-teszt néven vált ismerté. A feladat megoldása egy olyan gép, számítógépes program megépítését illetve megírását igényli, amely képes „gondolkodni". Azóta az MI egy rohamosan fejlődő tudományággá vált, számos területet foglalva magába: gépi tanulás, mesterséges neuronhálók, játékelmélet, evolúciós számítások, adatbányászat, ütemezés, mesterséges látás, beszédfelismerés, képfelismerés, szakértői rendszerek, robotika stb.

Ebben a dolgozatban az MI két nagyon népszerű területébe – evolúciós programozás és játékelmélet – 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 és ugyanakkor a játékok segítségével nagyon sok gazdasági, társadalmi, katonai folyamatot lehet szimulálni. Számos folyóirat (*Artificial Intelligence, Artificial Intelligence Review, Machine Learning, Evolutionary Computation, Journal of Artificial Intelligence Research, Journal of Machine Learning Research, Game Theory, Journal of Intelligent and Robotic Systems, Genetic Programming and Evolvable Machines, Artificial Life, IEEE Transactions* stb.) témáját képezi a fenti két terület, évente több száz cikk, dolgozat, jelentés lát napvilágot, valamint konferenciákat, rendezvényeket (*GA, GECCO, Artificial Life, IEEE, ANTS* stb.) szerveznek, ahol a megvitatott kérdések szintén ezekkel a témákkal kapcsolatosak.

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 1.1. 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.

A legtöbb szimbolikus mesterséges intelligenciai rendszer statikus. Ezek nagy része általában csak egy adott speciális problémát képes megoldani, ezért nehéz őket más feladatok megoldására átalakítani, nehezen adaptálódnak. A genetikus algoritmusok ezt és sok más problémát hivatottak kiküszöbölni, megoldani. Mivel a struktúrájuk általános ezért nagyon rugalmasak, minimális módosítással könnyen alkalmazhatók különböző típusú feladatokra. Nagyon egyszerű és alapvető elvet követnek: a természetes biológia szelekció. A genetikus algoritmus generációról-generációra a probléma lehetséges megoldásait fejleszti ki, minden lépésbe szűkítve a keresési teret, megtartva az aktuális lépésben a legmegfelelőbbnek tűnő megoldásokat.



1.1. ábra. A genetikus algoritmusok által elfoglalt hely az MI technológiák között.

A dolgozat szerkezete, kontúrja a következőképpen néz ki. A 2. fejezetben a genetikus algoritmusokat ismerjük meg, a fontosabb genetikus operátorokat, az algoritmus lehetséges változatait és számos egyéb jellemzőt, amire szükségünk lesz a dolgozat további fejezeteiben. Mivel az informatikában tudományosan csak azok a módszerek elfogadottak, amelyek matematikailag is bizonyítottak, így kitérünk a sémaelméleten keresztül a genetikus algoritmusok működési elvére. Röviden bemutatunk néhány fejlett kereső eljárást, amelyeket speciális, sajátos genetikus algoritmusoknak is tekinthetünk: hegymászó algoritmus, tabukeresés, szimulált hűtés. A genetikus algoritmus egy olyan programozási módszer, technika, amit több rokon kutatási területen alkalmaznak. Ennek értelmében kitérünk egy

korábban kifejlesztett módszerre (evolúciós programozás) és a legaktuálisabb alkalmazási területre (genetikus programozás). Persze nem maradnak el a példák sem, minden evolúciós módszer esetében bemutatunk egy-egy konkrét alkalmazást, érthetőbbé és megfoghatóbbá téve így az elméleti leírást.

A 3. fejezetben néhány játékelméleti fogalommal ismerkedünk meg: gépi tanulás, Nash egyensúly, játékok reprezentálása, tulajdonságai, osztályozása és bonyolultsága. Megtanuljuk a minimax eljárást, amely a legelterjedtebb kereső algoritmus kétszemélyes játékok számára.

A 4. fejezetben ötvözzük az első és második fejezetben elhangozott programozási módszereket, felhasználva a legújabb kutatási eredményeket. Az itt megjelenő újabb fogalmak valamilyen formában kapcsolódnak a már ismertetett evolúciós technikákhoz. Ezek tulajdonképpen finomítások, amelyeket a konkrét feladatra alkalmazva, növelik a genetikus algoritmus teljesítményét, hatásfokát. A játékok amelyeken alkalmazzuk az evolúciós eljárásokat az iterált fogolydilemma és az amőba.

Az 5. fejezetben egy teljes egészében saját készítésű számítógépes alkalmazás kerül bemutatásra. Az előző fejezetekben ismertetett algoritmusokat, technikákat igénybe véve, olyan szimulációs programot készítünk, amely az iterált fogolydilemma, az amőba és a kirakós játék számára próbál meg optimális stratégiákat kifejleszteni. Ugyancsak ebben a fejezetben vezetjük be a kirakós játékot, és itt ismerjük meg a játék számára kifejlesztett genetikus algoritmust is. A három különböző játék-típust alfejezetekként tárgyaljuk, dokumentálva a program megfelelő részeit ugyanakkor bemutatva az adott dialógusablak komponenseit, a könnyebb felhasználás érdekében. A fejezet fontos része a kísérleti eredmények kiértékelése, amelyeket grafikonok segítségével szemléltetünk.

2. Genetikus algoritmusok és evolúciós számítások

A genetikus algoritmusok (a továbbiakban GA) globális optimalizáló, sztochasztikus algoritmusok. Robusztus struktúrájukból kifolyólag képesek olyan nehéz feladatokat is megoldani, ahol nem rendelkezünk terület-specifikus leírással, ezért az algoritmus probléma-független. 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 [DeJong88].

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.

Mivel az utóbbi években nagy érdeklődés mutatkozott a genetikus algoritmusok iránt és széles körben alkalmazták különböző kutatási területeken [Alander01], [Alander00a], [Alander00b], így nagyon sok változata létezik. Mi több, a használt genetikus operátorok típusa és száma is problémáról-problémára változik, így a GA az adott feladatra testre szabható, biztosítva ily módon a jobb és gyorsabb konvergenciát. Számos könyv, monográfia jelent meg a genetikus algoritmusokról és minden szerző más-más módon közelíti meg őket, bemutatva a sajátos alkalmazási területeket [Goldberg89], [Mitchell96], [Davis91].

2.1. Kapcsolat a biológiával

Mivel a genetikus algoritmusok a természetben lezajló folyamatokból inspirálódtak, így a terminológia egy részét a biológiából (elsősorban a genetikából) kölcsönözték, amit szükség esetén az informatikában ismert fogalmakkal bővítettek ki. A genetikus algoritmusok jobb megértése végett, ismételjük át a genetikában használt fogalmakat, ugyanis ezek közül sokat fogunk használni az elkövetkezőkben.

Az egy fajon belüli csoportot populációnak nevezzük. A populációt egyedek alkotják, amelyeket a kromoszómájuk határoz meg. A kromoszóma gének sorozatából áll, amely a

tulajdonságok átörökítését (a szülőről az utódra) biztosítja. Egy egyedben levő gének összességét genotípusnak nevezzük. A fenotípus a genotípus megnyilvánulása az adott egyedben és tartalmazza a szerzett tulajdonságokat is. A keresztezés és a mutáció során az egyes egyedekből új egyedek alakulnak ki.

A genetikus algoritmus bizonyos mértékig kapcsolatban áll az evolúció darwini elméletével, ami a természetes szelekción alapszik. Az algoritmus általában nem tekinthető evolúciós folyamatok modelljének, sokkal közelebb áll a mesterséges szelekció módszeréhez, azaz úgy fogható fel, mint egy adott probléma számára megoldások kinemesítésére szolgáló módszer.

2.2. Történelmi áttekintés, irányzatok

A 60-as években megpróbáltak a természetben lezajló szelekciós folyamatok mintájára számítógépes modelleket létrehozni. Ezek a modellek képesek mérnöki (elsősorban optimalizálási) feladatok megoldására.

[Jelasity99] négy fő alkalmazási, kutatási területet sorol fel, ahol a genetikus algoritmusokon és általában a természetes szelekción alapuló szimulációkat használnak a különböző optimalizálási feladatok megoldására. Ezeket a módszereket nem lehet élesen elkülöníteni, hiszen ugyanazon az elven alapulnak csak mások a használt eszközök.

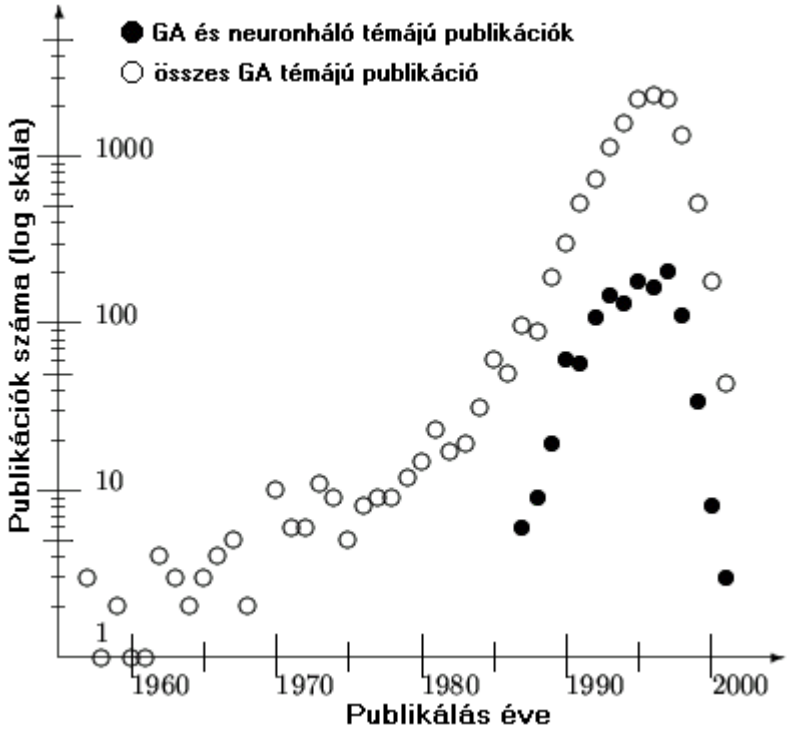
Egymástól függetlenül több próbálkozás is született. Németországban 1973-ban Rechenberg vezette be az **evolúciós stratégiák**at (*Evolutionsstrategie*), amelyeket repülőgép-szárnyak paramétereinek az optimalizálására használt. Az **evolúciós programozás**t (*evolutionary programming*) Fogel, Owens és Walsh dolgozta ki Amerikában, és egyszerű problémák megoldására szolgáló véges automaták automatikus kifejlesztésére használták. A **genetikus programozás** (*genetic programming*) egy még újabb terület és a genetikus algoritmus egy speciális alkalmazási területe, amikor is a cél meghatározott feladatokat végrehajtó számítógép programok (általában LISP nyelven) automatikus fejlesztése. Ez a próbálkozás Koza nevéhez fűződik (1992).

A **genetikus algoritmusok**at (*genetic algorithm*) Holland fejlesztette ki a michigani egyetemen és 1975-ben összefoglalta a kutatás eredményeit. Célja a szelekció és adaptáció számítógépes és matematikai modellezése volt.

A fent említett négy fő terület gyűjtőneve **evolúciós számítások** (*evolutionary computation*).

Az utóbbi évtizedben jelentősen megnőtt az érdeklődés a genetikus algoritmusok és általában az evolúciós számítások iránt. A 2.1. ábrán jól látszik, hogy a leközölt tudományos munkák száma évről-évre gyarapodott. Ez azzal magyarázható, hogy manapság viszonylag elég olcsó és gyors számítógépek állnak a kutatók rendelkezésére, lehetővé téve ily módon ezeknek a módszereknek a könnyebb és hatékonyabb használását, hiszen ezek nagyon hardverigényes számítási folyamatok. 1997-től egy csökkenő tendenciát mutat, amikor is a legmagasabb volt a publikációk száma.

A genetikus algoritmusok legnépszerűbb alkalmazási területei a következők: mesterséges neuronhálózatok, mérnöki tudományok, vezérlés, képfeldolgozás, robotika, gépi tanulás, 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 tudós tevékenykedik [Alander01].



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

2.3. A genetikus algoritmus általános bemutatása

A genetikus algoritmus egy sztochasztikus, globális optimalizáló módszer, amely a keresési tér minél rátermettebb egyedeit hivatott felfedezni. A módszer fő előnye, hogy széles

körben alkalmazható, általában nem használ területfüggő tudást, így akkor is működik, ha a feladat struktúrája kevéssé ismert. A GA az aktuális populációból minden lépésben egy új populációt állít elő. Az alapgondolat abból indul ki, hogy általában minden populáció az előzőnél rátermettebb elemeket tartalmaz és ezért a keresés folyamán egyre jobb megoldások állnak rendelkezésre.

A genetikus algoritmusok fogalmát először Holland (az alapreferenciává vált [Holland92]-ös monográfiában) és tanítványai (például De Jong [DeJong75]) vezették be. Holland a [Holland92]-ben a genetikus algoritmusok különböző illusztrálását (lásd az *Illustrations* fejezetet) mutatja be a következő kutatási területeken: genetika, közgazdaságtan, játékelmélet, keresések, mintafelismerés és statisztikai következtetés, vezérlés- és függvényoptimalizálás, központi idegrendszer. Holland kidolgozott és integrált két tételt:

1. az összetett, bonyolult struktúrák egyszerű reprezentálása bitsorozatok segítségével, és
2. az ilyen struktúrák javítása, tökéletesítése az egyszerű transzformációk segítségével.

Davis szerint a [Davis87]-ben egy genetikus algoritmusnak 5 komponense, összetevője kell legyen annak érdekében, hogy megoldjon egy problémát, feladatot:

1. A probléma megoldásának egy kromoszóma reprezentációja;
2. Kezdeti megoldásokból álló populációt létrehozó módszer;
3. Egy értékelő függvény, ami a környezet szerepét tölti be, a megoldások becslése a rátermettségük értelmében;
4. Genetikus operátorok, amelyek a gyermek, leszármazott összetevőit, komponenseit változtatják meg reprodukálás, szaporodás közben;
5. A genetikus algoritmus által használt paraméterek értékei (populáció mérete, genetikus operátorok használatának a valószínűségi értékei stb.).

2.4. A genetikus algoritmus alkotóelemei

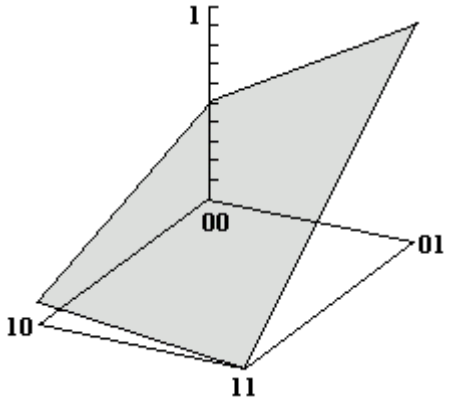
A genetikus algoritmus populációk sorozatát állítja elő operátorok segítségével úgy, hogy egyszerre több megoldással (egyeddel) dolgozik. Ahhoz, hogy a megoldásokból egyszerű operátorok segítségével könnyen újabb megoldásokat lehessen szerkeszteni, a megoldásokat kódolni kell. Minden megoldáshoz, hozzárendelünk egy szintaktikailag jól és könnyen kezelhető betű- vagy számsorozatot egy **kódoló függvény** segítségével. A 2.2. 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 levő értékek (számok vagy betűk) az **allél**ok.

0	1	1	0	0	0	1	0	1	1

2.2. ábra. A megoldást egy bitsorozatból álló kromoszóma kódolja. A harmadik gén allél értéke 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. Tegyük fel, hogy az *f* értékelő függvény a 00, 01, 10 és 11 kromoszómáknak (ami a jelen esetben koordináták a síkban) a 0,5, 0,9, 0,1 és 0 rátermettségeket felelteti meg. Ekkor a függvény által leírt tájkép a 2.3. ábrán látható.



2.3. ábra. Egy fiktív rátermettségi tájkép. A 00, 01, 10 és 11 kromoszómák rátermettsége: 0,5, 0,9, 0,1 és 0.

2.5. 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 során két szülő tulajdonságait ötvözve

rátermettebb utódokat hozunk létre. A leszármazottak bizonyos százaléka mutálódhat is, alkalmazva a **mutáció**s operátort.

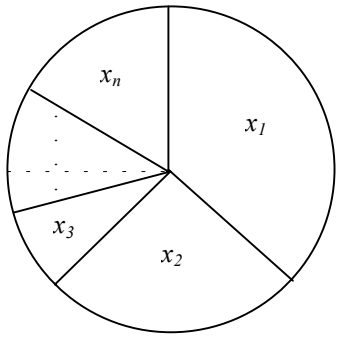
2.5.1. Szelekció, reprodukció

A szelekciós operátor probléma-független 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 [Jelasity99]. Ennél az algoritmusnál a populáció minden egyedét a rulettkerék egy-egy mélyedése képviseli, amely egyenesen arányos az egyed rátermettségével (2.4. ábra). A rulettkerék módszer e változatát még **költség szelekció**nak (*cost szelection*) is nevezzük [Generation03a]. Az egyedeket véletlenszerűen választjuk ki figyelembe véve a rátermettségi értékeket, általában a rátermettebb egyedek lesznek kiválasztva.



2.4. ábra. A rulettkerék módszer. Az $x_1, x_2, x_3, \ldots, x_n$ 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 rátermettségi 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*) [Generation03a].

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 [Generation03a]. Ha a verseny szelekció esetében kiválasztott csoport csak két egyedet tartalmaz akkor az operátort **pár-verseny szelekció**nak (*binary tournament selection*) nevezzük [Jelasity99].

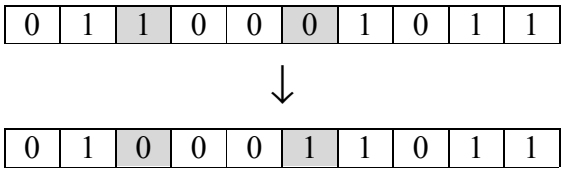
A fent leirt szelekciós operátorokat tetszőleges számszor alkalmazhatjuk, annak függvényében, hogy hány egyedre van szükségünk az új populációban. Ha a régi populáció összes tagját le akarjuk cserélni, akkor az adott szelekciós algoritmust annyiszor hajtjuk végre amennyi egyedet tartalmaz a populáció. Ilyenkor **nem átfedő populációk**ról beszélünk [Pécsy98]. Ha viszont a populációnak csak egy bizonyos százalékát cseréljük le akkor **átfedő populációk** jönnek létre, ami a **generációs rés** (*generation gap*) effektusban nyilvánul meg [DeJong75]. 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 ki, ahol N a populáció mérete. A lecserélendő egyedeket általában a legkevésbé rátermett kromoszómák közül választják ki.

2.5.2. 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 levő allél értékeket változtatunk meg. A 2.5. 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.

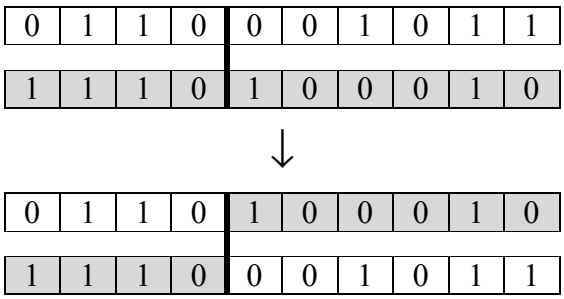


2.5. ábra. A kromoszóma harmadik és hatodik génje mutálódik.

2.5.3. Keresztezés, rekombináció

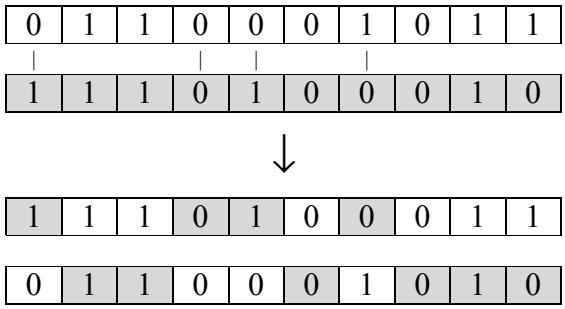
A rekombinációk (kereszteződés) több kiindulási megoldásból (szülőből) állítanak elő újabb megoldást, tehát egy bináris operátor. A célja az, hogy különböző egyedek tulajdonságait ötvözve újabb, esetleg rátermettebb egyedeket hozzunk létre. Az egyik ilyen rekombinációs operátor az **egypontos keresztezés** (*1-point crossover*). Véletlenszerűen választunk egy **kereszteződési pont**ot (*crossover point*) és a kapott két fél kódból új megoldást rakunk össze. A 2.6. 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.



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

A másik rekombinációs operátor az **egyenletes keresztezés** (*uniform crossover*). Itt a mutációkhoz hasonlóan, kiválasztott pozíciókon a kiindulási kódok betűket cserélnek. Egy pozíció kiválasztásának a valószínűsége rendszerint 0,5, ami jóval nagyobb mint a mutáció esetében. A 2.7. ábra szemlélteti egy példán keresztül ezt az operátort.



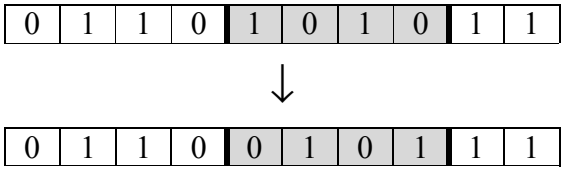
2.7. ábra. Az egyenletes keresztezés. A szülő kromoszómák közötti vonalak jelölik azokat a géneket, ahol az allél értékek kicserélődnek.

A keresztezési operátoroknak számos változata 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 [Michalewicz96]. Ugyancsak speciális keresztezési és mutációs operátort alkalmaznak az utazóügynök feladat (***T**raveling **S**alesman **P**roblem – TSP*) esetében, ahol szintén több változata létezik ezeknek az operátoroknak [Michalewicz96].

2.5.4. Inverzió

Az inverzió unáris operátor, egy kromoszómát, a mutációhoz hasonlóan (csak egy szülő információit tartalmazza) átalakít, úgy, hogy két véletlenszerűen kiválasztott gén közötti alél értékeket megfordítja [Holland92]. A jobb megértés érdekében tekintsük a következő általános szabályt. Legyen $A = a_1 a_2 ... a_l$ egy tetszőleges kromoszóma és $x_1^{'}$, $x_2^{'}$ két véletlenszerűen kiválasztott szám a $\{0,1,...,l+1\}$ halmazból, amelyek a gének indexének felelnek meg. Ha $x_1 = \min\{x_1^{'}, x_2^{'}\}$ és $x_2 = \max\{x_1^{'}, x_2^{'}\}$, akkor az új kromoszóma inverzió után a következőképpen néz ki: $a_1 ... a_{x_1} a_{x_2-1} ... a_{x_1+1} a_{x_2} ... a_l$. Az illusztráció kedvéért tekintsük a 2.8. ábrát.

pl. (a_1, a_2, a_3, a_4, a_5, a_6) egy egyed a populációból, a két véletlen szám legyen 6 és 2. Ekkor a 2 és 6 indexek között levő géneket felcserélve az (a_1, a_2, a_5, a_4, a_3, a_6) leszármazottat kapjuk.



2.8. ábra. Az inverziós operátor. A vastag vonalak közötti gének értékeit cseréljük fel.

Általában az algoritmusnak paraméterként megadhatjuk a rekombinációs, mutációs és inverziós valószínűségeket azért, hogy az egyedek csak egy bizonyos százalékára alkalmazzuk az előbb említett operátorokat.

2.5.5. Együtt-fejlődés és vándorlás

A genetikus algoritmusok a sok jó tulajdonságok mellett rendelkeznek egy pár hátulütővel. Az egyik ilyen probléma az, hogy beragadhatnak a lokális optimumba. Más szavakkal megkapnak egy elfogatható megoldást, de nem a legjobbat. A probléma kiküszöbölésére számos megoldás van (lásd a 2.5.1. és 2.5.2. fejezeteket), az együtt-fejlődés (*co-evolution*) vagy koevolúció is erre hivatott.

A fenti elv értelmében egyidejűleg több populációt is ki lehet fejleszteni párhuzamosan, azaz szálakon (*thread*) futó genetikus algoritmusok által. Egy többprocesszoros rendszeren, amely támogatja az SMP-t (*Simmetric Multiprocessing*) ez a feladat nem jelent plusz időt. A különböző populációk közötti adatcserét a kromoszómák vándorlásával oldjuk meg, biztosítva ily módon a fontos génanyagot [Generation03a].

2.6. Algoritmus

A genetikus algoritmus struktúrája nagyon egyszerű. Egy vázlatos és általános formában megadott algoritmus a 2.9. ábrán látható [Jelasity99].

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

2.9. ábra. A GA egy sematikus algoritmusa.

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).

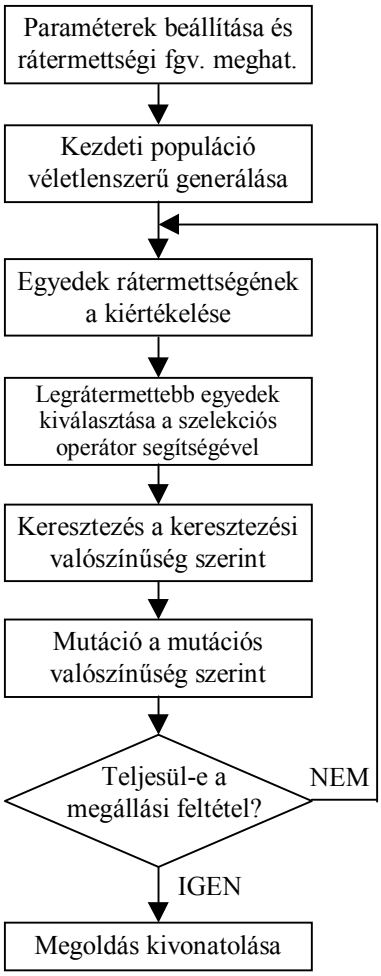
A `Kilépés(`P_t`)` függvény a már kiértékelt magoldások számát vizsgálja meg. Ha elértük a megengedett kiértékelések számát (tipikusan 1000 és 500000 között van), akkor az algoritmus leáll.

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 [Jelasity99].

A GA általános alakját folyamatábra formájában a 2.11. ábra szemlélteti. Összegezve az eddig leírtakat és kibővítve a fenti sematikus algoritmust a GA egy újabb algoritmusához jutunk, amit a 2.10. ábra mutat be.

```
Keresési terület azonosítása
Egyedek kódolása, genetikus reprezentáció
Megoldások értékelésére szolgáló rátermettségi függvény
meghatározása
Paraméterek beállítása (populáció mérete, rekombinációs,
mutációs valószínűségek, iterációk/generációk száma)
Kezdeti populáció létrehozása
Amíg a megállási feltétel nem teljesül
  Amíg a régi populáció rátermettebb mint az új
    Új populáció kiválasztása a szelekciós operátor segítségével
    Keresztezés, mutáció az új populációban
    Régi és új populáció rátermettségének a kiértékelése
    Összpopuláció (régi és új) csökkentése a fordított előjelű
    szelekció segítségével
  Amíg vége
Amíg vége
```

2.10. ábra. A GA egy részletesebb, kibővített algoritmusa.



2.11. ábra. A GA egy tipikus folyamatábrája.

2.7. Sémaelmélet, a genetikus algoritmusok matematikai alapjai

Az előző fejezetekben láthattuk, hogy milyen egyszerű szerkezete van egy genetikus algoritmusnak és mennyire egyszerű operátorokat használ. Ennek ellenére mégis egy robusztus, hatékony és széles körben elterjedt optimalizáló algoritmus. Mi ennek az alapja és mivel magyarázható ez a jelenség?

A genetikus algoritmus csak a célfüggvény adott pontjaiban vett kiértékelésekből származó információkkal rendelkezik. Ennek következtében a binárisan reprezentált kromoszómák fontos információk hordozói. A szemléltetés érdekében lássuk a következő példát: keressük meg az $f(x)=x^2$ függvény maximumát a [0,31] intervallumon [Pécsy98]. A kromoszómákat 5 bites stringekként kódoltuk. Tekintsük a 2.1. táblázatban megadott egyedeket és az általuk elért teljesítményeket. A rátermettségeket úgy számoltuk ki, hogy behelyettesítettük a célfüggvénybe a tízes számrendszerbe átalakított bitsorozatokat.

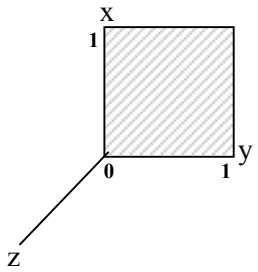
2.1. táblázat. A populáció 4 tetszőleges, eltérő rátermettségű egyede.

Bitsorozat	Rátermettség
01101	169
11000	576
01000	64
10011	361

Észrevesszük, hogy a stringek között bizonyos hasonlóságok vannak. Ha jobban megfigyeljük, akkor rájövünk, hogy bizonyos mintákra illeszkedő stringek rátermettsége jóval nagyobb mint a többié, nevezetesen azokra a mintákra, ahol az 1-es az első pozíción helyezkedik el. Ez annyira fontos észrevétel, hogy ez képezi a genetikus algoritmusok működésének az alapját: hasonlóságokat keresni a stringek között és megnézni, hogy ezek a hasonlóságok hogyan viszonyulnak a rátermettséghez.

A fenti észrevételre Holland jött rá és a [Holland92]-ben írta le matematikailag, bevezetve a hasonlósági minta vagy **séma** (*schema*) fogalmát. A séma egy l hosszúságú bit stringből alló minta, ami a stringek egy részhalmazát határozza meg, így a string egy hasonlósági osztályt reprezentál. Kiterjeszti a string ábécéjét egy * karakterrel, ami tetszőleges 0 vagy 1-es számjegyet helyettesít.

A mértani jelentése a sémának egy l dimenziós **hipersík** (*hyperplane*) [Generation03b]. Például legyen az S=**0 séma, aminek a hossza 3. Következik, hogy ez egy 3-dimenziós hipersíknak felel meg, amit a 2.12. ábra szemléltet.



2.12. ábra. Az S=**0 séma egy lehetséges hipersíkja a 3-dimenziós térben.

Példának okáért az S=**1***0*** séma olyan bit stringeket reprezentál, amelyek egyessel kezdődnek és az ötödik pozíción egy nullás van. Az S séma a következő stringekre illeszkedik: **1**011**0**1, **1**011**0**0, **1**1000**0** stb. Az S=****** séma (szintén l hosszúságú) 2^l darab kromoszómára illeszkedik. Általánosan, ha egy sémában a *-ok száma egyenlő k-val akkor a séma 2^k darab stringre fog illeszkedni.

Ezek után fogalmazzuk meg a séma értelmezését és a sémával kapcsolatos fogalmak értelmezéseit.

1. **Értelmezés**: Egy séma az összes olyan kromoszómát képviseli, amely illeszkedik az adott egyedre a *-on kivüli pontokon.

Jelölés: S

pl.: S_1=101**101**

S_2=*111000111

2. **Értelmezés**: A séma rendje (*defined bits*) a rögzített bitek számát jelenti a sémában.

Jelölés: $o(S)$

pl.: $o(S_1)$=6

$o(S_2)$=9

3. **Értelmezés**: A séma definíciós hossza (*defined length*) a séma első és utolsó rögzített bitek közötti távolságát jelenti.

Jelölés: $\delta(S)$

pl.: $\delta(S_1)$=8-1=7

$\delta(S_2)$=10-2=8

4. **Értelmezés**: A séma rátermettségi függvénye azoknak a kromoszómáknak a rátermettségi függvényének az átlaga, amelyekre a séma illeszkedik.

Jelölés: $f(S,t)$

A fenti értelmezés szerint a séma rátermettsége:

$$f(S,t)=\frac{\sum_{i=1}^{p} f(v_i)}{p},$$

ahol az S séma p darab kromoszómára illeszkedik: $(v_1,...,v_p)$; $f(v_i)$, $i=\overline{1,p}$, pedig a kromoszómák rátermettsége, ahol $f(v_i)>0$, $\forall i=\overline{1,p}$.

2.7.1. Szelekció hatása a sémára

Legyen $F(t)=\sum_{i=1}^{popméret} f(v_i)$ a populáció rátermettsége (összrátermettségi függvény) a t-ik időpillanatban (generációban), $P_i=\frac{f(v_i)}{F(t)}$ pedig az i-ik kromoszómának és a populáció rátermettségének az aránya, ahol $P(t)=(v_1,...,v_{popméret})$.

Jelöljük $\xi(S,t)$-vel azoknak a kromoszómáknak a számát, amelyek illeszkednek a sémára. Ekkor:

$$\xi(S,t+1) = \xi(S,t) \cdot popméret \cdot \frac{f(S,t)}{F(t)} \quad (2.1)$$

Az átlag rátermettségi függvény a következő alakban irható fel:

$$\overline{F}(t) = \frac{F(t)}{popméret}.$$

Innen kifejezve a *popméret*-t és behelyettesítve a 2.1-es egyenletbe az úgynevezett **reproduktív sémanövekedési egyenlet**et kapjuk:

$$\xi(S,t+1) = \xi(S,t) \cdot \frac{f(S,t)}{\overline{F}(t)} \quad (2.2)$$

Azoknak a kromoszómáknak a száma amelyekre az adott séma illeszkedik az egyenesen arányos a séma rátermettségével és a populáció átlagos rátermettségi arányával. Ismétlések során az átlagon felüli kromoszómák (amelyek illeszkednek a sémára) száma nő, ha kisebb akkor csökken és, ha egyenlő akkor nem változik.

$f(S,t) = \overline{F}(t) + \varepsilon \cdot \overline{F}(t)$

$\xi(S,1) = \xi(S,0) \cdot (1+\varepsilon)$

$\xi(S,2) = \xi(S,1) \cdot (1+\varepsilon)$

. . .

$\xi(S,t) = \xi(S,0) \cdot (1+\varepsilon)^t$

Jól látszik, hogy a kromoszómák száma exponenciálisan nő, ha a séma rátermettsége átlagon felüli.

2.7.2. Keresztezés hatása a sémára

pl.: S_1=*****111**

S_2=01******11

Legyen V_1=**1101111101** és V_2=0100001111 két kromoszóma amely illeszkedik az S_1 illetve S_2 sémákra (a rögzített biteket az aláhúzás vonal jelöli). Alkalmazzuk a keresztezést a két kromoszómára a 4-es keresztezési pontnál. A leszármazottak: V_1'=**1101**001111 és V_2'=0100**111101**. Észrevesszük, hogy ha a keresztezési pont a rögzített gének között van akkor a séma nem éli túl a keresztezést.

Ha egy tetszőleges V kromoszóma hossza m, akkor $m-1$ darab kereszteződési pozíció lehetséges. Felírhatjuk a séma elpusztulási valószínűségét: $p_S = \frac{\delta(S)}{m-1}$. Innen következik, hogy a túlélési valószínűség a kiválasztás után: $1-\frac{\delta(S)}{m-1}$. Ha p_k a keresztezés valószínűsége, akkor a túlélési valószínűség a kiválasztáskor a következőképpen módosul: $p_t(S) = 1 - p_k \cdot \frac{\delta(S)}{m-1}$.

Kibővítve a 2.2-es egyenletet az imént felirt túlélési valószínűséggel, a következő összefüggést (a **reprodukciós egyenlet a keresztezés figyelembe vételével**) kapjuk:

$$\xi(S,t+1) = \xi(S,t) \cdot \frac{f(S,t)}{\overline{F}(t)} \cdot \left[1 - p_k \cdot \frac{\delta(S)}{m-1}\right] \quad (2.3)$$

2.7.3. Mutáció hatása a sémára

pl.: S_1=****111***

Legyen V_1=0011<u>111</u>011 egy kromoszóma amely illeszkedik az S_1 sémára (a rögzített biteket az aláhúzás vonal jelöli). Alkalmazzuk a mutációt a kromoszómára a 8-as mutációs pozíciónál. A leszármazott: V_1'=0011<u>111</u>**1**11. Ha viszont a mutáció a 7-es pozíción lett volna, akkor a kromoszóma így néz ki: V_1'=0011<u>11</u>**0**011. Észrevesszük, hogy ha a mutációs pozíció a rögzített génekre esik akkor a séma nem éli túl a mutációt.

Ha p_m a mutáció valószínűsége (ahol $p_m << 1$), akkor a túlélési valószínűség a kiválasztáskor a következőképpen irható fel: $p_t(S) = (1-p_m)^{o(S)} \approx 1 - o(S) \cdot p_m$.

Kibővítve a 2.3-as egyenletet a felirt túlélési valószínűséggel, az úgynevezett **sémanövekedési egyenlet**et kapjuk:

$$\xi(S,t+1) = \xi(S,t) \cdot \frac{f(S,t)}{\overline{F}(t)} \cdot \left[1 - p_k \cdot \frac{\delta(S)}{m-1}\right] \cdot [1 - o(S) \cdot p_m]$$

Ezek után kijelenthetjük a Holland által kidolgozott séma tételt.

Tétel: A rövid (definíciós hossz), kisrendű, átlagon felüli (rátermettség) sémák exponenciálisan növekvő részt kapnak a genetikus algoritmus egymást követő generációiban.

A tételben szereplő sémákat nagy jelentőségük miatt **építő blokkok**nak (*building blocks*) is szokták nevezni ([Pécsy98], [Generation03b]), amelyek a generációk során mind nagyobb számban lesznek jelen a populációban és az optimum is ezek közül kerül ki. A

genetikus algoritmusnak éppen ez a feladata, hogy megkeresse, feltárja, kihangsúlyozza és kitenyéssze ezeket az építő blokkokat vagy építősémákat (ami tulajdonképpen egy feladat jó megoldásainak a képviselői) egy roppant párhuzamos módon. Ez a folyamat szimultán módon történik a populáció minden egyes sémájára és a genetikus algoritmus **implicit párhuzamosság**ának (*implicit parallelism*) nevezzük. Kimutatható, hogy egy N elemű populáció esetén lépésenként legalább N^3 darab sémafeldolgozás történik, ami az N darab célfüggvény-kiértékeléshez képest igen jó eredmény [Pécsy98].

2.7.4. Kódolási alapelvek

A korábbiakban láthattuk, hogy milyen fontos lehet a megoldás reprezentálása, a kromoszóma kódolása, hiszen a genetikus algoritmus akkor működik jól, ha vannak építő blokkok. A jó kódolás megadására nincsenek szabályok, csak alapelvek, amelyeket jó betartani és különböző technikák, amivel a meglevő kódoláson, leképzésen javítani lehet. Két egyszerűbb elv [Pécsy98]:

1. A jelentőségteljes építő blokkok elve: lehetőleg olyan kódokat válasszunk ahol a rövid sémák fontos információ hordozok. Ennek a leellenőrzése nagyon nehéz, így ennek az elvnek a betartása szinte lehetetlen.
2. A minimális ábécé elve: használjunk minimális ábécét, lehetőleg binárisat, hiszen így az egy string által reprezentált sémák aránya jóval nagyobb. Ez az elv aránylag könnyen betartható, ugyanis nagyobb méretű ábécék átkódolhatók bináris ábécére.

2.8. Kapcsolat a metaheurisztikákkal

A genetikus algoritmus a probléma-független metaheurisztikák osztályába tartozik, amelyek közül a legismertebbek a **szimulált hűtés** (*simulated annealing*), a **tabukeresés** (*tabu search*) és a **hegymászó** (*hill climber*) algoritmusok. Mindegyik módszer globális, vagyis tartalmaznak valamilyen technikát a lokális optimumokba való “beragadás” ellen. A metaheurisztika egy algoritmus váz, amit egy konkrét problémára való alkalmazás előtt ki kell tölteni a probléma leírásával, esetleg további, a problémára jellemző ismeretekkel és az algoritmus működését befolyásoló paraméterekkel.

A fennebb leirt genetikus algoritmust (lásd 2.6. fejezetet) csak kis mértékben kell módosítani ahhoz, hogy az előbb felsorolt heurisztikákra is alkalmazni lehessen [Jelasity99].

2.8.1. A szimulált hűtés

A szimulált hűtés a statisztikai mechanikából származtatott sztochasztikus számítási technika és terjedelmes optimalizálási feladatok globális minimális költségét hivatott megkeresni, vagy legalább is azt jól megközelíteni [Davis87].

[Kirkpatrick83] és társai voltak az elsők akik alkalmazták ezt a statisztikai fizikából vett szimulációs technikát a kombinatorikus optimalizálási feladatokra. Ezt a módszert elsősorban a huzal vezetésre (*wire routing*) és a magas fokon integrált áramkörök (***V**ery **L**arge **S**cale **I**ntegration – VLSI*) tervezésére használták, amikor is a cél az elektronikus komponensek elhelyezésének az optimalizálása volt.

A szimulált hűtés struktúrája [Michalewicz96] alapján a 2.13. ábrán látható.

```
t ← 0
T kezdeti hőmérséklet inicializálása
Válassz véletlenszerűen egy v_c stringet
Értékeld ki v_c-t
repeat
  repeat
    Válassz ki egy új stringet v_c környezetéből
    megváltoztatva v_c egyetlen bitjét
    if f(v_c)<f(v_n)
      then v_c ← v_n
      else if random[0,1)<exp{(f(v_n)-f(v_c))/T}
        then v_c ← v_n
  until (befejeződési feltétel)
  T ← g(T,t)
  t ← t + 1
until (megállási feltétel)
```

2.13. ábra. A szimulált hűtés algoritmusa.

A (befejeződési feltétel) megnézi, hogy beállt-e a „hőmérsékleti egyensúly”, azaz a kiválasztott új string valószínűségi eloszlása elérte-e a Boltzmann eloszlást. Általában ezt a ciklust egy előre meghatározott számszor végzik el. A T hőmérsékletet lépésekben csökkentik, amíg el nem ér egy alacsony értéket, amire a (megállási feltétel) figyelmeztet, vagyis a rendszer „megfagyott”. A végeredmény az algoritmus befejezésével a v_c változóban lesz.

A szimulált hűtés esetében a populáció egyelemű és a kereső operátor a mutáció. Az UjPopuláció(P_t',P_t) (lásd a 2.6. fejezetet) a következőképpen változik meg: ha az új megoldás rosszabb, mint a régi, akkor is elfogadjuk egy bizonyos valószínűséggel, amit a hőmérséklet paraméter ad meg. Ha a hőmérséklet nulla, csak akkor fogadjuk el, ha nem rosszabb mint a régi megoldás [Jelasity99].

2.8.2. A tabukeresés

A tabukeresés egy lokális keresési algoritmus, iteratív javító algoritmus. Olyan probléma típusokra alkalmazható, ahol a megoldások egy gráf csomópontjain helyezkednek el. Egy probléma megengedett megoldásai (állapotai) közül keresi meg a legjobbat, vagy pedig csak megközelíti azt, és így a megoldás csak lokálisan lesz a legjobb.

A tabukeresés egy lehetséges algoritmusát a 2.14. ábra szemlélteti és [Kis99] nyomán készült.

```
s ← (kezdeti megengedett megoldás kiválasztása)
s* ← s
k ← 1
T ← üres halmaz
while not (megállási feltétel) do
  s' ← legjobb elem s szomszédai - T halmazból
  T frissítése s'-el
  s ← s'
  if s' jobb mint s* then s* ← s'
  k ← k + 1
endwhile
```

2.14. ábra. A tabukeresés algoritmusa.

Az s az aktuális megengedett megoldást míg s^* a keresés során talált legjobb megoldást jelöli és az algoritmus leállása után ebben lesz a végső megoldás is. A T halmaz a tabuhalmaz, ami tartalmazza a legutóbb megvizsgált megoldásokat. Mivel nem lehet az összes utoljára megvizsgált műveletet eltárolni, így egy bizonyos idő után a legrégebbit kitörli. Ezért nevezik a tabuhalmazt még rövid távú memóriának (*recency based memory*) is. A tabuhalmazt a gyakorlatban egy előre megadott rögzített hosszúságú FIFO adatszerkezettel oldják meg. A `(megállási feltétel)` a következő feltételek egyike lehet: k túllép egy bizonyos határt, vagy k túllép egy bizonyos határt s^* utolsó módosítása óta, vagy az s szomszédai - T halmaz üres.

A tabukeresésnél is egyelemű a populáció és a kereső operátor szintén a mutáció. Itt is az `UjPopuláció(`P_t'`,`P_t`)` (lásd a 2.6. fejezetet) függvény az eltérő. Az új populáció kiválasztásához megnézzük, hogy az új elem nem szerepel-e a tabulistában. Ha, igen akkor elvetjük, ellenben, ha nem rosszabb mint a régi akkor elfogadjuk. Az új megoldás a tabulistába kerül és a legrégebbi elem törlődik a listából [Jelasity99].

2.8.3. A hegymászó algoritmus

Nem sok verziója létezik a hegymászó algoritmusnak. Nagyrészt abban különböznek, hogy hogyan választják ki az új elemet vagy stringet (mivel genetikus algoritmusokról van szó) a keresési térből annak érdekében, hogy ezt összehasonlítsa az aktuális legjobb értékkel.

Egy egyszerű iterált hegymászó algoritmus (legmeredekebb felszállás algoritmusa – *steepest ascent hillclimbing*) a [Michalewicz96] monográfia szerint a 2.15. ábrán bemutatott módon néz ki.

```
t ← 0
repeat
  lokális ← FALSE
  Válassz véletlenszerűen egy v_c stringet
  Értékeld ki v_c-t
  repeat
    Válassz ki n darab új stringet v_c környezetéből
    megváltoztatva v_c egyetlen bitjét
    Válaszd ki a v_n stringet az új stringek halmazából,
    amelynek az f célfüggvény-értéke a legnagyobb
    if f(v_c)<f(v_n)
      then v_c ← v_n
      else lokális ← TRUE
  until lokális
  t ← t + 1
until t = MAX
```

2.15. ábra. A hegymászó algoritmus.

Érdekes megjegyezni, hogy a fenti algoritmus egyszeri iterációjának sikere a kezdeti stringtől függ. Ha ez a string „nagyon rossz”, kicsi a függvény értéke, akkor egyetlen bit megváltoztatásával (egyből nulla és nullából egy) nem tudunk nagy javulást elérni, legfennebb a lokális optimumot kapjuk meg.

A sztochasztikus hegymászó algoritmusnál a populáció szintén egyelemű és a keresési operátor itt is a mutáció. Az új megoldás felváltja a régit, ha ugyanolyan rátermett vagy rátermettebb. Észrevehető, hogy a hegymászó egy nulla hőmérsékleten futó szimulált hűtés, vagy nulla hosszúságú tabulistás tabukeresés, sőt egyelemű populációt használó elitista genetikus algoritmusként is felfogható, annak ellenére, hogy ezek a módszerek egymástól független természeti folyamatok modelljei [Jelasity99].

2.8.4. Metaheurisztikák összehasonlítása egy példán keresztül

A keresési tér 30 hosszúságú v bináris stringek halmaza. Az f célfüggvény, aminek meg kell keresni a maximumát: $f(v) = |11 \cdot one(v) - 150|$, ahol $one(v)$ függvény megadja a v bináris string által tartalmazott egyesek számát [Michalewicz96].

A függvény nagyon egyszerű, de mégis nehézséget okoz még a fejlett kereső algoritmusoknak is, amelyeket ebben a fejezetben ismertünk meg. A nehézség abban nyilvánul meg és ami pont a függvény egy érdekes tulajdonsága, hogy van egy globális és lokális maximuma:

$v_g = (11111111111111111111111111111111)$, ekkor $f(v_g) = |11 \cdot 30 - 150| = 180$

$v_l = (00000000000000000000000000000000)$, ekkor $f(v_l) = |11 \cdot 0 - 150| = 150$

Jól látszik, hogy a hegymászó algoritmus bizonyos esetekben „beragad" a lokális optimumba, például akkor ha a kiinduló, kezdeti string 13 darab egyest tartalmaz, ekkor a függvényérték 7. A globális optimum felé való elmozdulás érdekében, az algoritmus növeli az egyesek számát (olyan változót választ ki a következő lépésben amiben több egyes van), de ez a függvényérték 4-re való csökkenéséhez vezet. Viszont, ha csökken az egyesek száma akkor a függvényérték 18-ra nő és a keresést rossz irányba tolja el, a lokális optimum felé.

A szimulált hűtés már könnyebben megbirkózik ezzel a feladattal, ugyanis az algoritmus bizonyos valószínűségekkel elfogad rosszabb megoldásokat is, ami kisegíti a lokális optimumból. Lássuk az előző példát: tegyük fel, hogy a v_c változó 12 darab egyest tartalmaz míg a v_n 13-at. Ekkor annak a valószínűsége, hogy az algoritmus elfogadja az új stringet, ami a globális optimum irányába tolná a keresést: $p = \exp\{(f(v_n) - f(v_c))/T\} = \exp\{(7-18)/T\}$ és ha T=20, akkor $p = e^{-\frac{11}{20}} = 0{,}57695$, ami azt jelenti, hogy az elfogadás esélye nagyobb mint 50%.

Ezzel a példával igazán jól szemléltethető a genetikus algoritmus felsőbbrendűsége. Viszonylag kevés iterációval nagyon gyorsan megkapja a globális maximumot, hiszen a genetikus operátorok segítségével (elsősorban a keresztezés) sokkal jobb leszármazottakat tud előállítani.

2.9. Genetikus algoritmusok alkalmazása

A GA-t általában olyan problémára érdemes használni, amelynek nagy a keresési tere, nem ismert a struktúrája vagy nem áll rendelkezésre terület-specifikus tudás.

Sok alkalmazása van a gépi tanulás területén, mivel minden tanulási probléma megadható optimalizálási feladatként [Goldberg89].

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 feladatoknál sokkal gyorsabban konvergál mint a hiba hátraterjedésének (*back-propagation*) a módszere. Neuronháló szerkezetének 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 – lásd a 2.2. és 2.12. fejezeteket) mégsem ajánlatos a használata ebben az esetben, mivel neuronhálókkal kombinálva hihetetlenül lassú lehet [Generation03d].

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 [Military01], a brókercégek pedig GA által vezérelt programokat használnak a tőzsdei események előrejelzésére.

2.9.1. Alkalmazás: egyszerű függvény optimalizálása

Keressük meg az alábbi függvény globális maximumát [Michalewicz96]:

$f(x) = x \cdot \sin(10\pi \cdot x) + 1,\ x \in [-1,2]$

2.9.1.1. Reprezentáció

Egy bináris vektort használunk kromoszómaként, aminek a hossza függ a várt eredmény pontosságától. Mivel 6 tizedes pontossággal kell az eredmény és az intervallum hossza 3=2–(–1), következik, hogy a vektor hossza 22 bit:

$2097152=2^{21}<3000000<2^{22}=4194304$.

2.9.1.2. Kódoló függvény

A $\langle b_{21}b_{20}...b_0 \rangle$ bináris stringet egy [–1,2] közötti x valós számmá alakítja két lépésben:

- átalakítja 2-esből 10-es számrendszerbe:

  $\left(\langle b_{21}b_{20}...b_0 \rangle\right)_2 = \left(\sum_{i=0}^{21} b_i \cdot 2^i\right)_{10} = x'$

- majd egy valós számmá:

  $x = -1 + x' \cdot \frac{3}{2^{22}-1}$, ahol –1 az értelmezési tartomány alsó határa.

pl. (10001011101101010001111) a 0,637197 valós számot ábrázolja.

2.9.1.3. Kezdeti populáció

22 bites kromoszómákból készítünk egy populációt, és a géneket véletlenszerűen inicializáljuk.

2.9.1.4. Értékelő (rátermettségi) függvény

$eval(v) = f(x)$, ahol a v bináris vektor (kromoszóma) az x valós számot képviseli, míg f az optimalizálandó függvény.

pl. $v_1 = (1000101110110101000111)$ $x_1 = 0{,}637197$ $eval(v_1) = 1{,}586345$

$v_2 = (0000001110000000010000)$ $x_2 = -0{,}958973$ $eval(v_2) = 0{,}078878$

$v_3 = (1110000000111111000101)$ $x_3 = 1{,}627888$ $eval(v_3) = 2{,}250650$

Amint látszik ebben az esetben a legjobb kromoszóma a 3-as, mivel a legnagyobb a rátermettsége.

2.9.1.5. Genetikus operátorok

Mutáció: tegyük fel, hogy a v_3-as vektor 5. génje mutálódik, ekkor:

$v_3' = (1110100000111111000101)$ $x_3' = 1{,}721638$ $f(x_3') = -0{,}082247$.

Ez azt jelenti, hogy a kromoszóma értéke nagyban csökkent az eredetihez képest. Ha viszont a 10. gén mutálódik, akkor a kromoszóma értéke növekszik:

$v_3'' = (1110000001111111000101)$ $x_3'' = 1{,}630818$ $f(x_3'') = 2{,}343555$.

Keresztezés: a v_2 és v_3 kromoszómák keresztezési pontja 5, az utódok:

$v_2' = (00000 \mid 00000111111000101)$ $eval(v_2') = f(-0{,}998113) = 0{,}940865$,

$v_3' = (11100 \mid 01110000000010000)$ $eval(v_3') = f(1{,}666028) = 2{,}459245$.

A második utód rátermettebb mint bármely két szülő.

2.9.1.6. Paraméterek

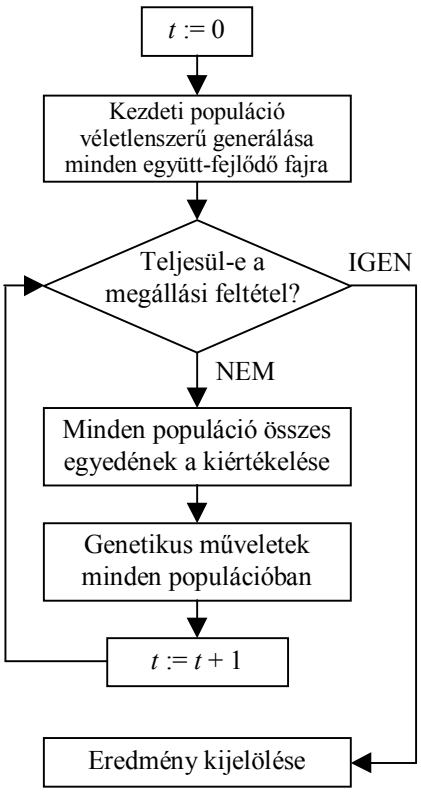
Populáció mérete pop_size=50, keresztezés valószínűsége p_c=0.25 és a mutáció valószínűsége p_m=0.01.

2.9.1.7. Kísérleti eredmények

150 generáció után a legjobb kromoszóma: $v_{\max} = (11110011010000100000101)$, ami az $x_{\max} = 1{,}850773$ valós értéknek felel meg és a függvény értéke ebben a pontban valamivel több mint 2,85, ami pont a várt eredmény [Michalewicz96].

2.10. Együtt-fejlődés

A 2.5.5. fejezetben már használtuk ezt a fogalmat két vagy több együttműködő populáció kifejlesztésére. Itt is két vagy több egymástól független populáció párhuzamos fejlesztésére használjuk, csak ezek a populációk genetikailag eltérő ellenfelek szerepét töltik be és a cél az, hogy egyik a másiknak környezete legyen, egyre rátermettebb egyedek kifejlesztésére késztetve egymást. A 2.16. ábra vázolja az együtt-fejlődés folyamatábráját [Koza92] alapján.



2.16. ábra. Az együtt-fejlődés egy vázlatos folyamatábrája. A *t* változó a generációk számát tartja nyilván. A megállási feltétel lehet az, hogy *t* elért egy bizonyos számot.

Példának okáért, ha egy sakk játékost akarok kifejleszteni, akkor sokkal hatékonyabb stratégiát kapok, ha mind a két játékost (fehér és fekete) párhuzamosan fejlesztem. Egymással ezeket szembeállítva, arra kényszerítem a genetikus algoritmust, hogy egy sokkal jobb

stratégiát tanuljon meg. Ezzel azt is biztosítjuk, hogy a populációk nem fogják a másiknak a hibáit kihasználni, hiszen ezek a populációk folyamatosan változtak, tökéletesedtek a generációk során, biztosítva ily módon egy erős ellenfelet és környezetet, amelyhez alkalmazkodniuk kell.

Ennek az ötletnek, technikának a megfogalmazásakor a kutatók szintén a természetből ihletődtek: a **gazda** (*host*) és **parazita** (*parasite*) vagy **ragadozó** (*predator*) és **áldozat** (*prey*) élőlények kölcsönös kapcsolata. Általanosan fogalmazva, a két különböző faj között létrejövő folytonos küzdelem, csata az úgynevezett „**fegyverkezési versengés**ben” (*arms race*) nyilvánul meg, hiszen arra kényszerítik egymást, hogy mind jobban és jobban alkalmazkodjanak az ellenfélhez. Ennek a mintájára Koza megfogalmazott egy üldöző-menekülő játékot (*A Differential Pursuer-Evader Game*) [Koza92].

Hillis 1980-ban elsőként használta az együtt-fejlődést rendező hálók optimalizálására. A feladat a híres „n=16” rendezés volt, azaz 16 elemű lista rendezése. A feladat felfogható úgy is mint egy játék, ahol az állapottér 16! különböző konfigurációt, stratégiát tartalmaz. A cél az volt, hogy a rendezéshez szükséges helycserék számát csökkentse. Habár az általa elért minimális számú összehasonlítás (61 darab) eggyel több volt mint az 1969-ben kidolgozott algoritmussal, mégis bebizonyította a koevolúció hasznosságát. E technika nélkül 65 darab összehasonlításra volt szükség az általa kifejlesztett GA alapú programmal. [Generation03e].

2.11. Evolúciós programozás

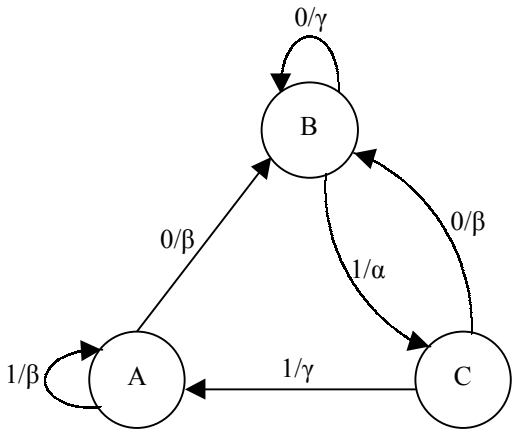
Az evolúciós programozás – röviden EP – az evolúciós számítások egyik legkorábbi kutatási területe, aminek a megalkotói Fogel, Owens és Walsh [Fogel66]. Az intelligens viselkedés (*behaviour*) feltételezi egy rendszer környezetének (*environment*) az előrejelzését és ezeknek az előrejelzéseknek a megfelelő válaszokká való alakítását. A környezet egy véges ábécéből vett szimbólumok sorozatával van leírva. Az EP feladata egy olyan algoritmus kifejlesztése, amely ezekkel a szimbólumokkal dolgozik és amely egy olyan kimeneti szimbólumot állít elő ami a környezet következő szimbólumát jelenti. Egy értékelő függvény megnézi, hogy ez a kimeneti szimbólum mennyire tudta előre jelezni a környezet szimbólumát, azaz meghatározza a különbséget a kimeneti – és az aktuális szimbólum között. Ezeknek a folyamatoknak az elvégzésére a véges automaták (***F**inite **S**tate **M**achine – FSM*) voltak a legmegfelelőbbek.

A cél így véges automaták automatikus fejlesztése lett. Az EP is a természetes szelekció elvét követi, így nagyon hasonlít a genetikus algoritmusokhoz, csak azzal a

különbséggel, hogy a kromoszómák nem bitsorozatok, hanem automaták (lásd a [Fogel00] könyv *3.3 Evolutionary Programming* fejezetét).

2.11.1. Véges állapotú automaták

A véges állapotú automata (Mealy automata) egy átalakító, amelynek a be- és kimeneti szimbólumok egy véges ábécé elemei és véges számú különböző belső állapota van. Egy állapot és a megfelelő bemeneti szimbólum meghatározza bármely véges állapotú gép viselkedését, azaz milyen állapotba megy át és közben milyen kimeneti szimbólumot produkál. A példa kedvéért a 2.17. ábra egy 3 állapotú automatát mutat be.



2.17. ábra. Egy 3 állapotú véges automata. A kezdőállapot az *A*. A bementi szimbólumok a vonal baloldalán vannak, a kimeneti szimbólumok pedig a vonal jobboldalán.

A bemeneti szimbólumok halmaza {0, 1}, míg a kimeneti szimbólumok halmaza {α, β, γ}. A véges automata egy bementi szimbólumsorozatot átalakít egy kimeneti szimbólumsorozattá. A 2.2. táblázat az automata válaszait tartalmazza bizonyos szimbólumokra, feltéve, hogy az automata a *C* állapotban van.

2.2. táblázat. A 2.17. ábrán levő véges állapotú automata válaszai egy megadott szimbólumsorozatra. Az automata a *C* állapotban van.

Aktuális állapot	C	B	C	A	A	B
Bemeneti szimbólum	0	1	1	1	0	1
Következő állapot	B	C	A	A	B	C
Kimeneti szimbólum	β	α	γ	β	β	α

Atmar 1976-ban kiszámolta egy *n* állapotú, *a* darab bemeneti – és *b* darab kimeneti szimbólummal rendelkező véges automata összes lehetséges konfigurációinak a számát:

$N = (n^a b^a)^n$. [Fogel66] olyan kísérleteket végeztek, ahol a környezet bitsorozatokból állt és a maximális állapotszám 25 volt. Ezért az állapottér mérhetetlenül nagy volt: $(25^2 2^2)^{25} = (5^{100})(2^{50}) \approx 2^{233} \cdot 2^{50} \approx 10^{85}$.

Az EP ilyen véges állapotú automatákkal dolgozik. Egy véges automatákból álló populáció ki van téve a környezetnek (ami nem más mint az eddig megfigyel szimbólumok halmaza). A szülő gépek esetében minden kimeneti szimbólum össze van hasonlítva a következő bemeneti szimbólummal. Az előrejelzés értékét egy értékelő (rátermettségi) függvény adja meg (pl. minden-semmi, abszolút hiba, négyzetes hiba). Miután megvolt az utolsó előrejelzés is, ez a függvény visszaadja a gép rátermettségét.

Leszármazott gépeket a szülők véletlenszerű mutálásával lehet előállítani. Az egyszerűség kedvéért minden szülő csak egy utódot hoz létre. A mutációkra 5 különböző lehetőség van (variációs operatoroknak is nevezik):

1. Kimeneti szimbólum megváltoztatása;
2. Állapotok közötti átmenetek módosítása;
3. Új állapot hozzáadása;
4. Állapot törlése;
5. Kezdőállapot megváltoztatása.

Az állapot törlése és a kezdőállapot módosítása csak akkor megengedett, ha a szülő gépnek legalább két állapota van. A mutációt általában úgy választják meg, hogy egyenletes eloszlású legyen. A mutációk száma lehet előre rögzített vagy megfelelhet egy valószínűségi eloszlásnak (pl. Poisson). A leszármazottakat ugyanúgy értékeljük ki a létező környezet felett mint a szülőket.

A legrátermettebb gépek kiválasztásra kerülnek, ezek lesznek a következő generáció szülő automatái. Ezt a folyamatot addig ismételjük amíg a környezet által igényelt szimbólum előrejelzésre kerül.

Az állapotok számának a növelése a keresésre szánt lépésszám csökkenéséhez vezet, és ugyanakkor a struktúra fejlesztésének finomabb hangolását biztosítja. Solomonoff, aki erősen kritizálta ezt a programozási módszert, azt állította, hogy az EP nagyrészt egy egyszerű hegymászó algoritmus, ami egyetlen szülő gépből egyetlen leszármazott gépet fejleszt ki [Fogel00].

2.11.2. Algoritmus

Az algoritmus a GA elvét követve nem sokban különbözik a 2.6. fejezetben megismert algoritmussal. Mivel ez a módszer időben egy korábbi kísérletezése volt az evolúciós számításoknak, így itt nincs jelen a keresztezés, és a hatékonyság növelésének érdekében véges automatát (az algoritmusban FSM) használ [Fogel93a]. Az EP egy lehetséges algoritmusát a 2.18. ábra szemlélteti.

```
t := 0
P(t) kezdeti populáció inicializálása véletlen egyedekkel
E inicializálása
P(t) egyedeinek kiértékelése a rátermettség alapján
while not (megállási feltétel) do
  if (előrejelzés szükséges) then
    Legjobb FSM használata a szimbólum előrejelzésére
    Aktuális szimbólum hozzáadása E-hez
  endif
  t := t + 1
  P(t) kiválasztása P(t-1)-ből a rátermettség függvényében
  P(t) mutáció általi sztochasztikus módosítása
  P(t) kiértékelése
Endwhile
```

2.18. ábra. Az EP algoritmusa.

Az E halmaz a környezetet jelenti. A mutációra a fennebb felsorolt variációs operátorok kombinációit használhatjuk. A `(megállási feltétel)`-t megadhatjuk t függvényében, azaz az algoritmus megáll, ha a t túllépet egy bizonyos határt.

2.11.3. Az evolúciós programozás alkalmazása

Az evolúciós programozás módszerét széles körben alkalmazták, elsősorban numerikus optimalizálásra és nehéz, NP-teljes feladatok megoldására (lásd a prímszámos feladatot a 2.11.3.1. részben).

Fogel és Burgin az evolúció programozást két személyes, zérusösszegű játékokra alkalmazták. Egyszerű játékok (kevés lépésszámú) esetében az EP képes volt a legjobb stratégia felfedezésére és akár az emberi ellenfeleket is le tudta győzni. Később a módszert kiterjesztették nem zérusösszegű, koevolúciós játékokra is. Az EP egy üldözőt kellett irányítson egy mozgó céltárgy felé (ez volt a menekülő) a 3-dimenziós térben. Mivel a menekülő mozgékonyabb volt, így az üldöző sikere attól függőt, hogy milyen mértekben tudja a céltárgy pozícióját előre megjósolni anélkül, hogy előzetes ismerete lett volna e tárgy dinamikájáról.

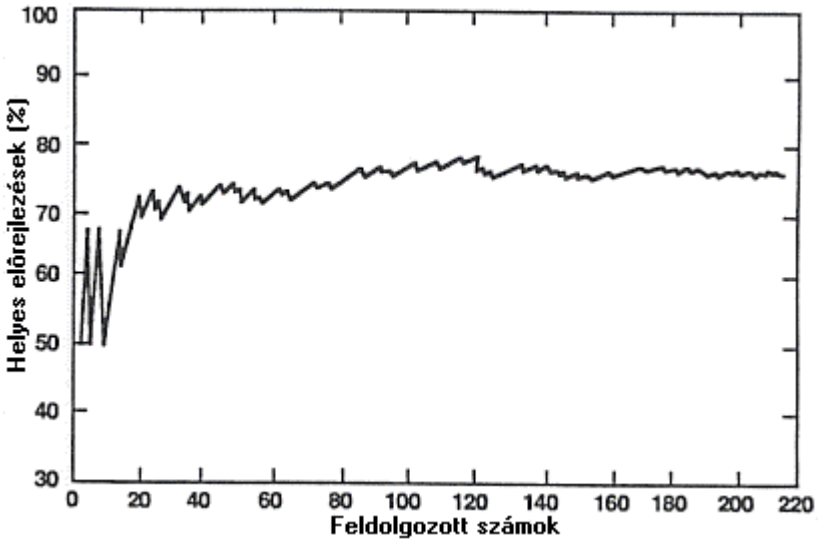
Bizonyos kutatásokban az evolúciós modell képes volt gyorsan újratanulni a környezetet amit korábban megtapasztalt, még akkor is, ha időközben a környezet radikális változásokon ment keresztül. A véges állapotú automata beépítette a memóriába a korábbi szimbólumsorozatokat.

Jelenleg az EP technikákat különböző kombinatorikus, optimalizálási problémákra alkalmazzák. A véges automaták helyett problémafüggő reprezentációkat használnak, az operátorok pedig úgy vannak megválasztva, hogy erős viselkedési kapcsolatot tartson fenn a szülő és leszármazott között. Sikeresen alkalmazták ezeket a módszereket útvonal tervezésre, neuronhálók tanítására és tervezésére, automatikus vezérlésre, játékokra és általános függvény optimalizálásra.

2.11.3.1. Alkalmazás: prímszámok előrejelzése

A prímszámok osztályozására egy változó szimbólumsorozat volt generálva, ahol az 1-es azt jelentette, hogy az adott pozitív egész szám prím, illetve a 0 azt, hogy nem prím. Így, a környezet a 01101010001… sorozatból áll, ahol mindegyik szimbólum az 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,… természetes számok prím tulajdonságát írja le.

Az előrejelzést értékelő függvény a minden vagy semmi (*all-or-none*) volt, ami azt jelenti, hogy 1 pont járt minden helyes előrejelzésre, 0 pont minden hibára és ebből még le volt vonva 0,01 minden egyes állapotszámra. A komplexitást érintő büntetésre azért volt szükség, hogy az automata takarékos legyen a memóriával. A kísérletek során megvizsgálták az első 200 pozitív egész szám prímségét. Az előrejelzések helyességének kumulált százaléka elérte a 78%-ot a 115-ik szimbólumnál és nagyrészt konstans maradt egész 200-ig (lásd a 2.19. ábrát). Ekkor a legjobb gépnek 4 állapota volt, a 201-ik számnál 3 és a 202-iknél pedig csak egy állapota volt a legjobb automatának. A 719-ik szimbólum után az elért helyességi kumulált százalék 81,9 volt, de az aszimptotikus érték 100%-os, hiszen a prímszámok mind ritkábban fordultak elő és az automata továbbra is nem-prímet jelzett előre.



2.19. ábra. Az egész számok prímségének az előrejelzése. 0,01-es komplexitási költség/állapot, egy generációban 5 gép kerül kiválasztásra, 10 generáció/előrejelzés.

A rátermettségi függvényt úgy módosították, hogy a ritka esemény előrejelzését jobban értékelje, azaz az automata egy prímszám helyes előrejelzésére megkapta az egy pontot plusz az őt megelőző nem prímek számának összegét, valamint egy nem prím előrejelzésére 1 pont plusz a számot megelőző prímek összege járt. Az első 150 darab szám előrejelzése alatt 30 helyes prím előrejelzés volt, 37 helytelen jóslás és 5 darab elvesztett prímszám. A 151-ik és az 547-ik szimbólumok között 65 prím jóslás volt és 67 téves riasztás. Ez azt jelenti, hogy míg az első 35 prímszám közül 5 elmulasztás volt, addig a következő 65 prím előrejelzésekor nem volt mulasztás. Az evolúciós algoritmus gyorsan megtanulta felismerni a kettővel és hárommal (az öttel kevésbé) osztható számokat, anélkül, hogy előzetes ismerete lett volna a prímszámok fogalmáról vagy képes lett volna osztani. Mi több, nem létezik olyan véges automata ami képes lenne prímszámokat generálni.

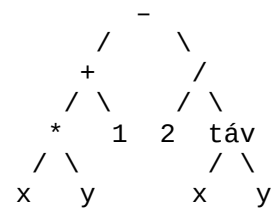
2.12. Genetikus programozás

A genetikus programozás – röviden GP – a genetikus algoritmus legújabb alkalmazási területe, aminek Koza az úttörője és a [Koza92] monográfiájában írta le részletesen ezt a programozási módszert. Sokkal komplexebb mint a GA, hiszen képes általános, hierarchikus gépi programokat (általában LISP nyelven íródott) fejleszteni, tanítani, amelyek különböző méretűek és alakúak lehetnek. Mivel a genetikus algoritmusok elvét követi, a GP is a természetes szelekción alapszik és a megismert genetikus operátorokat (szelekció, mutáció, keresztezés) használja, persze nagyobb figyelmet fordítva ezeknek az operátoroknak az alkalmazásakor. Amíg a genetikus algoritmusok rögzített hosszúságú, függvények

paramétereihez rendelt bit stringeken dolgoztak, addig a GP programok vagy függvények struktúráját kezelik. Innen is látszik, hogy mennyire robusztus, erőforrás-igényes ez a módszer. Ezért is tudott ez a terület az utóbbi években népszerűvé válni, hiszen a számítógépek rohamos fejlődése lehetővé tette a hatékonyabb kutatást (Koza-nak egy 1000 csomópontos Pentium II architektúrájú rendszer és egy 70 csomópontos Alpha processzoros rendszer áll a rendelkezésére).

2.12.1. Elemző fák

Egy gépi program generálásakor fontos követelmény, hogy szintaktikailag helyes legyen. Bizonyos nyelvekben (például C/C++, Pascal, Java stb.) íródott programokat lehetetlen az eredeti formájukban kezelni. Az **elemző fák** (*parse trees*) segítségével egy elegáns, egyszerű és homogén formában lehet ábrázolni ezeket a programokat, amely lehetővé teszi a könnyebb kezelhetőséget is. Így az elemző fa a GP „anyanyelve”, biztosítva ily módon a genetikus operátorok egyszerű és összefüggő használatát. Egy másik előnye, hogy a LISP programok könnyen megírhatók elemző fák formájában és fordítva. A jobb megértésért lássuk a következő példát: tegyük fel, hogy a (x * y + 1) – (2 / (táv x y)) kifejezést akarjuk kiértékelni. Ehhez a kifejezéshez tartozó elemző fa a következőképpen néz ki:



Minden paraméteres függvény (a példában a *táv* – a paraméterként megadott *x* és *y* közötti távolságot számolja ki – illetve az aritmetikai operátorok) a fa egy nem terminális csúcsában vannak, míg a változók, konstansok és paraméter nélküli függvények a fa leveleiben helyezkednek el. Ezzel a megoldással tömören és egyszerűen leírtuk a programunkat. Mivel a LISP program valójában elemző fa csak azzal a különbséggel, hogy prefix jelölést használ, a fenti példa a következőképpen irható: (– (+ (* x y) 1) (/ 2 (táv x y))).

Továbbá vigyázni kell arra, hogy a használt függvények mindegyike le tudjon kezelni minden lehetséges értéket (pl. zéróval való osztás esetén megfelelő értéket adjon vissza és ne álljon le a program). Azt is fontos megvizsgálni, hogy a felhasznált függvényekkel és terminális elemekkel meg lehet-e oldani az adott feladatot (pl. a {+, –} operátorok illetve az

{1, 2, …} természetes számok segítségével lehetetlen a *log*(*x*) függvényt implementálni) [Generation03c].

2.12.2. Algoritmus

A genetikus programozásnál felhasznált algoritmus ugyanaz mint a genetikus algoritmus esetében (lásd a 2.6. fejezetet), azzal a különbséggel, hogy itt az egyedek, azaz a kromoszómák nem rögzített hosszúságú bit stringek, hanem elemző fákkal ábrázolt, LISP szintaxisú programok. Ennek értelmében az algoritmus egy lehetséges változata a 2.20. ábrán látható.

```
Hozzuk létre véletlenszerűen egy kezdeti populációt
felhasználva a kiválasztott szimbolikus kifejezéseket
t := 1
while not (megállási feltétel) do
  Egyedek kiválasztása az aktuális populációból a szelekciós
  operátor segítségével a rátermettség figyelembe vételével
  Keresztezés és mutáció alkalmazása a kiválasztott egyedeken
  Kiválasztott egyedek lecserélése a leszármazottakkal
  t := t + 1
endwhile
return (legrátermettebb egyed az utolsó populációból)
```

2.20. ábra. A GP algoritmusa.

A `(megállási feltétel)` megvizsgálhatja, hogy a `t` változó elért-e egy bizonyos értéket, amikor is az algoritmus megállhat. Ekkor az algoritmus megkapta, vagy elég jól megközelítette az optimális megoldást. Kezdeti populációt úgy hozunk létre, hogy különböző mélységű és alakú elemző fákat generálunk. Csak arra kell ügyelni, hogy véges mélységű legyen a fa és a levelek terminális elemeket tartalmazzanak. Terminális elem a fa közbeeső csúcsaiban is lehet amikor is a fa alakja nem lesz szimmetrikus.

A rátermettség kiszámolásának érdekében megnézzük, hogy az adott program milyen jól közelíti meg a megoldást. Ezt úgy oldják meg a gyakorlatban, hogy lefutatják a programot különböző paraméterekre, kiértékelve a kimeneti adatokat. Az egyed (program) rátermettsége egyenlő a kapott érték és az elvárt érték közötti különbségekkel, összegezve az összes futási teszt ily módon kiszámolt eredményét. Ekkor a legkisebb fitnesz-értékű egyed a legrátermettebb, a zéró érték pedig a legjobb rátermettségnek felel meg.

Keresztezéskor két egyed tetszőleges részfája vagy terminális eleme felcserélődik. A két fán addig megyünk végig, amíg két csomópontot véletlenszerűen kiválasztunk, amelyeket felcseréljük. Mutációkor tetszőleges függvényszimbólumot, terminális elemet vagy részfát

lecserélünk egy másik függvényszimbólumra, terminális elemre vagy egy új részfára (amelyet ugyanúgy generáltunk mint a kezdeti populációt).

Ha belegondolunk, hogy egy átlagos program hány függvényt, változót, konstanst, operátort és feltételt használ, akkor a keresési tér roppant nagy terjedelmű, hiszen ezeknek az elemeknek a permutációjából áll. Ha a használt genetikus algoritmus 500 egyedet tartalmaz, az iterációk száma 50 és figyelembe vesszük a rátermettségi függvény meghatározására szánt időt (amit a végrehajtott teszt programok igényelnek) akkor egy genetikus program lefutatásához szükséges idő hihetetlenül nagy lehet. A genetikus programozás nagyon számításigényes, még akkor is, ha egy jól megválasztott művelethalmaz és vezérlési algoritmus áll a rendelkezésünkre [Generation03c].

2.12.3. A genetikus programozás alkalmazása

A genetikus programozás korlátja a nagy keresési térben nyilvánul meg. Ezért fontos, hogy minimális számú operátort, függvényt és terminális elemet használjunk a fejlődésnek alávetett programban. A gyakorlatban jól alkalmazható a tőzsdei előrejelzésekben, a felsőbb matematikában és a katonai szimulációkban [Generation03d].

Koza sikeresen alkalmazta diszkrét játékok (*Discrete Game*) [Koza92] megoldására, hiszen a stratégia megtalálását jelentheti egy gépi program megtalálása (lásd a 2.10. fejezetet).

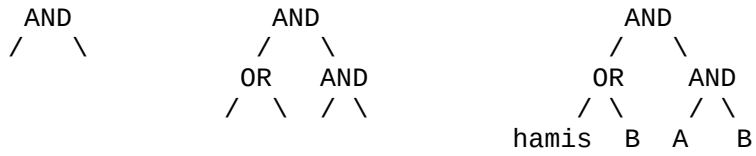
2.12.3.1. Alkalmazás: XOR függvény kivitelezése, tanítása

Implementáljuk az XOR (kizáró vagy) operátort GP segítségével, csak az AND (és), OR (vagy) és NOT (tagadás) logikai operátorokat használva [Generation03c]. A kizáró vagy csak annyiban különbözik a vagytól, hogy ha mindkét operandus (legyen A és B) igaz, akkor az XOR hamisat eredményez.

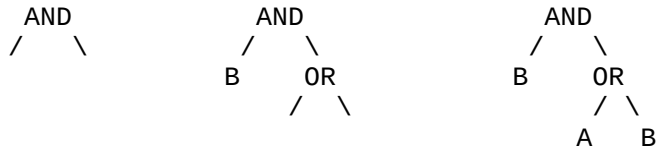
A feladat megkötéséből következik, hogy a függvények halmaza {AND, OR, NOT} operátorokat tartalmazza, míg a terminális elemek halmaza {A, B, igaz, hamis}.

2.12.3.1.1. Kezdeti populáció generálása

Az egyszerűség kedvéért 3 mélységű értelmező fákat generálunk. Minden szinten véletlenszerűen generálunk egy függvényszimbólumot, mikor elértük a kívánt mélységet akkor kötelező módon terminális szimbólumot generálunk. Íme a lépésenként generált fa:



A következő példában a bal részfa egy terminális szimbólumból áll, ekkor a fa aszimmetrikus:



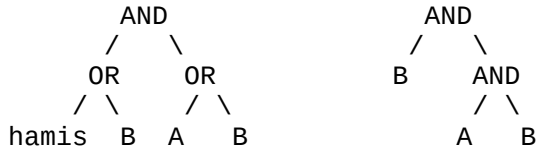
2.12.3.1.2. Rátermettség kiszámolása

Minden A és B operandus-párra megnézzük, hogy az AND mennyivel tér el az XOR operátortól. Mivel a kettő eredménye csak akkor egyezik meg amikor mind a két operandus hamis, a többi 3 esetre eltérő eredményeket kapunk, így következik, hogy a rátermettségi érték 3.

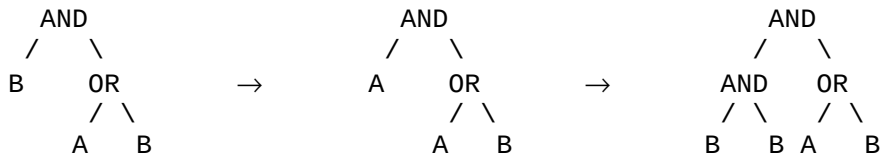
2.12.3.1.3. Genetikus operátorok

Keresztezés: legyen a következő két egyed (ami a mi estünkben egy elemző fa által reprezentált program):

Tegyük fel, hogy a keresztezés következtében a két egyed bal részfáját felcseréljük. A leszármazottak:

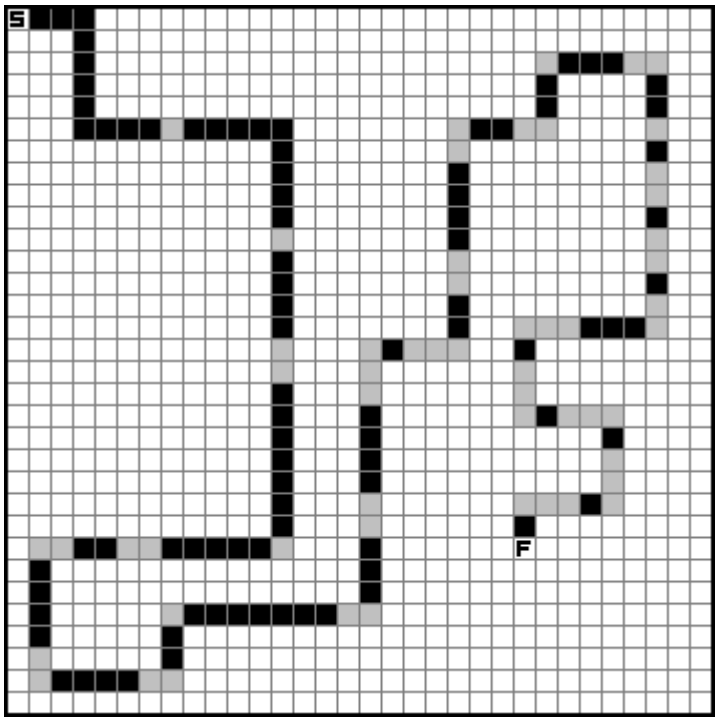


Mutáció: a bal részfa terminális elemét lecseréljük egy másik terminálisra, majd ezt egy újonnan generált részfára:



2.12.3.2. Alkalmazás: mesterséges hangyaprobléma

A Santa Fe nyom (*trail*) vagy a mesterséges hangyaprobléma (*Artificial Ant Problem*) számítógépes szimulációja egy útvonalnak, amely egy valódi hangya lehetséges problémáit eleveníti fel egy nyomnak a követésében [Koza92]. A nyomnak a nevét az amerikai Santa Fe Intézet (*Santa Fe Institute*) viseli, hiszen ennek az intézetnek a kutatói fogalmazták meg először ezt a feladatot. A nyomot egy 32x32-ös négyzethálóra helyezzük a 2.21. ábrán látható módon.



2.21. ábra. A Santa Fe nyomra alkalmazott mesterséges hangyaprobléma.

A hangya (a mi esetünkben inkább egy mesterséges ágens, amely a hangyát szimulálja) az *S*-el jelölt kezdőcelláról indul el és a cél az, hogy érjen el egy előre megadott lépésszám alatt az *F* végállapotig. A fekete cellák az élelmet vagy a ferómont (*pheromone*) – biológiában használt szakkifejezéssel élve – képviselik, míg a szürke cellák helyén hiányzik a ferómon. Az ágensnek pont ez jelenti a problémát: képes kell legyen megtalálni az utat, még akkor is, ha előtte nincs ferómon. A nyomnak a komplexitása nő ahogy haladunk rajta előre a célállapot felé. Az elején csak egy celláról hiányzik a ferómon, majd kettőről, később ezek a hiányosságok a sarkokon fordulnak elő mind gyakrabban és gyakrabban, majd az út vége felé három egymást követő celláról, amelyik ráadásul sarkon helyezkedik el. Mindez azért van, hogy az ágensnek lehetősége legyen a nyomkövetés folyamán fejlődni, megtanulni a szükséges mozgásokat (pl. előre lépés, jobbra-balra fordulás, lóugrásszerű lépés).

Annak ellenére, hogy egy komplex problémát jelent az ágens számára, a rátermettségi függvény roppant egyszerű: az ágens által érintett ferómon szagú cellák, blokkok száma. Annak érdekében, hogy elősegítsük a hangya lépéseit, a ferómon szétoszlik azokról a cellákról amelyeken már járt, így többé-kevésbé megakadályozzuk, hogy a hangya előre-hátra járjon ugyanazon a blokkon. A kísérlet során az algoritmusban a hangyának 600 lépés alatt kellett eljutnia a végállapothoz, annak ellenére, hogy az elérhető maximális fitnesz-érték csak 89 (ami egyenlő a fekete cellák számával), míg az út hossza 144.

Következésképpen egy ágensnek, hogy bejárhassa a Santa Fe nyomot, meg kell tanulnia két képességet: lépjen előre, ha érzékeli a nyomot és keresse meg az utat, ha ferómonok hiányában nem érzékeli azt.

Ahhoz, hogy a feladatot megoldhassuk a genetikus programozás módszerével definiálni kell a terminális elemeket és a függvényszimbólumokat. Mivel a hangyának előre (ezzel léphet hátra is, ha kétszer egymás után ugyanabba az irányba fordult), jobbra és balra kell lépnie, így a terminális elemek halmaza T = {ElőreLép, JobbraFordul, BalraFordul}. Ezek beállítják az aktuális irányt és visszaadják a megfelelő rátermettségi értéket (a JobbraFordul és BalraFordul 0 értéket ad vissza). A függvényszimbólumok paraméterként megkapnak egy gént (ami egy cellát reprezentál) és kiszámolják a rátermettséget: F = {VanElőlÉlelem, KövCella, KövKövCella}. A KövCella megadja az aktuális és rákövetkező cella fitnesz-értékét, míg a KövKövCella abban különbözik az előző függvénytől, hogy hozzáveszi az aktuálistól számolt második cella fitnesz-értékét is. Ezek a függvények a genetikus program változatosságát és képességét hivatottak biztosítani.

A fent megadott programelemek segítségével, betartva a feladat szabályait legenerálunk egy szimbolikus programot, ezt a GA segítségével kifejlesztjük, ami megoldja a kért problémát [Fraser94].

3. Játékok és játékelmélet

Az IBM kutatói által készített *Deep Blue* [Campbell02] nevet viselő sakkprogram határkő a mesterséges intelligencia (nevezetesen a játékelmélet) történelmében, ugyanis 1997 májusában egy 6 meccses mérkőzésben a gép megverte a többszörösen világbajnok Kasparov-ot. A szakmédia hajlamos volt azt mondani, hogy a gépek okosabbak lettek mint az emberek. Ne felejtsük el, hogy a programot több ember készítette éveken keresztül, no meg a segítségre hívott roppant erős számítógépeket. A 32 darab IBM szuperszámítógép 512 különböző sakkjátékot volt képes játszani és elképesztően nagy teljesítményt ért el: másodpercenként 200 millió sakkpozíciót ellenőrzött le [Clark97].

A mesterséges intelligencia olyan szellemi feladatok számítógépes megoldását tűzte ki célul, amelyek az ember részéről magasfokú intelligenciát, kognitív tevékenységet igényelnek. A kétszemélyes játékok témaköre az elsők között keltette fel a kutatók érdeklődését, hiszen a kihívás is nagy volt: olyan programokat készíteni, amelyek túlszárnyalják az ember teljesítményét, képességet.

A játékelmélet (*game theory*) a modern számítógépek megjelenésével kezdett fejlődni, amikor lehetőség nyílt bonyolultabb matematikai számítások végrehajtására és a valós életből vett szimulációk megvizsgálására. A matematika, közgazdaságtan és informatika szimbiózisa. A gazdasági élet felgyorsulása és a gazdasági szereplők mind nagyobb számú jelenléte a piacon komplex interakciókat generált. Ezeknek a folyamatoknak a megvizsgálása szükségessé vált, hiszen ily módon többé-kevésbé ki lehetett számítani a másik fél szándékát. Az első játékelméleti próbálkozás Neumann nevéhez fűződik, amikor is a játékokon keresztül a gazdasági magatartásformákat vizsgálta meg [Neumann44]. A matematikai játékok a valós-világbeli modellek segítségével intelligens viselkedési formák tanulmányozását teszik lehetővé. Sokan úgy értelmezik a játékelméletet mint egy magyarázó elmélete a stratégiai érveléseknek, következtetéseknek [Ross02].

A számítógépes játékok a mesterséges intelligencia egyik legrégebbi és legérdekesebb kutatási területe. Az 1950-ben [Shannon50] által ajánlott sakkprogramtól a *Deep Blue* megírásáig majdnem fél évszázad telt el, ami idő alatt a szakirodalom hatalmasra nőtte magát, újabbnál-újabb módszereket fejlesztve ki, amelyeket sikerrel alkalmaztak a mesterséges intelligencia más területein is. A *Deep Blue* sikere alátámasztja a több évtizedes

munka fontosságát, ennek eredményességét és tovább inspirálja a gépi tanulás terén folytatott kutatásokat.

3.1. Gépi tanulás

A gépi tanulás (*machine learning*) az emberi tanulást próbálja meg gépek szintjére átvinni. Olyan tanulási módszerek után kutat, amelyeket meg lehet értetni a gépekkel (számítógépek, robotok stb.), rá lehet venni a gépeket, hogy megtanuljanak bizonyos műveleteket, stratégiákat és döntéseket hozzanak. Olyan számítástechnikai módszerek összessége, amelyek révén lehetővé válik új ismeretek beépítése az MI programokba – új készségek kialakítása, illetve a meglevő ismeretek átrendezése, átértékelése céljából. E módszerek erősen építenek az induktív módszerekre, úgymint a példákon keresztüli tanulásra, illetve a megfigyelésre és felfedezésre támaszkodó osztályozásra.

A gépi tanulásnak fontos kutatási területét képezik a játékok és ezen belül a stratégiai játékok megoldása. Egy játék megoldása alatt azt értjük, hogy találtunk egy olyan módszert, technikát amelyet egy stratégiával kombinálva a játék verhetetlen lesz, vagy mesteri, világbajnoki szinten játszik, azaz olyan emberi ellenfelekkel képes jól játszani, akik az adott pillanatban az adott játékban világbajnokok.

Első látásra azt hinnénk, hogy a játszás és játékokkal való foglalkozás nem tudományos munka és csak szórakoztató jellege van. Ez nem így van. Láttuk, hogy a játékelmélet komoly matematikai alapokat biztosit és a számítógépes szimulációk során elkészített tanulási módszerek, stratégiák jól alkalmazhatók a robotikában. Észrevehetjük, hogy ily módón milyen szorosan összefüggnek a mesterséges intelligencia területei.

3.2. Játékok ismertetése

Ebben a szakaszban csak két játékot ismertetünk részletesebben, ugyanis ezekre alkalmazzuk a 4. fejezetben az evolúciós módszereket. A többi szóban forgó játékok szabályait ismertnek tekintjük. A egyszeri fogolydilemma és iterált fogolydilemma ismertetése a 4.2.1. fejezetben található, a kirakós játéké pedig a 5.3. fejezetben.

3.2.1. X-O

Az X-O (szokták még amőbának is nevezni) – angolul ***T**ic-**T**ac-**T**oe* és innen a *TTT* rövidítés – kétszemélyes, teljes informáltságú, zérusösszegű táblajáték. A játékosok felváltva

helyezik el egy 3x3-as négyzethálón a saját jeleiket (*X* illetve *O*, általában az *X* kezd) és az első aki 3 darab ugyanolyan típusú egymásmelletti jelet gyűjt ki vízszintesen, függőlegesen vagy átlósan az lesz a nyertes. 8 darab ilyen nyertes sor van.

Ha nincs nyerés egyik játékos részéről sem és nincs már üres négyzet (cella) akkor a játék döntetlen.

3.2.2. Amőba

Az amőba, vagy a szakirodalomban *Go-Moku* (mivel japán eredetű, ezért a neve is japán hangzású) a TTT-nek úgymond a kiterjesztett, általánosított változata. Ez nem azt jelenti, hogy a TTT-ből fejlődött ki, hanem csak azt akarjuk sugallni, hogy a játékszabályok kis különbségekkel ugyanazok. Itt is egy négyzetrácson két játékos felváltva helyezi el a jeleit. A cél itt is ugyanaz mint a TTT-nél, csak itt nem 3 hanem 5 darab egymást követő azonos jelet kell kigyűjteni.

Több változata létezik. Az olimpiai játékok listáján az úgynevezett standard Go-Moku szerepel, ami 15x15-ös négyzetrácsot használ. Ha a tábla megtelt anélkül, hogy valamelyik játékos nyert volna, a játékot döntetlennek nyilvánítjuk.

3.3. Játékok tulajdonságai, osztályozása és bonyolultsága

A játékokat több kritérium szerint lehet osztályozni. Tulajdonképpen ezek a kritériumok a játékok tulajdonságai, jellemzői, amiket az elkövetkezőkben ismertetünk. A tulajdonságok közötti eltérések játékosztályokat határoznak meg.

3.3.1. Játékosok száma

Egy játékban egy, kettő vagy több játékos vetélkedhet egymással. A legtöbb logikai, táblás játék kétszemélyes, a 8-as játék viszont egyszemélyes, de a játékelméletben léteznek többszemélyes játékok is: például a fogolydilemmát kiterjesztették több játékosra is.

3.3.2. Informáltság

Teljes információjú (vagy informáltságú) és nem teljes információjú játékok léteznek. Egy teljes információjú játékban minden játékosnak, a játék bármely pillanatában hozzáférése van minden információhoz, ami a játék állapotát és lehetséges folytatását illeti. Például a sakk teljes információjú: mindkét játékos ismeri a tábla konfigurációját (a játék jelenlegi állását), az eddig megtett összes lepést és a sakk szabályait. Minden olyan játék ami nem tartozik be ebbe a csoportba azok nem teljes információjúak. Például ilyenek elsősorban a kártyajátékok

(pl. Bridge), ahol a játékosok csak a saját kártyáikat ismerik, a többit nem, és sokszor a szerencse (véletlen) dönti el a játék következő állását. Egy másik nem teljes információjú játék a fogolydilemma.

3.3.3. Zérusösszegű

A játék zérusösszegű ha egy játékos nyereségeinek és veszteségeinek összege nulla. A táblás játékok zöme ilyen, de például az iterált fogolydilemma nem zérusösszegű (lásd a 4.2.2. fejezetet).

3.3.4. Végesség

Egy játék véges ha egy adott állapotból véges darab lépéslehetőség van és a játék véges idő alatt befejeződik. A játékok nagy része ilyen típusú. Az iterált fogolydilemma viszont végtelen lépésű, azaz egyik játékos sem tudja, hogy mikor ér véget a játék.

3.3.5. Hirtelen halál

E jellemző szerint létezik olyan játék ami hirtelen halálú (*sudden death*) és olyan ami rögzített végződésű (*fixed termination*). Egy hirtelen halálú játék befejeződhet váratlanul, ha létrejön egy előre megadott mintahalmaz. Ilyen játék a Go-Moku, ahol, a játék befejeződik, ha valamelyik játékos elérte azt, hogy 5 jele legyen egymás mellett folytonosan. A hirtelen halál akkor is beáll, ha megtelt az összes négyzet és a játékosok nem tudnak lépni. Ekkor a játékot döntetlennek nyilvánítjuk. Az Othello rögzített végződésű, a játék megáll, ha a játékosok nem tudnak lépni vagy valamelyik játékosnak nincs korongja a táblán. Ez egy nagyon fontos tulajdonság lehet a játékfa méretének a korlátozásánál [Allis94].

A fentiek értelmében a kétszemélyes teljes információjú játékok a következő tulajdonságokkal rendelkeznek:

- Két játékos felváltva lép, a megadott szabályok szerint;
- Mindkét játékos birtokában van a játékkal kapcsolatos összes információnak, azaz ismerik a korábban megtett lépéseket és minden részletét látják a kialakult állásnak. A szerencsének nincs semmilyen szerepe az adott játszma alakulásában;
- Minden állapotban véges számú szabályos lépés közül lehet választani;
- Véges játszmák vannak;
- A játszmák végén van egy nyertes és vesztes játékos, illetve bizonyos játékoknál döntetlen is előfordulhat.

3.3.6. Bonyolultság

A játékokkal kapcsolatos bonyolultságot két különböző mérték kifejezésére használják: állapottér komplexitás és játékfa komplexitás.

3.3.6.1. Állapottér bonyolultság

Az állapottér bonyolultság úgy definiálható mint a játék kezdőállapotából elérhető legális játékpozíciók száma [Allis94]. A Tic-Tac-Toe esetében egy durván felülről becsült állapottér bonyolultság 3^9=19683, mivel 9 darab négyzet egyenként 3 különböző jelet tartalmazhat: az *X*, a *O* vagy az üres négyzet. Egy szűkebb felső korlát lenne a 6046, ahol figyelembe vettük azt, hogy a *X*-ek száma egyenlő kell legyen vagy eggyel meghaladhatja az *O*-k számát. Továbbá, egy állapot illegális, ha egy újabb lépés volt végrehajtva miután a nyerés beállt az egyik oldalon. A így kapott 5478 legális állapot meghatározza a TTT állapottér komplexitását.

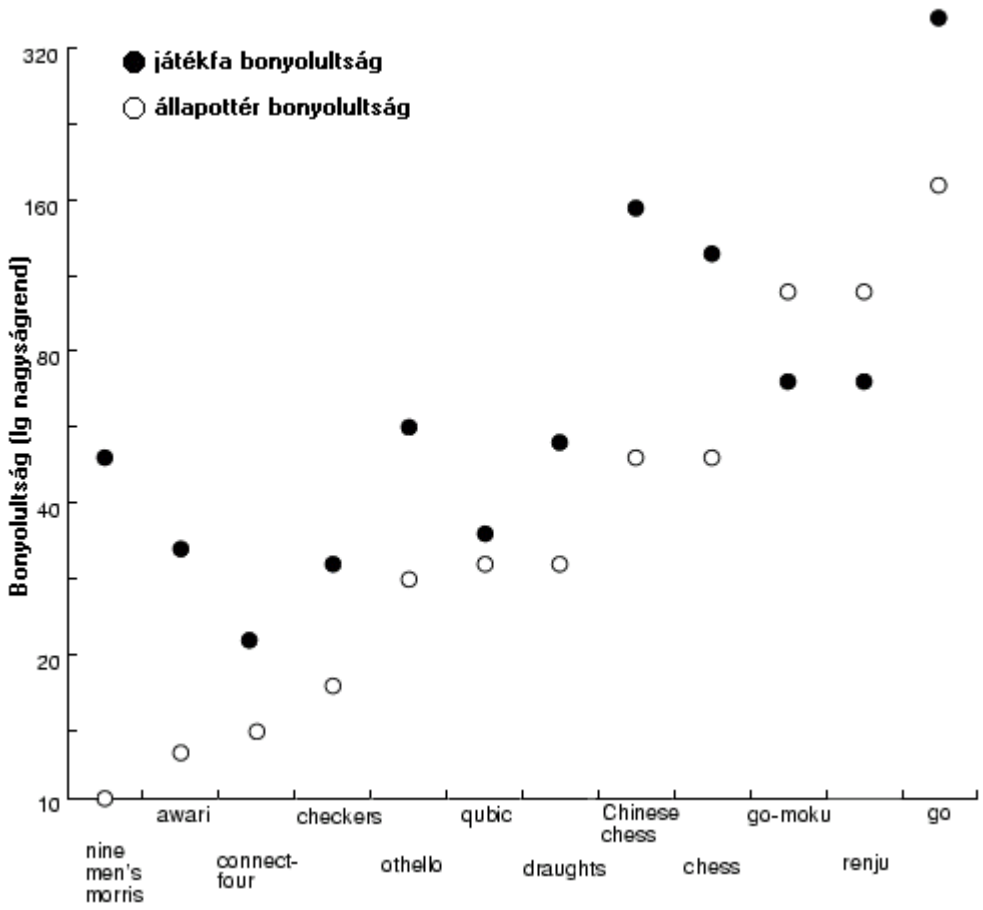
3.3.6.2. Játékfa bonyolultság

A játékfa bonyolultsága egy játéknak a terminális csúcsok száma a kezdőállapothoz tartozó keresőfában (játékfában) [Allis94]. A TTT esetében az átlag lépésszám 9, mivel sok játék döntetlen. Az *i*-ik szinten az elágazási tényezője (*branching factor*) a játékfának 9-*i*. Az elágazási tényező az adott szinten a legális lépések számát jelenti. Ezért a 9 mélységű fa 9!=362880 darab terminális csúcsot tartalmaz, ami nem más mint a TTT játékfa komplexitása. Ez nagyobb lehet mint az állapottér bonyolultsága, mivel ugyanaz a pozíció (állapot) több helyen is előfordulhat a fában.

A 3.1. ábra grafikusan szemlélteti a két típusú bonyolultságot különböző játékok esetében.

A malom (*Mill* vagy *Nine Men's Morris*) állapottere a legkisebb az olimpiai játékok közül: 10^{10}. A játékfa bonyolultsága jóval nagyobb. Az átlag elágazási tényező 10, míg a két játékos által megtett átlag lépésszám 50. Következik, hogy a játékfa leveleinek száma 10^{50}.

A sakk (*Chess*) játékfájának a becslésekor tegyük fel, hogy az átlagos jászmában 40 lépésváltás történik és az egyes állásokban a legális lépesek átlagos száma 35. Ez azt jelenti, hogy a fa mélysége 80, az elágazási tényezője pedig 35. Innen következik, hogy a fának $35^{80} \approx 10^{123}$ kiértékelendő levele van. Egy finomabb becslésnél vegyük figyelembe azt, hogy statisztikai adatok szerint az állások átlagában csak 1,76 „jó" lépés van [Fekete99]. Még ekkor is $1{,}76^{80} \approx 10^{20}$ terminális csúcsot tartalmaz a fa. A sakk állapotainak a száma 10^{50}.



3.1. ábra. Néhány kétszemélyes teljes információjú, zérusösszegű, olimpiai játék állapottér bonyolultsága és játékfa bonyolultsága.

A Go-Moku az olimpiai játékok közül a legegyszerűbb szabályokkal rendelkezik, mégis az egyik legbonyolultabb amit az állapottér és a játékfa illet. A játék állapottere $3^{225} \approx 10^{105}$ pozíciót tartalmaz, míg a játékfa bonyolultsága $210^{30} \approx 10^{70}$, ahol tudjuk, hogy egy játék átlagosan 30 lépést tartalmaz és az *i* szinten a legális lépések száma 225-*i*, ami azt jelenti, hogy az átlag elágazási tényező 210.

A Go a legkomplexebb játék. Jelenleg ezen a játékon kutatnak a legtöbbet. Azon kevés játékok közé tartozik, amely még nincs megoldva, vagyis nem létezik olyan stratégia ami képes lenne nagymesteri szinten játszani. Mivel a két játékos a saját korongúkat egy 19x19-es négyzetrácson helyezik el ezért az állapottér bonyolultság $3^{361} \approx 10^{172}$. A játékfa bonyolultságának a meghatározására feltételezzük, hogy egy játék átlagosan 150 lépésből áll és az elágazási tényező 250. Ekkor a komplexitás: $250^{150} \approx 10^{360}$.

3.3.7. Olimpiai játékok

Azok a játékok amelyek részt vesznek az évente megrendezett világbajnoksági versenyeken. Általában a kétszemélyes, teljes információjú táblás játékok ezek (a 3.1. ábrán

feltüntetett játékok mind ide tartoznak), de vannak nem teljes információjú szerencsejátékok is. Ezek nagyrészt a kártyajátékok (pl. Bridge) és a dobókocka alapú játékok (pl. Backgammon). A kutatók által egy játék számára kifejlesztett stratégiákat itt tesztelik le. Az olimpiai játékok teljes listája az [Allis94] doktori tézis 6.3. fejezetében található.

3.4. Minimax algoritmus

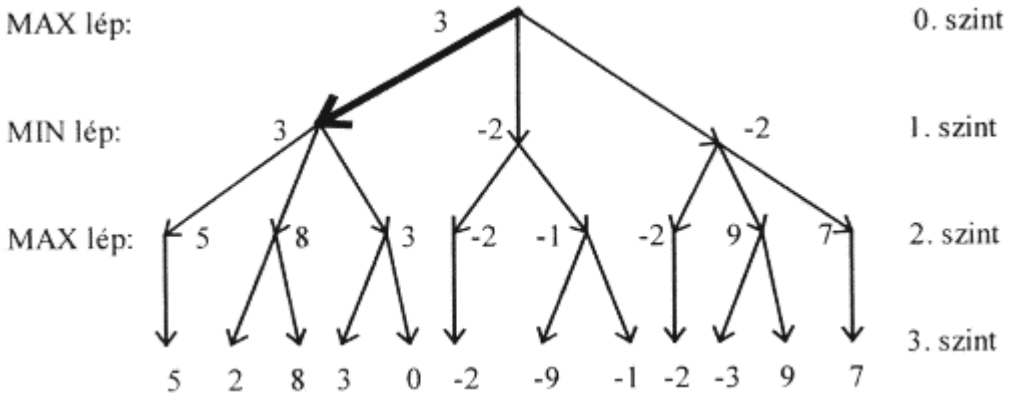
A játékok nagy kihívást jelentenek a számítástechnika és ezen belül a mesterséges intelligencia számára, ugyanis a bonyolultabb játékok keresési tere végtelenül nagy, még a ma létező számítógépnek is komoly problémát jelent egy mesteri szinten játszó stratégia előállítása. A játékok (elsősorban a teljes információjú táblás játékok) tulajdonképpen **produkciós (keresési) rendszer**ek, hiszen a játékok állapottere lépéssorozatok permutációja, ezért a cél nem más mint a legjobb lépéssorozat megkeresése ebben a roppant nagy halmazban. A kutatások során nagyon sok **vezérlési stratégiá**t, azaz keresési módszert (nem informált, informált, heurisztikus, gráfkereső) fejlesztettek ki, amelyek elsősorban a keresési teret próbálják szűkíteni.

A játékok (főleg a kétszemélyes, teljes információjú) reprezentálására a reprezentációs gráfot vagy más néven a **játékfá**t használják. A teljes játékfa tartalmazza a játék összes lehetséges állapotai esetében az összes lépéslehetőséget (ismétlődő állapotok is lehetnek, hiszen több után eljuthatok ugyanabba az állapotba), ezért a fának a mérete egyenesen arányos a játék bonyolultságával. A két játékos lépései páros illetve páratlan szinteken váltják egymást. Az előző részben láttuk, hogy a sakk játékfájának a bonyolultsága 10^{20}. Az analógia kedvéért a Föld kialakulása óta 4,6 milliárd év telt el, ami 10^{18} másodperc. Ha a számítógép 100 állást vizsgál meg másodpercenként, akkor egy földtörténeti időre lenne szükség a játék teljes elemzéséhez, persze, ha ezt a fát valamilyen formában fel is tudnánk építeni.

Tehát a **játékfa teljes mélységű kiértékelése** lehetetlen, ezért egy **részleges kiértékelés**t végzünk. Ekkor, viszont nem a **biztos nyerő stratégiá**t határozzuk meg, hanem a soron levő lépést, az adott állásból kiindulva. Ehhez használjuk a minimax eljárást.

A minimax eljárás a soron következő játékos (nevezzük ezt *MAX*-nak, ellenfelét pedig *MIN*-nek) **legjobb, legkedvezőbb első lépés**ét választja ki. Az adott állásból egy korlátos mélységig felépítjük a játékfát és annak leveleire alkalmazunk egy úgynevezett **statikus kiértékelő függvény**t. Ez a függvény értékeli az illető csúcsban levő állás „jóságát”, azt, hogy ez mennyire jó a *MAX* játékos nyerése szempontjából és a függvény értéke csak az adott

állapottól függ. Konvenció szerint a *MAX* szinten levő csúcsokhoz pozitív értékeket rendelünk, a *MIN* szinten negatívakat, döntetlen játék esetén pedig nulla értéket. Tekintsük a 3.2. ábrán látható 3 mélységű játékfát, és feltételezzük, hogy a leveleket már kiértékeltük.



3.2. ábra. Egy 3 mélységű fiktív játékfa minimax kiértékelése. A *MAX* játékos legjobb első lépését a vastag vonal jelöli.

Mivel a 2. szinten a *MAX* lép, ezért a rákövetkező csúcsok (a mi esetünkben ezek levelek) értékeinek maximumát adjuk vissza ezeknek a *MAX* csúcsoknak. Az 1. szinten levő *MIN* csúcsoknak a 2. szinten található minimális értékeket választjuk ki. A végén a gyökérelem kap értéket újból a maximum-képzés elve szerint. A **visszaadott érték** a 3 és a *MAX* legjobb első lépését a vastag vonal jelöli. Ha a *MAX* a jobboldali lépést választaná akkor elérhetne egy 9 vagy 7 értékű állásba, de fennáll annak a veszélye, hogy a -2 értékű állásba jusson. Ezért van az, hogy a minimax lepés a **legnagyobb, de kockázatmentes, biztos előnyszerzés elv**ét követi.

Általánosítva a fenti műveletsort, a minimax módszer egy sematikus algoritmusa a következőképpen néz ki:

1. Ha n terminális csúcs, akkor $v(n) := f(n)$, ahol $f(n)$ a statikus kiértékelő függvény;
2. Ha n nem terminális és rákövetkezőit $n_1,...,n_k$ jelöli, akkor

   $v(n) := \max(v(n_1),\ldots,v(n_k))$, amennyiben n páros szinten van, illetve

   $v(n) := \min(v(n_1),\ldots,v(n_k))$, ha n páratlan szinten található.

Ezzel az eljárással végül a fa s-el jelölt gyökérelemének is értéket adunk. A *MAX* számára a legjobb első lepés egy olyan, s-ből kiinduló él, amely egy $v(s)$ értékű rákövetkezőjéhez vezet [Fekete99], [Nagy99].

A minimax eljárás egyszerűsített változata a negamax algoritmus, a hatékonyabb változata pedig az alfa-béta vágás módszere.

3.5. Nash egyensúly

Ezt az egyensúlyt a megalapítójáról, John Nash-ről, a Nobel-díjas matematikusról nevezték el, aki 1950-ben a legtöbbet tett, hogy kibővítse és általánosítsa Neumann és Morgenstern játékelméletét.

A játékelmélettel foglalkozó kutatók egy játék megoldására úgy hivatkoznak mint a rendszer egyensúlyi helyzetére (*equilibria*). Egy fizikai rendszer egyensúlyban (*equilibrium*) van ha stabil állapotban van, ami azt jelenti, hogy a rendszer belső erői, energiái kiegyenlítik egymást és nyugalmi állapotban hagyják a rendszert amíg külső erők nem zavarják meg.

Egy stratégiahalmaz Nash egyensúly (***N**ash **E**quilibrium – NE*), ha egyik játékos sem tud javítani a pontszámain a saját stratégiainak a módosításával és a többi játékos stratégiája adva van. Észrevehető, hogy egy szigorúan uralkodó stratégia nem lehet NE stratégia. Így a szigorúan dominál stratégiák kizárása egy minimális követelménye a racionalitásnak. Ez azt jelenti, hogy ha egy játéknak a végeredménye egy egyedülálló NE, akkor ez a játéknak az egyetlen (*unique*) megoldása.

A véges, kétszemélyes, teljes információjú, zérusösszegű játékok esetében az NE nem csak szükségszerű hanem elégséges feltétele a megoldás létezésének. Ezekben a játékokban az egyik játékos akkor lehet jobb, ha a másikat arra kényszeríti, hogy rosszabbul csinálja (pl. a Tic-Tac-Toe esetében: minden lépés ami közelebb viszi az egyik játékost a nyeréshez, addig a másikat a vesztés felé kényszeríti és fordítva). Egy zérusösszegű játékban, ha mind a két játékos olyan stratégiát játszik ami maximalizálja a minimális költségeket (*payoff*), akkor a játékosok a legjobb stratégiát játsszák, garantálva ily módon a játék egyetlen megoldásának a megtalálását, ami nem más mint a játékhoz tartozó egyetlen NE. A TTT-nél ez a döntetlennek felel meg. Egyik játékos sem tud jobb eredményt elérni a döntetlenen kívül, ha mindkettő azt akarja, hogy nyerjen és ne veszítsen [Ross02].

4. Evolúciós módszerek alkalmazása játékelméleti problémákra

A 2. fejezetben bemutatott evolúciós módszerek, különösen a genetikus algoritmusok igen hatékony optimalizálóknak bizonyultak. De ennél is fontosabb az a tulajdonságuk, hogy általános struktúrájuk van, nem használnak terület-specifikus ismereteket és jól megbirkóznak a nehéz, azaz bonyolult feladatokkal. A játékok pont ilyen feladatokat képviselnek, éppen ezért alkalmazhatók az evolúciós módszerek.

Az intelligenciához hozzátartozik, hogy egy cél elérése érdekében egy rendszer megfelelő döntést hoz és ugyanakkor képes adaptálódni a környezetéhez. A döntéshozatal egy komplex környezetben nemlineáris inger-válasz leképezést igényel. A cél, kiválasztani a legmegfelelőbb reprezentációt és tanulási algoritmust.

Ebben a fejezetben megvizsgáljuk, hogy milyen mértékben és hogyan alkalmazhatók ezek a módszerek konkrét játékokra, bemutatva az eddig elért kutatási eredményeket. Egy ilyen esetben (mint azt láttuk a genetikus algoritmusoknál) a legnehezebb a megfelelő reprezentációt és rátermettségi függvényt megadni. Ezen múlik a stratégia hatékonysága.

A játékok amelyeken alkalmazzuk az evolúciós módszereket különböző típusúak, egyszerű leírással és szabályokkal rendelkeznek. Az iterált fogolydilemma egy kétszemélyes, nem teljes információjú, nem zérusösszegű stratégiai játék, amely inkább egy szimulációs modell. Az amőba egy kétszemélyes, teljes információjú, zérusösszegű stratégiai táblás játék (*board game*). Az 5. fejezetben pedig a kirakós játékra alkalmazzuk a genetikus algoritmusokat.

4.1. Evolúciós próbálkozások

Már a 80-as években voltak olyan kísérletek, ahol a genetikus algoritmusokat megpróbálták alkalmazni a gépi tanulásra. Ezek a próbálkozások a GA megalapítóihoz fűződnek: [DeJong88], [Goldberg88]. Később a GA-t a szimbolikus tanulás módszereivel ötvözték. [Janikow93] 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 applikációval bebizonyította, hogy a GA képes magas szintű fogalmak

feldolgozására. [Whitley93] és társai 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).

Számos olyan próbálkozás van, ahol direkt vagy indirekt (hibrid) módon felhasználták az evolúciós módszereket, azért hogy a játékok számára jobb stratégiát állítsanak elő. A direkt azt jelenti, hogy valamilyen evolúciós számítással (GA, GP, EP) kifejlesztettek egy stratégiát. 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.

A neuronhálós megoldást kevesen alkalmazták és csak egyszerű játékokra (ahol a keresési tér kicsi), hiszen a GA és neuronháló szimbiózisa hihetetlenül lassú lehet. Az egyik ilyen próbálkozás Fogel nevéhez fűződik és a 3x3-as amőbára (TTT) alkalmazta ([Fogel93b] és lásd a [Fogel00] monográfiában az *5.2 General Problem Solving: Experiments with Tic-Tac-Toe* fejezetet). [Chellapilla99a] és társai több játékra is használták: fogolydilemma, TTT és dámajáték (*Checker*). A [Chellapilla99b] és [Kim02] cikkek azt írják le, hogy az evolúciós algoritmus talált egy olyan neuronhálót, ami segítségével a Checker majdnem szakértői, profi szinten játszik. [Pollack97], [Pollack98] pedig egy jobb Backgammon stratégia előállítására használták ezt a hibrid módszert. [Konidaris02] genetikus algoritmust használ a neuronháló tanítására, ami a Go egy egyszerűbb változatát képes játszani.

Az egyszerű evolúciós módszereket nagyon sok játékra alkalmazták. [Ferrer95] genetikus programozást használt egy táblás játék értékelő függvényének a fejlesztésére. [Faybish99] genetikus algoritmussal Othello stratégiát fejleszt ki. [Kojima98] pedig a Go-t tanulmányozza egy evolúciós megközelítésben.

[Hong02] sajátos módon ötvözi a GA-t és az egyszemélyes játékfát, a legjobb első lépés meghatározásának érdekében. A cél a játékfa bejárásának az elkerülése ami nagyobb mélységnél roppant lassú művelet. A kifejlesztett módszer teljesítményét összehasonlítja a hagyományos minimax eljárással a 9-es (3x3-as kirakós) játékot használva. A kísérletek eredményei azt mutatják, hogy ez a módszer által igényelt idő és memória jóval kevesebb mint a minimax esetében.

4.2. Iterált fogolydilemma

„Egy személynek mikor kell együttműködőnek és mikor kell önzőnek lennie egy másik személlyel egy folyamatos interakcióban? Milyen feltételek mellett fog az együttműködés felbukkanni egy egoista világban központi hatalom nélkül?" *("When should a person cooperate, and when should a person be selfish, in an ongoing interaction with another person? Under what conditions will cooperation emerge in a world of egoists without central authority?"*). Ezekre és sok más kérdésre keresett választ Axelrod 1984-ben az [Axelrod84]-es munkájában.

Ezt a híres társadalmi játékot Merill M. Flood és Melvin Dresher a RAND cég munkatársai vezették be 1952-ben azzal a céllal, hogy egy kis irracionalitást vigyenek be a játékelméletbe [SMAC99b]. Azóta világszerte több kutató foglalkozott a problémával és manapság, az Internet rohamos fejlődésével, még aktuálisabb témává vált.

4.2.1. A játék általános bemutatása

A feladat jobb megértése végett lássuk a következő analógiát. A játék Albert Tucker egyszerű történetén alapul. Tegyük fel, hogy két személyt éppen most tartoztattak le egy korábban elkövetett bűncselekedetért. Börtönbe zárják őket, de két különböző cellába, úgy, hogy nem tudnak egymással kommunikálni. A vádló az egyiknek egy alkut ajánl: ha mind a ketten ártatlannak vallják magukat akkor 1 év börtönbüntetést kapnak, ha viszont bűnösnek vallják magukat akkor 5 évet kapnak. Ellenben, ha úgy dönt, hogy elpártol, beárulja társát és tanúskodik ellene, akkor szabadon engedik, míg a társát 25 év szabadságvesztésre ítélik. A vádló ugyanezt az ajánlatot teszi a másik fogolynak is. A 4.1. táblázat vázolja az előbb leirt problémát [Generation03f].

4.1. táblázat. A börtönbüntetések terjedelme a két fogoly kölcsönös döntésétől függ, attól, hogy minek vallja magát.

		Második fogoly bevallása	
		Ártatlan	Bűnös
Első fogoly bevallása	Ártatlan	**1 év**	**25 év**
	Bűnös	**0 év**	**5 év**

Ezt a feladatot nevezik a fogolydilemmának (***P**risoner's **D**ilemma – PD*), hiszen több kérdés merül fel, mi több a fogoly dilemmában van. Milyen döntést hozzanak a foglyok külön-külön nem ismerve a másik szándékát? Milyen magatartásformát válasszon a fogoly?

A dilemma onnan fakad, hogy az egyik fogoly döntése a másik fogoly cselekedetétől függ, és ezt nem ismeri, addig amíg nem hoz egy döntést. Jól látszik a 4.1. táblázatból, hogy az elpártolás, azaz bűnösnek vallani magát a legésszerűbb döntés, hiszen ekkor maximálisan 5 évet kapna, ha a másik fogoly is ezt választotta, illetve lehetősége nyílik arra, hogy visszakapja a szabadságát. Kooperatív cselekedet mellett a fogoly legrosszabb esetben 25 évet kapna, legjobb esetben pedig 1 évet, ami jóval rosszabb mint az elpártolás esetében.

Most tekintsünk el az előbbi valós esettől, tegyük fel, hogy egy játékról van szó, ahol a büntetési évek pontszámoknak felelnek meg és a cél az, hogy a két játékosnak minél kevesebb pontja legyen. Fontos még hozzátenni, hogy miután mind a két játékos hozott egy döntést, azaz választott kooperáció és elpártolás közül, az eredményeket közöljük a másik féllel. Mivel ez nem olyan érdekes, ismételjük meg az előbbi folyamatot többször. Iteráljuk a fogolydilemmát egy előre nem meghatározott számszor. Egy új feladatot kapunk, amely az iterált fogolydilemma (***I**terated **P**risoner's **D**ilemma – IPD*) nevet viseli. Ekkor egy játékos végeredménye a lépeseiből származó pontszámok összege lesz. Milyen döntéseket hozzanak a játékosok ekkor? Mi a legmegfelelőbb cselekedet, ha azt akarjuk, hogy a két játékos által külön-külön összegyűjtött pontszám minimális legyen? Ez még jobban bonyolítja a fogolydilemmát, hiszen az egyik játékos által hozott döntés befolyásolja a partnere (a másik játékos) döntését a későbbi lépésekben.

Az IPD fontos kutatások alapját képezi, hiszen különböző társadalmi, gazdasági, katonai és politikai interakciók szimulációjává vált. A PD esetében a megoldás a kölcsönös elpártolás volt, ami tulajdonképpen a játék Nash egyensúlya (lásd a 3.5. fejezetet), de ez a cselekedet az IPD esetében nagyon hatékonyatlan, hiszen a kooperációval ennél sokkal jobb eredmények érhetőek el mindkét fél által.

4.2.2. A játék stratégiái

A fogolydilemma egy kétszemélyes, nem zérusösszegű, nem teljes információjú, azaz nem kooperatív játék. A zérusösszegű fogalom azt jelenti, hogy ha az egyik játékosnak bármilyen előnye származik, a másikra szükségszerűen nem hat ki ugyanolyan mértékű büntetéssel. A nem kooperatív fogalom pedig azt jelenti, hogy a játékosok közötti kommunikáció nem megengedett, a döntés eredményeit a játékosok csak a játék végén tudják meg (lásd a [Fogel00] könyv *5.3 The Prisoner's Dilemma: Coevolutionary Adaptation* fejezetét).

Egy játék a fogolydilemmában két lépést jelent: a két játékos lépése külön-külön, ami tulajdonképpen egy választás, döntés az együttműködés és elpártolás között.

Ahhoz, hogy a játékot könnyebben lehessen kezelni, Axelrod az összes lehetséges döntés-párokat paraméterként kezelte, ezeknek nevet adott, ezek között bizonyos összefüggéseket határozott meg és pontokkal jutalmazta a játékosok magatartásformáit (kooperáció vagy elpártolás). A 4.2. táblázat tartalmazza ezeket a pontszámokat. Mivel ez mátrix alakú, ezért fizetség vagy jutalom (*payoff*) mátrixnak szokták nevezni.

4.2. táblázat. A fogolydilemma jutalom mátrixa.

		Második játékos	
		Együttműködik (***C**ooperate*)	Elpártol (***D**efect*)
Első játékos	Együttműködik (***C**ooperate*)	**(3 , 3)** Jutalom a kölcsönös együttműködésért (***R**eward for mutual cooperation*)	**(0 , 5)** Balek fizetsége és csábítás az elpártolásra (***S**ucker's payoff, and **T**emptation to defect*)
	Elpártol (***D**efect*)	**(5 , 0)** Csábítás az elpártolásra és a balek fizetsége (***T**emptation to defect, and **S**ucker's payoff*)	**(1 , 1)** Büntetés a kölcsönös elpártolásért (***P**unishment for mutual defection*)

A feladat akkor fogolydilemma, ha teljesíti a következő feltételeket:

$T > R > P > S$ és $2R > S + T$,

vagyis a mi esetünkben: $5 > 3 > 1 > 0$ és $2 \cdot 3 > 0 + 5$, ami az összes feltételre igaz, tehát valóban egy fogolydilemmáról van szó. A második feltételre azért van szükség, hogy favorizáljuk, támogassuk a kooperációt és az sem kedvez a játékosoknak ha felváltva együttműködnek illetve elpártolnak.

Mivel egyik játékos (vagy általánosabban ágens) sem tudja, hogy mikor ér véget a játék, így lehetőség van arra, hogy megvizsgáljuk az ágensek stratégiáit, azt, hogy hogyan és mikor próbálnak együttműködni. A stratégiák magatartásformáinak (*behaviour*) a tanulmányozására kétfajta becslést lehet végezni:

1. **Round robin verseny** (*tournament*), amikor is minden stratégia játszik mindegyikkel. A stratégia végeredménye az összes szembesítésből származó pontok összegével egyenlő. Ezt a módszert alkalmazta Axelrod;
2. **Ökológiai fejlődés** (*ecological evolution*), ahol egy populációban az egyedek (a jelen esetben stratégiák) egyenlő számban képviseltetik magukat, majd ezek között egy round robin versenyt rendezünk. A nyertes egyedek számát növeljük, a vesztesekét pedig csökkentjük. Ezt a folyamatot addig iteráljuk amíg a populáció

stabilizálódik (nincs több változás). A rosszhiszemű stratégiák (akik először választják az elpártolást) nem stabilak, eltűnnek és helyüket a jóindulatú egyedek veszik át. A módszert a [SMAC99ab] francia kutatócsoport alkalmazta.

Axelrod 1979-ben felkérte 5 tudományág (pszichológia, közgazdaságtan, politikai tudományok, matematika, szociológia) tudósait, kutatóit, hogy írjanak játszási stratégiákat a fogolydilemmára. Egyesek igencsak bonyolultak voltak: például az egyik stratégia Markov-folyamatokat használt, hogy modellálja az ellenfél viselkedését, majd a Bayes-következtetés segítségével kiválasztotta a legjobbnak tűnő lépést [Axelrod87]. A benyújtott 14 stratégiához hozzávette az úgynevezett RANDOM stratégiát, ami véletlenszerűen játszott, minden lépésben 0,5-ös valószínűséggel együttműködött vagy elpártolt. Mind a 15 stratégia játszott mindegyikkel egy **round robin** versenyben: összesen 120000 darab egyszeri játék, azaz 240000 önálló választás volt végrehajtva. A nyertes az úgynevezett FOGAT FOGÉRT (**T**IT **F**OR **T**AT – TFT) stratégia volt 504 ponttal. Ezt a stratégiát Anatol Rapoport készítette, a Torontoi Egyetem pszichológus professzora. Egy 200 lépést tartalmazó játékban a legjobb elérhető összpontszám 600, ami azt jelenti, hogy minkét játékos minden lépésben együttműködött, azaz 3-3 pontot kapott [Axelrod84].

Később Axelrod újabb versenyt rendezett, ahol 62 stratégia volt jelen és ekkor is a TFT nyert.

Lássunk néhány gyakori IPD stratégiát:

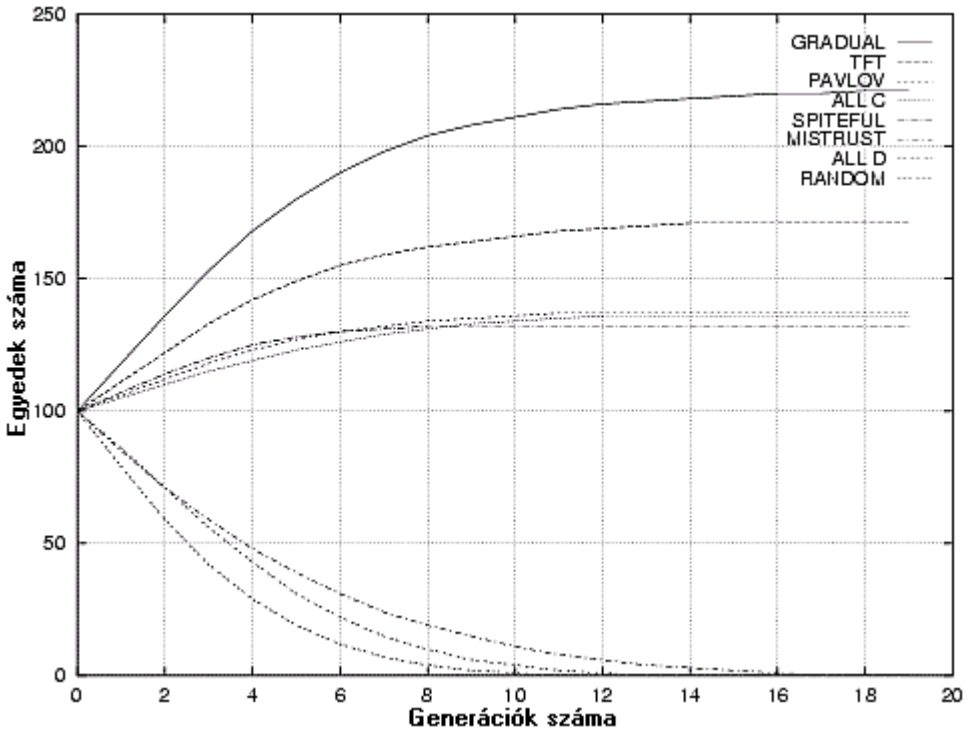
1. TIT FOR TAT: kooperációval kezdi, majd azt lép amit a partnere az előző lépésben választott.
2. ALL C: mindig együttműködik.
3. ALL D: mindig elpártol.
4. SPITEFUL: kooperál amíg a másik fél el nem pártol, azután mindig elpártol.
5. PAVLOV: akkor és csak akkor kooperál, ha az előző lépésben mindketten ugyanazt az opciót választották.
6. TIT FOR TWO TATS (TF2T): kooperál, kivéve, ha az ellenfél kétszer egymás után elpártolt.
7. MISTRUST: elpártol, majd azt lép amit az ellenfél az előző lépésben választott.
8. RANDOM: véletlenszerűen játszik.

A TF2T stratégia még a TFT-nél is hatékonyabb, de ez nem vett részt az Axelrod által rendezett versenyekben [Generation03f]. Általában azok a stratégiák amelyek kooperációval kezdik jobb eredményt érnek el. Ahhoz, hogy egy stratégia jó eredményt érjen, eleget kell tegyen a következő követelményeknek:

- Legyen kedves, azaz ne legyen az első aki elpártol,
- Legyen reaktív, válaszoljon az együttműködésre illetve az elpártolásra
- Bocsásson meg, és
- Ne legyen túl okos, vagyis legyen egyszerű, hogy az ellenfelek megérthessék.

A TFT teljesítette ezeket a tulajdonságokat, ezért volt olyan hatékony.

Évekig azt gondolták, hogy a TFT a legoptimálisabb stratégia. Később Axelrod egy olyan GA programot irt, ahol jobb stratégiát fejlesztett ki (lásd a 4.2.3. fejezetet), a SMAC csoport pedig kitalált egy újabb stratégiát, ami szintén legyőzte a TFT-t. A GRADUAL stratégia abban áll, hogy először kooperál, majd az ellenfél első elpártolásánál elpártol egyszer és kooperál kétszer, a második elpártolásnál kétszer pártol el és megint kétszer kooperál és így tovább. Általánosan fogalmazva, az ellenfél *n*-ik elpártolásánál elpártol *n*-szer – legyen ez a megtorlás szakasza (*punishment period*) és kooperál kétszer – nevezzük ezt tűzszünetnek (*lull time*). Ez a stratégia jobb eredményeket produkált mint a TFT. A 4.1. ábrán a különböző stratégiák teljesítménye látható. Ezek a stratégiák egy 19 iterációs ökológiai fejlődésen vettek részt, amelyet a SMAC által készített program (akár 37 stratégiát is képes összehasonlítani) szimulált.



4.1. ábra. Az egyedek számának a változása egy 19 generációs ökológiai fejlődés során. Az elején mind a 8 stratégia 100 egyed által képviseltette magát.

[Beaufils97] a GRADUAL stratégiát továbbfejlesztették, úgy, hogy egy genetikus algoritmussal optimalizálták.

4.2.3. Stratégia fejlesztése genetikus algoritmussal

Axelrod, felhasználva a genetikus algoritmusokat, programot írt, amellyel fogolydilemmát játszó stratégiát fejlesztett ki. A továbbiakban a genetikus algoritmus elemeit ismertetjük [Axelrod87], [Michalewicz96] és [Fogel93a] alapján.

4.2.3.1. Reprezentáció

A könnyebb kezelés érdekében jelöljük az együttműködést 1-el (és ne C-vel) míg az elpártolást 0-val (D helyett). Egy játékos az aktuális lépésben az előző 3 játék (azaz 6 darab lépés) eredményeinek alapján hoz döntést. Ha vesszük a 6 hosszúságú bit stringnek (a 3 egymás utáni játékból származó 0-1-esek) az összes lehetséges variációját, akkor ez azt jelenti, hogy 2^6-on azaz 64 darab különböző bitsorozat létezik. Ezt a következőképpen is fel lehet fogni: mivel egy játék alkalmával 4 darab különböző eredmény (CC, CD, DC, DD vagy 11, 10, 01, 00) lehetséges, így 3 egymás utáni játékból 4^3=64 darab eredmény származik.

A stratégia nem más mint egy 64 hosszúságú bit string, amely tartalmazza az összes lehetséges játéksorozatra adott választ. Ehhez hozzáadjuk a kezdeti 6 darab lépést, és így a kromoszóma egy 70 hosszúságú bitsorozat lesz. Következésképpen a keresési tér $2^{70} \approx 10^{21}$ nagyságrendű, ami ennyi darab különböző stratégiát jelent. Egy kimerítő (*exhaustive*) keresés ki van zárva, hiszen ha egy komputer 100 stratégiát vizsgálna meg másodperceként akkor is kevesebb mint 1%-át ellenőrizte volna le a világ kezdetétől [Axelrod87]. Éppen ezért van szükségünk egy olyan erős és hatékony technikára mint a GA.

4.2.3.2. Rátermettségi függvény

20 darab véletlenszerűen kiválasztott stratégia (ez tulajdonképpen az egyedek száma a populációban) játszott a 62-ből kiválasztott 8 darab stratégia mindegyikével. Egy stratégia-pár 151 darab lépést játszott le. Az eredményeket a jutalom mátrix alapján összegezve, megkapjuk az egyed (ami a mi esetünkben egy stratégia) rátermettségét.

4.2.3.3. Genetikus operátorok

Az egypontos keresztezést és mutációt a klasszikus formájukban alkalmaztuk a 0-1-es bitekből álló kromoszómákra. A keresztezési és mutációs valószínűségek úgy voltak meghatározva, hogy átlagosan 1 keresztezés illetve ½ mutáció legyen egy kromoszómára végrehajtva egy generáció alatt.

4.2.3.4. Szimulációs eredmények

A GA iteráció száma 50 volt. Egy generáció alatt a 20 egyed a 8 stratégiával több mint 24000 lépést játszott le. A program 40-szer volt végrehajtva ugyanazokkal a feltételekkel. A kapott eredmények figyelemreméltóak voltak: egy teljesen véletlenszerű populációból olyan egyedek fejlődtek ki, amelyeknek az átlagrátermettsége (az adott esetben az átlagpontszáma) közel voltak a TFT eredményeihez. A kromoszómákban 5 viselkedési allél alakult ki, ami 5 viselkedési mintának felel meg:

1. Ne hintáztasd a csónakot (*Don't rock the boat*): együttműködés folytatása 3 kölcsönös kooperálás után (vagy formálisan C RRR után, azaz C (CC)(CC)(CC) után).
2. Legyél provokáló (*Be provokable*): elpártolás, ha a másik fél váratlanul elpártolt (D RRS után, vagy D (CC)(CC)(CD) után).
3. Fogadj el egy bocsánatot (*Accept an apology*): együttműködés folytatása miután a kooperáció helyreállt (C TSR után, vagy C (DC)(CD)(CC) után).
4. Felejts (*Forget*): együttműködés miután a kooperáció helyreállt egy kizsákmányolás után (C SRR után, vagy C (CD)(CC)(CC) után).
5. Fogadd el a kitaposott utat (*Accept a rut*): elpártolás 3 kölcsönös elpártolás után (D PPP után, vagy D (DD)(DD)(DD) után).

A kifejlesztett szabályok nagyrészt úgy viselkedtek mind a TFT. Akárcsak a TFT, a 8-ból 7 stratégiával majdnem kölcsönös kooperációt alakítottak ki. A 40 tesztből 11 esetében a GA által kifejlesztett stratégia jobb eredményt ért el mint a TFT: az eredmény középértéke 450 pont volt a TFT 428 pontjához képest. A többi esetben az eredmények közel voltak a TFT eredményeihez.

Aszexuális reprodukció, azaz keresztezés nélküli tesztek esetében csak feleannyi sikeres populáció van mint a szexuális reprodukcióval, azaz a 40 tesztből csak 5 alkalommal fejlődnek ki olyan stratégiák, amelyek jobbak mint a TFT.

Eddig a populáció egy konstans környezetben fejlődött. Mi van akkor, ha a környezet is folyamatosan változik? Ennek érdekében Axelrod egy újabb kísérletet végzett, ahol a környezetet maga a fejlődő populáció képezte. Ekkor a populáció minden tagja (a 20 egyed) egymás között játszik IPD-t. Ahhoz, hogy egy egyed sikeres legyen, az általa elért átlagpontszám magasabb kell legyen mint az összes többi 19 egyed átlagpontszáma.

Az elején az elpártoló viselkedés a népszerű. Ez azért van, mert még kevés olyan egyed van akik értékelnék a kooperatív kezdeményezéseket. Az alacsony szintű kooperáció

következtében az egyedek kevés pontot kapnak, mivel a kölcsönös elpártolás mind népszerűbb. 10-20 generáció után ez a tendencia megfordul, mivel néhány egyed kifejlesztett egy kooperatív viselkedést és pozitívan válaszolnak az együttműködésre. A populáció átlag pontszáma elkezd növekedni és a kölcsönösségen alapuló kooperáció elterjed az egyedek között.

A bemutatott genetikus szimulációk roppant absztrakt rendszerek. A populáció nagyon kicsi és a generációk száma is kevés. A genetikus folyamatnak két operátora van, a rekombinációnál nincs nemi megkülönböztetés és mindig két utód származik egy párosodás során. Ezek ellenére a szimulációs rendszer képes kifinomult, adaptív stratégiák kifejlesztésére egy viszonylag komplex környezetben.

4.2.4. Magatartásformák fejlesztése az evolúciós programozás módszerével

Fogel a [Fogel93a] cikkében megpróbált egy jobb stratégiát kifejleszteni az evolúciós programozást alkalmazva. Ennek érdekében egy 8 állapotot tartalmazó véges automatát (lásd a 2.11. fejezetet) épített fel, amelynek a bemeneti ábécéje a {(C, C), (C, D), (D, C), (D, D)} halmaz, a kimeneti ábécéje pedig a {C, D} állapotok halmaza volt. Mindegyik állapot össze volt hasonlítva mindegyikkel (round robin). A használt evolúciós algoritmus struktúrája megegyezett azzal amit a 2.11.2. fejezetben megismertünk. A véges állapotú automaták kifejlesztéséhez [Chellapilla99a] szerint a következő lépések szükségesek:

1. Kezdeti populáció inicializálása véletlen FSM-ekkel. Minden állapotra és minden bementi szimbólumra (a játékosok lépései az előző játékban) generálj egy új lépést (C vagy D) és egy állapot-átmenetet (*state transition*).
2. Minden FSM-párra játszódj le 151 lépésből álló játékot. Számold ki minden FSM által elért átlagpontszámot az összes játék alatt.
3. Szelekció alkalmazása a populációra, a leggyengébb egyedek (automaták) kizárása.
4. Mutáció (lásd a 2.11.1. fejezetet) alkalmazása a megmaradt egyedekre. A leszármazottak az új generációban a szülők szerepét töltik be.
5. Menj a 2. lépéshez és iteráld amíg a rászánt idő nem járt le.

Az algoritmus 200 generációból állt, és a populáció 100 egyedet tartalmazott. Az eredmény átlagértéke 3 volt, ami minden lépésben együttműködést jelentett.

Az együttműködési magatartás kialakítása függ a jutalom mátrixtól és a mátrix paraméterei közötti összefüggésektől. Ha a jutalom mátrixban szereplő R, S és P paraméterek konstansok akkor T-től (T=5,25/5,5/5,75/6) függ a kooperatív vagy önző viselkedés. Minél

nagyobb értéke van *T*-nek, annál kisebb mértékben fordul elő kooperáció. A paraméterek közötti összefüggések módosítása 3 kritikus zónához vezetett:

1. Ha $R > (S+T)/2$ akkor a kooperáció a jellemző;
2. Ha viszont $R < (S+T)/2$ akkor kevés a kooperáció, az önző magatartás dominál;
3. És végül, ha $R = (S+T)/2$ akkor egyenlő arányú együttműködés és elpártolás észlelhető.

[Chellapilla99a] és társa megpróbálták az evolúciós algoritmusok és neuronhálók segítségével egy jobb 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.

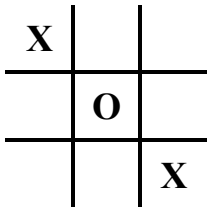
4.3. Amőba

Több ötlet is született arról, hogy hogyan lehetne a TTT (Tic-Tac-Toe) számára jó stratégiát kifejleszteni evolúciós algoritmusokkal. Mivel a játék nagyon egyszerű és a keresési tér is kicsi, így számos új módszert elsősorban erre a játékra alkalmaznak és ezzel végzik a kísérleteket is.

Mielőtt rátérnénk az amőbára alkalmazott evolúciós technikákra, nézzük meg, hogy milyen módszerekkel próbálkoztak a gépi tanulás területén.

4.3.1. A gépi tanulás módszereinek az alkalmazása

[Epstein92] és [Epstein94] cikkekben a szerző azt mutatja be, hogy sokszor a tanuláshoz a tervezés fölösleges, az igényelt memória ezért minimális, illetve azt, hogy a szembeállítás hogyan befolyásolja a tanítási folyamatot. A felhasznált tanítási módszer az úgynevezett **versengő tanulás** (*competitive learning*), ahol az oktató (*trainer*) pozitív-negatív példákon keresztül vagy egy viselkedési modellen keresztül tanítja a tanulót (*learner*). A kísérletek elvégzésére egy Hoyle nevezetű tanuló programot implementált, ami a kétszemélyes, teljes információjú, véges táblajátékok (többek között a TTT) számára fejleszt ki stratégiákat. Megadva az új játék definícióját, a program elkezd játszani különböző képességű ellenfelekkel (újonc, véletlen, szakértő), a játékok végén egy kiértékelő az eredmények függvényében pontokat ad, így a program fokozatosan fejlődik, azaz tanul. Egy szakértő (*expert*) stratégia megbízható – nem követ el hibákat, bármenyire tapasztalt is lenne az ellenfél – és erős – teljes mértékben kihasználja az ellenfél hibáit – kell legye. Habár a program sok játékot meg tud tanulni szakértői szinten, mégis a 4.2. ábrán látható állapot problémát jelent számára, ahol az ellenfél által kialakított szétágazás (*fork*) félrevezeti a programot és a sarkok egyikére tesz.



4.2. ábra. Egy nehéz helyzet Hoyle és a legtöbb ember számára, ahol helytelenül játszanak. A helyes lépés *O* számára egy nem sarkon levő pozíció.

Egy tipikus gépi tanulásos rendszerben egy tanárt (edző környezet) használnak a közvetlen visszacsatolás szolgáltatásának érdekében. Sokszor ez a tanár hiányzik, vagy megépítése sokba kerül. Ezért használják a felügyeletlen tanulást, ami a játékok esetében az önjátszást (*self-play*) jelenti.

[Angeline93a] TTT játékost fejleszt ki a GliB (lásd a következő részt) és önjátszás segítségével. A rátermettség mérésére rögzített és véletlen stratégiákat használ, amelyekkel adott számú játékot játsztat le a fejlődő stratégiával.

[Freisleben96] egy megfelelő neuronhálót használ a szabad pozíciók kiértékelésére és ezt a hálót a megerősítésen alapuló tanulási módszerrel tanítja egy sor játékon keresztül.

4.3.2. Stratégia fejlesztése genetikus programozással

[Angeline92] és [Angeline94] cikkekben a GliB (***G**enetic **Li**brary **B**uilder*) programcsomagot építik meg és ezt alkalmazzák különböző feladatok megoldására. A GliB egy dinamikus genetikus algoritmus, ami a genetikus programozás paradigmára épül. A GliB abban különbözik a GP-től, hogy tartalmaz két újszerű mutációs operátort. Az egyik a tömörítés (*compression*) operátor, ami kivonatolja a genetikai anyagból a környezet specifikus kiegészítéseket a primitív nyelv (LISP) számára. A genotípus minden tömörített része – nevezzük ezt modulnak (*module*) – növeli a nyelv kifejezőképességét és csökkenti a genotípus átlagos méretét a populációban. A második operátor, a kiterjesztés (*expansion*) a tömörített modult helyettesíti az eredeti definíciójával. Mint minden mutáció, a tömörítés és kiterjesztés megváltoztatja a genotípus struktúráját, viszont a fenotípust nem.

A GliB-et moduláris programok fejlesztésére alkalmazzák, ahol a feladatot (Hanoi tornyai, TTT) primitívek segítségével írják le.

A TTT-nél használt primitívek:

1. *pos00…pos22:* a tábla pozíciói, a bal felső sarokból kezdve a számozást. A többi függvény esetében a visszatérített érték vagy egy pozíció ezek közül vagy NIL (LISP-ben ez hamisat jelent).
2. *and*: két paramétere van. Ha egyik sem NIL akkor visszaadja a másodikat, ha legalább az egyik NIL akkor NIL-t ad vissza.
3. *or:* két paramétere van. Visszatéríti az első nem NIL paramétert, különben NIL-t ad vissza.
4. *if:* if <teszt> then <arg1> else <arg2>.
5. *open:* a paraméterként megadott pozíciót adja vissza ha az adott hely szabad, különben NIL-t.
6. *mine:* a paraméterként megadott pozíciót adja vissza ha ott a játékos jele van, különben NIL-t.
7. *yours:* a paraméterként megadott pozíciót adja vissza ha ott az ellenfél jele van, különben NIL-t.
8. *play-at:* ha a paraméter egy üres pozíció a táblán akkor az aktuális játékos jelét teszi a pozícióra. Különben visszaadja a paraméterként kapott értéket.

A rátermettségi függvény egy genotípus átlagrátermettségét határozza meg egy szakértő játékossal játszott 4 játék alapján. A program minden szabályos lépés után kap 1 pontot. Akkor is kap még 1 pontot, ha ez a lépés megakadályozza a szakértő nyerését. Ha a

játék döntetlen akkor 4 pont, nyerés esetén pedig 12 pont jár. A populáció 100 programot tartalmazott. Az algoritmus 200 generációig futott.

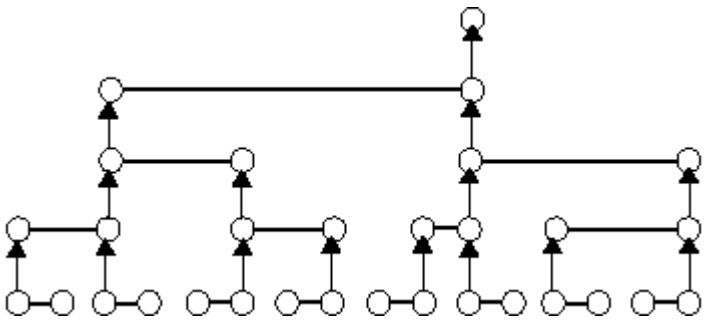
[Angeline93b] és társa az előbbiekben leírt GliB programcsomagot és a TTT primitíveket használják stratégia fejlesztésére, a versengő tanulás (lásd az előző részt) elvét követve.

A standard vagy egyszerű rátermettségi függvényt **független rátermettségi függvény**nek (*independent fitness function*) is lehet nevezni, mivel független a populáció összetevőjétől, egy precíz, biztos és konzisztens értékelést adva az evolúció folyamán. A probléma az, hogy ha egy „szakértő” (*expert*) stratégia is létezne amely lehetővé teszi az egyedek jó értékelését, az erre megépített független rátermettségi függvény csak ezzel a szakértő stratégiával szemben lenne „optimális”. A **versengő rátermettségi függvény** (*competitive fitness function*) olyan kiszámítási módja a rátermettségnek amely valamilyen fokon függ az aktuális populációtól. Az egyedek rátermettségét az aktuális populációhoz viszonyítva értékeli és nem a globális optimumra vonatkoztatva.

[Axelrod87] mind a kettőt használta az IPD stratégia fejlesztésére (lásd a 4.2.3. fejezetet). A független rátermettségi függvény egy súlyozott összpontszám volt 8 előre kiválasztott stratégiával szemben. A versengő rátermettségi függvény esetében a populáció minden egyedét teszteli a populáció összes többi tagjával. Erre a célra a teljes versenyzést (*full competition*) használta. Ha a populáció mérete n akkor egy generációban n^2 darab verseny, összehasonlítás szükséges.

[Hillis92] a rendezőhálók (lásd a 2.10. fejezetet) esetében az állapotokat tartalmazó populációt két részre osztotta és ezek egymás közt versenyeztek egy úgynevezett kétoldali versenyben (*bipartite competition*). Ha az egyedek száma a populációban n akkor összesen $n/2$ darab verseny van egy generáció alatt.

[Angeline93b] bevezeti a **mérkőzés rátermettség** (*tournament fitness*) fogalmát. Ekkor egy kizárásos, eliminációs, bináris mérkőzés (*binary tournament*) zajlik le a relatív rátermettség rangsorolásához. A 4.3. ábra ezt a típusú mérkőzést illusztrálja.



4.3. ábra. Mérkőzés rátermettség: a vízszintes vonalak a versenyt jelentik, a felfele nyilak pedig a győztest.

Eleinte minden egyed részt vesz a mérkőzésben, amit az ábrán a legalsó szint képvisel. Véletlenszerűen kiválasztunk két egyedet, amelyek egymással versenyeznek és a nyertes előrelép a következő szintre. Mikor befejeződött az összes első szintű verseny, a nyerteseket véletlenül összepárosítjuk, hogy meghatározzuk a következő szint nyerteseit. A mérkőzés addig tart, amíg egyetlen nyertes marad. Abban az esetben, ha a versenyzők száma páratlan, akkor egy egyedet áthelyezünk a következő szintre. Egy n elemű populáció esetében az összes versenyek száma:

$$\sum_{i=1}^{\lceil \log n \rceil} \left\lceil \frac{n}{2^i} \right\rceil = n-1,$$

ami eggyel kevesebb mint egy független rátermettségi függvény által igényelt verseny esetében, ahol a populáció összes tagja egy „szakértő” stratégia ellen játszik.

4.3.3. Stratégia fejlesztése együtt-fejlődéssel

[Rosin95], [Rosin97a] és [Rosin97b] bevezetik a **versengő együtt-fejlődés** (*competitive co-evolution*) fogalmát, ahol az egyik populáció egyede versenyben van egy másik populációból származó egyeddel, a rátermettsége pedig e verseny eredményén alapszik. Erre egy jó analógia a szoftverfejlesztésben a komplex applikációk tesztelése, ahol az applikáció megbízhatóságát kimutató tesztsorozat elkészítése majdnem olyan fontos mint maga az a applikáció. A szoftver megoldásait tekinthetjük úgy mint egy populáció, a tesztsorozatot pedig mint egy másik populációt; a versengő együtt-fejlődés lehetővé teszi mind a kettő szimultán keresését.

Minden együtt-fejlődő populáció egyedei fel kell öltsék a tesztelő és tesztelt szerepét. Az egyedeknek szerepnevük van: a **gazda** (*host*) az aminek a rátermettsége éppen kiszámolás alatt van, a **parazita** (*parasite*) pedig az ami teszteli a gazdát, vagyis a rátermettség meghatározására használnak. A gazda sikere maga után vonja a parazita kudarcát. Azért, hogy elkerülje ezt, a parazita fejlődik, újabb kihívásokat teremtve a gazda számára. Ennek a

fejlődésnek a folytonos körforgása a „**fegyverkezési versengés**hez” (*arms race*) vezet (lásd a 2.10. fejezet). Az együtt-fejlődés biztosítja a rátermettség variálását (*fitness variation*) is, ami azt jelenti, hogy a populációk fokozatosan fejlődnek a generációk során: a parazitáknak növekedik a kihívóképességük (immunitásuk), a gazdáknak pedig a kompetenciájuk. Nem fordulhat elő az, hogy olyan parazita alakuljon ki már a legelső generációkban, amit egyetlen gazda sem tud legyőzni.

4.3.3.1. Reprezentáció

Egy kromoszóma egy játékpozíciónak, táblakonfigurációnak felel meg: 9 hosszúságú, hármas alapú számsorozat, ahol a 0 az üres, szabad cellát jelenti, az 1 a gép által lefoglalt cellát, a 2 pedig az ellenfél (pl. játékos) által lefoglalt cellát. Minden pozíciót numerikus sorrendben generálunk és hozzárendeljük egy kromoszómához, ha: van legalább két szabad négyzet (azaz szükség van egy döntésre), nem tartalmaz egy nyerést és nincs egy már létező szimmetrikus (elforgatott és/vagy tükrözött) konfigurációja. Ez összesen 593 darab különböző állapotot jelent.

4.3.3.2. Rátermettségi függvény

Minden GA lelke a rátermettségi függvény. Az együtt-fejlődés nem lenne túl sikeres ha csak egy egyszerű értékelő függvényt (*simple fitness*) használnánk. A következőkben a Rosin és társa által implementált speciális rátermettségi függvényt ismertetjük, amivel már korábban [Angeline93b] is foglalkozott.

4.3.3.2.1. Versengő rátermettségi részesedés

A versengő rátermettségi osztozkodás vagy **versengő rátermettségi részesedés** (*competitive fitness sharing*) a rátermettség mérésére használt újszerű függvény. Az egyszerű fitnesz esetében az egyed rátermettsége egyenlő a versenyekből származó pontszámok összegével. A fitnesz osztozkodás figyelembe veszi az egyedek közötti hasonlóságokat. Az egyed rátermettségét elosztja a vele hasonló egyedek számával, jutalmazva a szokatlan egyedeket. Ennek értelmében az X indexhalmazú parazitákat legyőző gazda rátermettsége:

$$\sum_{j \in X} \frac{1}{N_j},$$

ahol N_j azoknak a gazdáknak az összessége amelyek legyőzik a j indexű parazitát. Ez a függvény azokat az egyedeket részesíti előnybe amelyek olyan parazitákat győznek le, amelyeket kevés gazda győz le, és nem azokat amelyek számszerűleg sok parazitát győznek

le. Ezeknek a szegényesen képviselt gazda egyedeknek a rátermettségük nagyobb lesz, hiszen fontos génanyagot (információt) tartalmazhatnak.

4.3.3.2.2. Osztott mintavétel

A megosztott mintázás vagy **osztott mintavétel** (s*hared sampling*) paraziták kiválasztási módszere a gazdák rátermettségének a tesztelésére. Ahelyett, hogy az összes parazitát felhasználnánk a teszteléskor, csak egy bizonyos számú, változékony halmazú parazitát választunk ki, lehetőleg olyanokat amelyek az előző generációban próbára tették a gazda populáció minden szegmensét és nem kimondottan azokat amelyek a legrátermettebbek voltak. A technika algoritmusa a 4.4. ábrán látható. Megjegyzendő, hogy az ellenfelek azok az egyedek (paraziták), amelyek az előző generációban voltak mintázva, azért, hogy kiszámolják a jelenleg mintázott parazita-populáció (ami az előző generációban a gazda volt) rátermettséget.

```
Aktuális minta üres halmazzal való inicializálása
Minden i ellenfélre az előző generációból
  vesztésnr[i] = 0
Amíg az aktuális minta nincs tele
  Minden j parazitára ami még nincs a mintában
    mintafit[j] = 0
    Minden i ellenfélre amit legyőzött j az utolsó generációban
      mintafit[j] += 1/(1+vesztésnr[i])
  Legyen j olyan, hogy mintafit[j] maximális
  j parazita hozzáadása az aktuális mintához
  Minden i ellenfélre az előző generációból
    Ha j legyőzte I ellenfelet az utolsó generációban
      vesztésnr[i]++
```

4.4. ábra. Az osztott mintavétel algoritmusa.

A `vesztésnr[i]` tömb megmutatja, hogy az `i` parazitát hányan győzték le az előző generációban. A `mintafit[j]` tömb pedig az aktuális mintában levő paraziták rátermettséget tárolja.

A megosztott mintázás még versengő rátermettségi részesedés nélkül is képes jól működni, hiszen ez az algoritmus elsősorban olyan parazitákat választ ki, amelyeket kevés gazda győzött le. Ezért egyszerű fitnesz esetében is azok a gazdák kapnak nagyobb rátermettséget, amelyek olyan parazitákat győznek le, amelyeket kevés gazda győzött le.

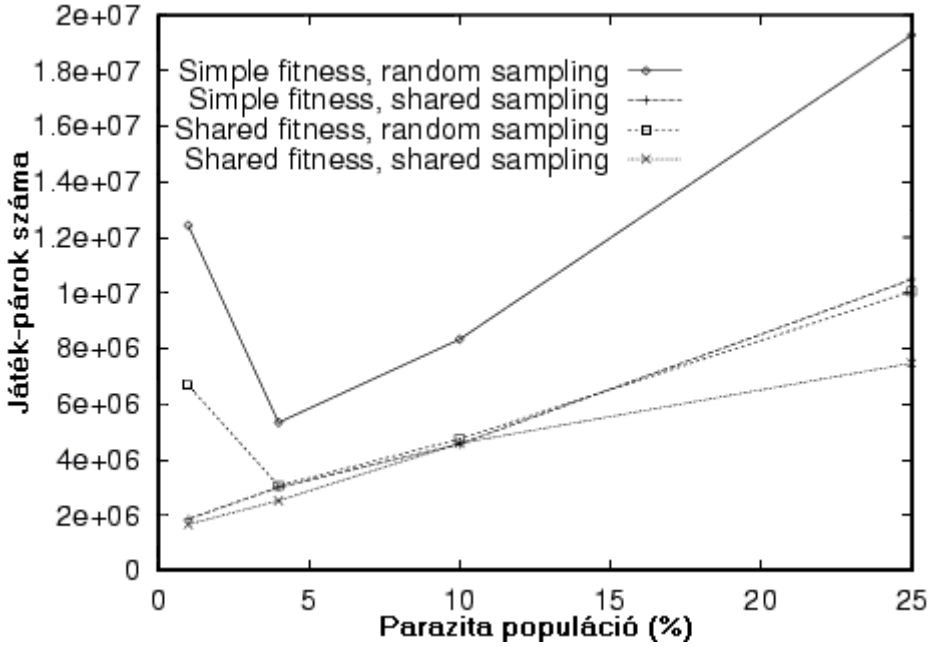
4.3.3.3. Genetikus operátorok

A verseny szelekció jobb volt mint a rulettkerék módszer, az elitizmus pedig biztosította a jó stratégia megőrzését.

Az egypontos keresztezést és mutációt a klasszikus formájukban alkalmazták az egyedekre. Minden utód két szülő (amelyek két független versenyből voltak kiválasztva) rekombinációjából származik. A keresztezési pontok 0,004-es valószínűséggel vannak kiválasztva egy génre. A mutációs valószínűség úgy volt beállítva, hogy egy gén mutációjának valószínűsége 0,0066. Mutációkor egy véletlen, megengedett allél értéket generál a gén számára.

4.3.3.4. Kísérleti eredmények

A GA a két populáción függetlenül működik. Az első populáció egyedei a tökéletes stratégia után kutatnak, a második populáció pedig szembeszáll ezekkel az egyedekkel. Mindegyik populáció 400 egyedet tartalmaz. Versengő rátermettségi részesedés és a megosztott mintázás használatával már 10%-os parazita populációval 1,5 millió játék-pár (3 millió játék) után meg lehet találni a tökéletes stratégiát. A második populáció hiányosságokat fedez fel az elsőben, amelyeket az első populáció kijavít, és ez a folyamat addig tart amíg egy tökéletes megoldáshoz nem jutunk. A 4.5. ábra a 10 különböző futatásból származó átlagteljesítményt mutatja, ahol a rátermettségi függvény (egyszerű vagy osztott) és mintavétel (véletlen vagy megosztott) 4 lehetséges kombinációját használtuk. Érdemes megjegyezni, hogy már az 1%-os (azaz 4 darab) parazita populációt használó teszt is sikeres.



4.5. ábra. TTT teljesítményének a változása különböző rátermettségi függvény és mintavétel esetén.

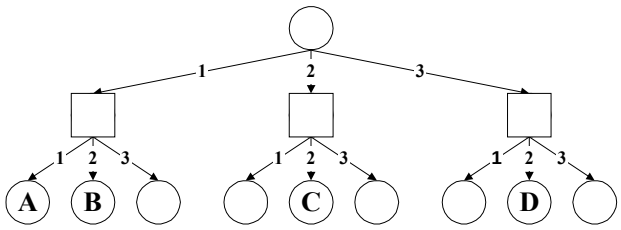
4.3.4. Stratégia fejlesztése genetikus algoritmussal

[Hong98] és társai a genetikus algoritmust a játékfa bejárására alkalmazzák a teljesítmény növelésének érdekében. A bemutatott módszer szimulálja a minimax eljárást azért, hogy meghatározza a legjobb első lépést. Lássuk erre a célra megtervezett genetikus algoritmust.

4.3.4.1. Reprezentáció

Minden GA esetében a megfelelő reprezentáció megadása a legnehezebb feladat és ugyanakkor a legfontosabb is. Kétszemélyes játékfa esetében minden lehetséges lépés a 4.6. ábrán látható módon van kódolva: minden elágazás egy lépéslehetőségnek felel meg. A kör a *MAX* játékos lépéseit jelölik, míg a négyzetek a *MIN* játékosét. Bármely út a gyökértől egy tetszőleges levélig egy lépéssorozatot jelent, amit jelölhetünk úgy mint elágazások sorozata, ahol a két játékos felváltva választ a léplehetőségeik közül. Ezért az *A* levelet kódolhatjuk úgy mint 11, *B*-t mint 12, *C*-t mint 22 és *D*-t mint 32.

A kromoszóma hossza a fa mélységétől függ, így ha a fa mélységét megnöveljük akkor csak a kromoszóma hosszát kell növelni egy újabb számjeggyel a végén.

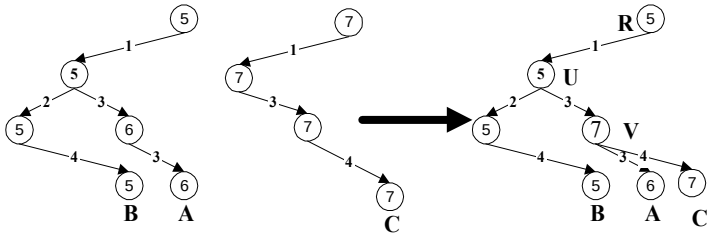


4.6. ábra. Egy játékfa kódoló sémája.

4.3.4.2. Rátermettségi függvény

Olyan rátermettségi függvényt használ, ami a játékfa részleges kiértékelése alapján értékeli az egyedet. Erre a célra bevezették a foglalási fát vagy **rezervációs fá**t (*reservation tree*), ami a rátermettség kiértékelésében segít. A foglalási fa eredetileg egy üres fa, amihez hozzáadjuk a kromoszómák által képviselt részfákat, ami tulajdonképpen egy út a gyökértől a levélig. Az így kialakult fában alkalmazzuk a minimax eljárást (lásd a 3.4. fejezetet): a *MAX* szinten levő csúcsokra a maximális kiértékelést, míg a *MIN* szinten a minimális kiértékelést alkalmazzuk. A 4.7. ábrán az *A*, *B* és *C* kromoszómák rátermettségét számoljuk ki. Az *U* a legkisebb közös őscsomópontja az *A* és *B* csúcsoknak. *U* értéke 5, mivel ez a csúcs egy *MIN*

szinten van és ezért az *R* gyökér értéke is 5. A *C* egyed a *V* csúcsban kapcsolódik a rezervációs fához, és az értéke 7 mivel *MAX* szinten van, az *U* és *R* pedig változatlan marad.

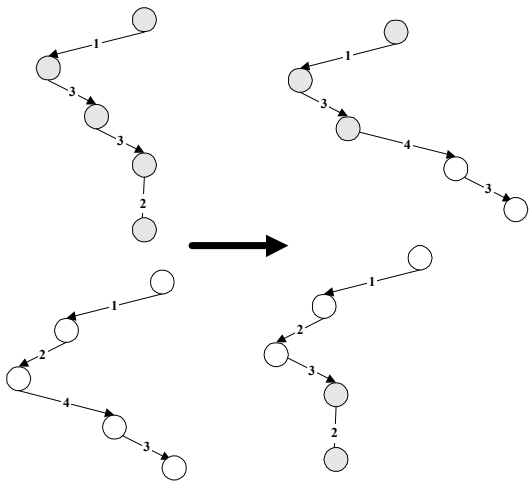


4.7. ábra. A három baloldali egyedet tartalmazó rezervációs fa (a nyíl jobboldalán).

Egy kétszemélyes játékfában az a legjobb lépéssorozat (útvonal) ahol a levél értéke minél magasabb szintre tud terjedni, azaz az út minden csomópontja ugyanazt az értéket tartalmazza mint a levél. Egy kromoszóma (a mi esetünkben egy út) rátermettségét a $H–K+1$ összefüggés adja meg, ahol a H a fa magassága (mélysége) és K az a legmagasabb szint, amire a levél értéke el tud jutni. Ennek értelmében az előző példában szereplő *A* egyed rátermettsége 1, a *B*-é 4 és a *C*-é pedig 2, ami azt jelenti, hogy *B*-nek van a legnagyobb esélye a kiválasztásra.

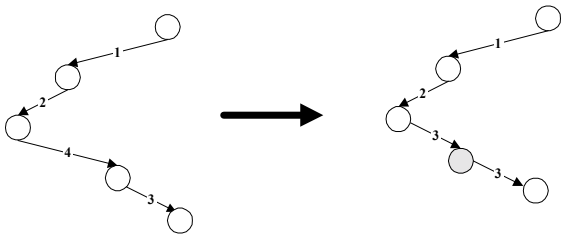
4.3.4.3. Genetikus operátorok

Az egypontos keresztezést használtuk a leszármazottak generálására. Ez azért jó, mert a stringeknél létrejövő adatcsere következtében az egypontos keresztezés megőrizni a lépéssorozatok egy részét. A 4.8. ábra ezt az operátort alkalmazza két lépéssorozatra.



4.8. ábra. Két egyed keresztezése. A keresztezési pont a negyedik szinten van.

A mutáció véletlenül felcserél bizonyos elemeket egy kiválasztott egyedben, ami további genetikai változatossághoz vezet, elkerülve a lokális optimum csapdákat. A 4.9. ábra a mutációt szemlélteti.



4.9. ábra. Egyed mutációja. A mutáció pozíciója a negyedik szinten van.

4.3.4.4. Kísérleti eredmények

Az eredmények azt mutatják, hogy ez a módszer sokkal pontosabb megoldásokat rövidebb idő alatt szolgáltat mint a hagyományos minimax algoritmus. A kísérleteknél egy 7 szintes, 7 elágazást tartalmazó játékfát használtak; a populáció mérete 40 volt.

5. Az alkalmazás dokumentálása. Kísérleti eredmények

Ebben a fejezetben egy olyan számítógépes alkalmazást ismerünk meg, amely teljes egészében saját készítésű. A programcsomag három különböző játék-típus számára fejleszt ki stratégiát, felhasználva az előző fejezetben leirt evolúciós módszereket, nevezetesen a genetikus algoritmusokat és a különböző metaheurisztikákat. A három típusú játék: iterált fogolydilemma, amőba és a 3x3-as kirakós játék két változata. Ezek a játékok nemcsak, hogy különböző típusúak, de eltérő tulajdonságokkal rendelkeznek és ily módon különböző játék-osztályokba tartoznak.

5.1. Az applikáció általános bemutatása. Módszerek ismertetése

Az applikáció Borland Delphi 5-ben íródott. A főablakból lehetőségünk adódik a négy játék kiválasztására. Bármelyiket is választanánk egy új dialógusablak jelenik meg, ahol fel vannak sorolva a rendelkezésre álló kereső eljárások, az algoritmusok paramétereit szabályozni lehet a megfelelő szerkesztő mezők segítségével és számos kimeneti eredményt megjelenítő doboz: kromoszóma konfiguráció, legjobb egyed rátermettsége, átlag pontszáma, hibája illetve a populáció átlag rátermettsége, pontszáma, hibája.

A *Search* nyomógomb segítségével végre lehet hajtani egyszeri keresést, a *Runs* gomb segítségével pedig több keresést lehet automatikusan elindítani. Előnye, hogy a kilistázott eredmények a végrehajtott próbálkozásokból származó eredmények (a legjobb egyed rátermettsége és a populáció átlag rátermettsége) átlagát adják meg.

A program fontos része a genetikus algoritmus alkotóelemeinek a reprezentálása. A populációnak egy olyan tömb fel meg, amelynek minden eleme egy újabb tömb, ami nem más mint az egyedek, kromoszómák az adott populációban. A kromoszómákban levő gének allél értékeit egész típusú vektorokban tároljuk. A másik nagyon fontos komponense a genetikus algoritmusnak a rátermettségi függvény. Ezért a populáció egyedeinek rátermettségét egy újabb tömbben tároljuk.

A felhasználó a rádiógombok segítségével kiválaszthatja, hogy milyen kereső módszer segítségével próbálja meg a program az adott játék számára az optimális stratégiát megkeresni. Általában négy különböző módszer áll rendelkezésre, de például az amőba

esetében csak a genetikus algoritmus volt implementálva. A megírt algoritmusok csak kis mértékben térnek el egymástól a különböző játékok esetében. Általában a reprezentáció más és ebből kifolyólag az egyedek rátermettségének a meghatározása is más eljárást igényel.

A metaheurisztikus kereső eljárások – értem ezalatt a hegymászó algoritmust, a tabukeresést és a szimulált hűtést – a 2.8. fejezetben bemutatott algoritmusok alapján voltak implementálva. A programban lehetőség van az algoritmusok paramétereinek az állítására: a szomszédok (a programban a *Neighbors* címkével jelölve) száma megadja, hogy az aktuális állapotnak mekkora környezetében keressük a következő legjobb állapotot, a hegymászó algoritmus esetében állítani lehet az iterációk számát (*Iterations*), a szimulált hűtés módszernél szintén az iterációk számát (*Iterations*) és ezen felül a kezdeti hőmérsékletet (*Temp*), és végül a tabukeresésnél az iterációk számát (szintén az *Iterations* címke jelöli) és a tabulista nagyságát (a programban a *Tabulist* mező).

Az optimalizáláshoz felhasznált genetikus algoritmus a 2.5. és 2.6. fejezetekben leírtak alapján volt kivitelezve. Az algoritmus folyamán előforduló paraméterek értékeit az applikációból állítani lehet, lehetővé téve a hatékonyabb keresést és gyorsabb konvergenciát. Inicializáló lépésként a kezdeti populációt feltöltjük véletlenszerű egyedekkel, persze a lehetséges értékek az adott feladat (a mi esetünkben játékok) kromoszóma reprezentációjától függnek. Az iteráció első lépésében kiválasztjuk a szelekció segítségével a legrátermettebb egyedeket és ezek töltik be a szülők szerepét. Az iterációk vagy generációk számát a programból az *Iterations* mezőből lehet szabályozni, amit pedig a szelekció illet, a program két különböző szelekciós operátort (lásd a 2.5.1. szakaszt) kínál fel: verseny szelekció (az applikációban a *Tournament* rádiógomb) és rulettkerék módszer segítségével megoldott rátermettség-arányos szelekció (*Roulette*). Jellemző az, hogy mindig ugyanannyi egyedet választunk ki mint amekkora a populáció mérete. Ezért a rátermettebb egyedek általában több példányban képviseltetik magukat, míg a kevésbé rátermettebbek lehet, hogy nem lesznek kiválasztva. Persze mindez nagyobb mértékben a rátermettség értékétől és kisebb mértékben a véletlentől, „szerencsétől” függ.

Az egyedek kiválasztása után alkalmazhatjuk a keresztezést (2.5.3. fejezet) és a mutációt (2.5.2. fejezet) a megfelelő valószínűségekkel, amit a programban a *Crossover* illetve a *Mutation* mezők értéke ad meg. A rekombinációnál az egypontos keresztezést alkalmaztuk, két leszármazott jött létre, a szülők helyét pedig elfoglalják a leszármazottak. A kereszteződési/mutációs arányokat (ha százalékban adjuk meg) vagy valószínűségeket (a százalékban megadott értéket egy 0 és 1 közötti számmá alakítjuk) a populáció egészére alkalmazzuk, ezért előfordulhat az, hogy a keresztezésből származó utódok is mutálódnak.

Ha az *Elitist* jelölődoboz ki van választva, akkor a szelekció elitista lesz, vagyis az előző generáció legjobb egyede elfoglalja a frissen létrejött populáció legrosszabb egyedének a helyét. Ez biztosítja azt, hogy ha az algoritmus megtalált egy jó stratégiát az őrződjön meg a generációk során. Ezzel a művelettel lezárult egy teljes iteráció, az új populáció a következő generációban a szülők szerepét töltik be, addig ismételve ezt a műveletsorozatot amíg az iterációs szám elér egy adott korlátot.

Megjegyzendő, hogy ha a kereső eljárások közül a genetikus algoritmus rádiógombja van kiválasztva, akkor aktívvá válik a *Write to File* jelölőgomb, amit bejelölve az applikáció kilistázza egy állományba a generációk során kifejlesztett egyedek konfigurációját és rátermettségét. A *Seconds* mező az egyszeri keresés (a *Search*-el elindított) során eltelt másodpercek számát adja meg.

A továbbiakban játékokként mutatjuk be az alkalmazásnak azon részeit amelyek eltérnek egymástól, kitérve a stratégiák által elért eredményekre is.

5.2. Iterált fogolydilemma

Az iterált fogolydilemma kétszemélyes, nem teljes információjú, nem zérusösszegű stratégiai játék. A játék részletes megismerése érdekében az olvasónak figyelmébe ajánljuk a 4.2. fejezetet, ahol kitértünk a játék szabályaira, ismertettük a lehetséges stratégiákat és az optimális stratégiák fejlesztésére felhasznált evolúciós módszereket.

Az implementált metaheurisztikus eljárások és a felhasznált genetikus algoritmus az előző fejezetben bemutatott elveken alapszik és ugyanazokat a paramétereket lehet állítani a program felületén. A kromoszóma reprezentáció megegyezik azzal amit a 4.2.3.1. szakaszban ismertünk meg, azaz a kromoszóma tárolja az előző 6 lépés kimeneti eredményeit (0 vagy 1, ami az elpártolásnak illetve az együttműködésnek felel meg), ami 64 darab stratégiát jelent. A játék megkezdéséhez újabb 6 darab gén szükséges, ami összesen egy 70 hosszúságú kromoszómát eredményez. Az egyedek rátermettségének a kiszámolásához a versengő rátermettségi függvényt (lásd a 4.3.2. fejezetet) használtuk, amikor is minden egyed bizonyos számú játékot játszott le a populáció összes tagjával. A játék-párok (egy-egy lépés, döntés mindkét fél részéről) számát a *Gamenr* mező segítségével szabályozható. Ha a *Self-Play* jelölődoboz ki van választva akkor az egyedek önmagukkal is játszanak. Ily módon egy egyed (stratégia) által elérhető rátermettségi érték egyenlő lesz az összes játék alatt (játékok száma szorozva a populáció méretéve, ha van önjátszás) az adott egyed által kapott pontszámok (a jutalommátrix alapján) összegével.

Ami az általános szolgáltatásokkal szemben pluszt jelent, az az, hogy a GA-val kifejlesztett stratégiát és egyéb 8 stratégiát (amelyeket a 4.2.2. részben ismertettünk) össze lehet hasonlítani a véletlen, azaz a RANDOM stratégiával. Ezt a felhasználó megteheti, ha kiválasztja a *Strategies vs RANDOM* rádiógombot, majd ezután a megfelelő stratégia rádiógombját és végül a *Test* nyomógombot. Mi több, lehetőség van interaktív módon bármely stratégiákkal játszani. Az ehhez szükséges gombok és kijelző dobozok a *Play* keretben találhatóak, ami csak akkor aktív, ha a fent említett *Strategies vs RANDOM* rádiógomb előzőleg ki volt választva.

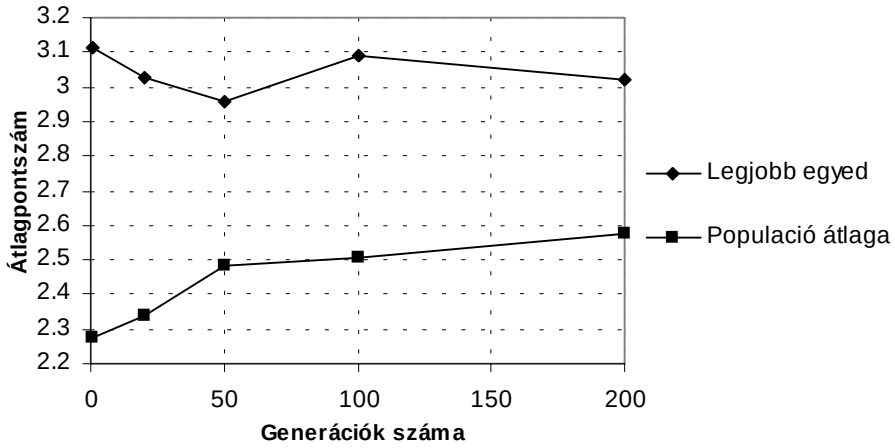
Továbbá érdemes megjegyezni, hogy a program megmondja, hogy milyen viselkedési minták (az 5 minta a 4.2.3.4. részben volt értelmezve) fordulnak elő a legjobb kromoszómában és ezeket a *Behavioral Patterns* név alatt listázza.

A következőkben a genetikus algoritmusra koncentrálva, grafikonokon keresztül vázoljuk a kapott eredményeket. A paramétereket (iteráció szám, populáció mérete, keresztezési arány) széles határok között változtattuk és a programmal több tesztet végeztünk. Az rulettkerék szelekció jobbnak bizonyult mint a verseny szelekció, ugyanis a verseny szelekció a nagyon rátermett egyedeket részesítette előnybe, ami a fejlődési szakasz első felében a változatosság csökkenéséhez vezetet (az elitizmus is ezt idézte elő). Habár az elért pontszámok nagyobbak voltak mint a rulettkerék esetében, a RANDOM stratégiával szemben mégsem teljesítettek olyan jól. Az ábrák, az algoritmusnak 10 egymásutáni végrehajtásából származó átlagpontszámok számtani közepét jelenítik meg. Megjegyzendő, hogy egy egyed által elérhető maximális átlagpontszám 5, abban az esetben, ha az ellenfele végig kooperált, míg a populáció maximális átlagpontszáma 3, ami akkor következik be ha mindkét fél kölcsönösen együttműködött. Ennek értelmében egy egyed (stratégia) által elért átlagpontszám a következőképpen számítható ki:

```
ha van önjátszás akkor
  átlagpontszám= összpontszám/populáció méret/játék-párok száma
különben
  átlagpontszám= összpontszám/(populáció méret-1)/játék-párok száma
```

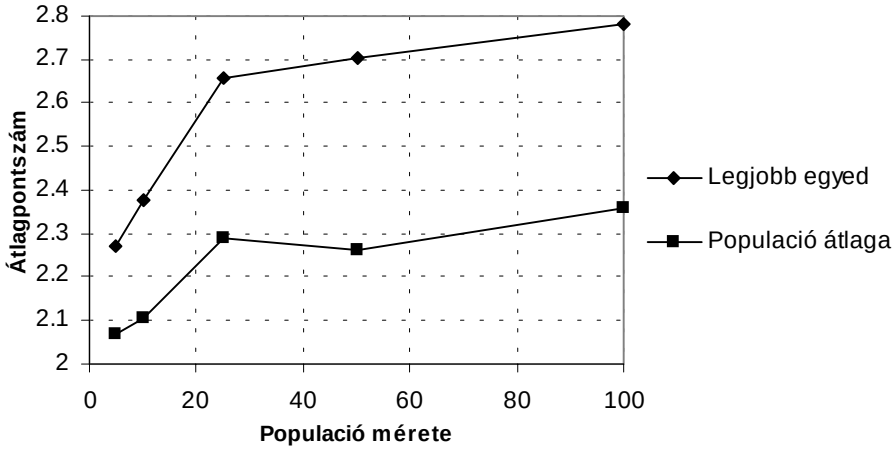
Minden egyes algoritmusnál a legfontosabb paraméter az iteráció szám, ugyanis ez mutatja meg, hogy az adott algoritmus mennyire hatékony, mennyire gyorsan konvergál. A GA viszonylag kevés iterációval jó eredményeket ért el, amit az 5.1. ábra szemléltet. A grafikonon megjelenik a legjobb egyed átlagpontszáma és a populáció átlaga is. Az itt elért eredmények beigazolják a 4.2.3.4. részben bemutatott szimulációs eredményeket. A generációk első felében az elpártoló magatartás a jellemző, de az iterációk során mind több és

több egyed kezd el kooperál, így az átlagpontszám a 3-as felé konvergál, ami azt jelenti, hogy az interakciók során mindkét fél együttműködött.



5.1. ábra. A GA-val elért eredmények. A populáció mérete 50, a keresztezési valószínűség 0,25, a mutációs valószínűség 0,01, a játék-párok száma 150, rulettkerék szelekció.

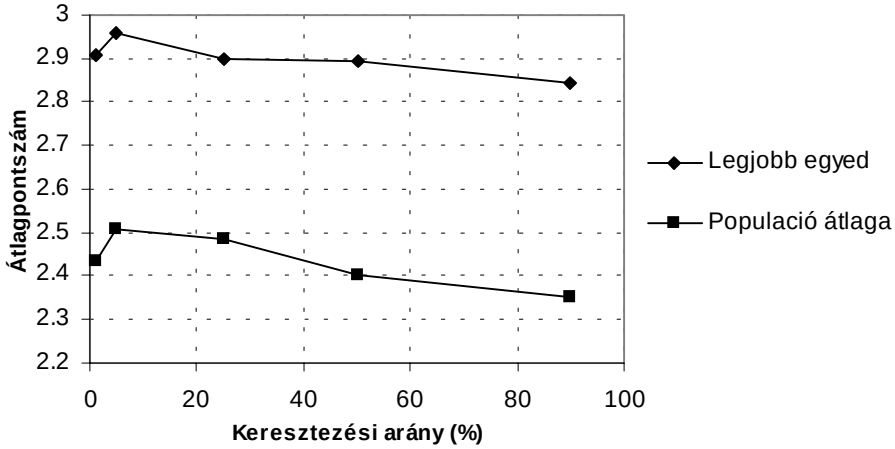
A genetikus algoritmusnál a legfontosabb paraméter a generáció-szám mellett a populáció mérete. A nagyobb populáció genetikai változatosságot biztosít, fontos génanyag fejlődik ki, amely pont az optimális megoldást tartalmazza. Az IPD esetében ez még fontosabb, ugyanis az optimális stratégia így még több egyedet kell legyőzzön, alkalmazkodva ily módon egy erős és változatos környezethez. Az átlagpontszám nagy mértékben növelhető a populáció számának növelésével, viszont nagyon erőforrás-igényes. A kiértékelés az 5.2. ábrán látható.



5.2. ábra. Az elért pontszám a populáció méretének függvényében. Az iteráció szám 50, a keresztezési valószínűség 0,25, a mutációs valószínűség 0,01, a játék-párok száma 150, rulettkerék 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 az 5.3. grafikon ábrázol. Az 5%-os keresztezési arány szolgáltatta a legjobb eredményt. Ennél nagyobb számú keresztezés negatív hatással volt a populáció egyedeire, ezzel alátámasztva azt a megállapítást, hogy bináris ábrázolás esetén kisebb keresztezési valószínűség ajánlott.



5.3. ábra. A keresztezés hatása. Az iteráció szám 100, a populáció mérete 50, a mutációs valószínűség 0,01, a játék-párok száma 150, rulettkerék szelekció.

5.3. Kirakós játék

Ebben az alfejezetben a 3x3-as kirakós játék két változatát ismerjük meg. E játék esetében egy 3x3-as hálón 9 darab számozott négyzet alakú lapocska van (5.4. ábra). A játék lényege, hogy kiindulva egy adott kezdőállapotból el kell jussunk egy célállapothoz, úgy, hogy közben a négyzeteket adott szabály szerint cserélgetjük egymás között. Az első változatban (9-es játéknak szokták nevezni) minden négyzet felcserélhető olyan négyzetekkel ami ugyanabban a sorban vagy oszlopban van és nem feltétlenül egymás mellettiek. A második változatú kirakós játékot még szokták 8-as játéknak is nevezni [Fekete99], ugyanis itt az egyik négyzet hiányzik, így csak 8 lapocska marad. Ebben az esetben csak azok a négyzetek mozdíthatók el amelyek mellett ott van az üres hely, a lapocska helye pedig a csere következtében üresen marad.

8	**4**	**7**
2	**1**	**9**
6	**5**	**3**

$\rightarrow$

1	**2**	**3**
4	**5**	**6**
7	**8**	**9**

5.4. ábra. Példa a 9-es játékra. A baloldalon egy lehetséges kezdőállapot a jobboldalon pedig a végállapot van.

A kirakós játékok az egyszemélyes, teljes információjú játékok osztályába tartoznak. A 4.1. fejezetben már említettük, hogy [Hong02] és társai olyan genetikus algoritmust javasoltak, amit sikerrel alkalmaztak a 9-es játékra. Az elkészített GA a 4.3.4. szakaszban bemutatott módszeren alapszik, hiszen ott a kétszemélyes játékfára alkalmaztuk az optimalizálást, itt pedig ugyanezt tesszük az egyszemélyes játékfa esetében.

5.3.1. Reprezentáció

A két játéknak megfelelő genetikus algoritmus csak a reprezentációban és rátermettségi függvényben tér el, a többi komponens ugyanaz. A 9-es játék esetében úgy jutunk át az egyik állapotból a másikba, hogy szabályosan felcserélünk két négyzetet. Ha a két lapocska számát egy kóddal helyettesítjük, akkor a gén allél értéke egy ilyen kód lenne, a kromoszóma pedig a kódok sorozatából állna, ami áttranszformálja a kezdőállapotot egy másik állapotba, ami megoldás esetén a végállapot kell legyen. Az állapotok közötti átmenetek számát a *Depth* mező értéke adja meg, ami tulajdonképpen a játékfa mélységének felel meg, és mivel a kromoszómával a játékfát reprezentáljuk, így a kromoszóma hosszát is jelenti. Az 5.1. táblázat a lehetséges cseréket kódolja, amihez hozzávesszük a 0-s kódot, ami azt jelenti, hogy nincs csere.

5.1. táblázat. A felcserélendő cellák kódolása a 9-es játék esetében.

1. cella	1	2	3	4	5	6	7	8	9	1	4	7	2	5	8	3	6	9
2. cella	2	3	1	5	6	4	8	9	7	4	7	1	5	8	2	6	9	3
Kód	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

A 8-as játéknál nincs szükségünk kódokra, ugyanis a mozgatásokat az üres cellára vonatkoztatjuk. Minden lépésben csak azt mondjuk meg, hogy milyen irányba mozgassuk az üres lapocskát és ezeknek feleltetünk meg numerikus kódokat: fel (a kód az 1-es), balra (2-es kód), le (3) vagy jobbra (4). Persze ide is bevesszük a 0-ást, ami azt jelenti, hogy nincs semmilyen mozgás. Így a kromoszóma 0 és 4 közötti számokból áll.

5.3.2. Rátermettségi függvény

A játék akkor tekinthető befejezetnek, ha minden lapocska a helyén van, azaz az aktuális állapot egybeesik a végállapottal. Ez azt jelenti, hogy minél nagyobb az eltérés az aktuális állapot és a végállapot között, azaz minél kevesebb négyzet van a helyén annál rosszabb az adott állapot, tehát a rátermettsége is annál kisebb kell legyen. A rátermettségi függvényt a következő összefüggés adja meg:

$$25 - \sum_{i=1}^{9} d(A_i, V_i),$$

ahol A_i az *i* szám aktuális pozíciója, V_i pedig a végállapotban a pozíciója, *d* pedig a távolság a két pozíció között. Például az 5.4. ábrán az össztávolság a két állapot között 20. Tudjuk, hogy a rátermettség és az összeltérés fordítottan arányos valamint azt, hogy a legrosszabb esetben az össztávolságok értéke 24, és, hogy ne legyen 0 értékű rátermettség (a rulettkerék szelekciónál a 0 értékű populáció-rátermettség problémát jelent) ezt az értéket kivonjuk 25-ből.

A 8-as játéknál a rátermettségi függvény csak annyiban változik, hogy az össztávolságokat nem 25-ből, hanem 26-ból vonjuk ki, mivel előfordulhat az is, hogy az üres négyzet olyan helyre kerül ahonnan nem lehet a megadott irányba lépni. Ebben az esetben a rátermettség értéke a legrosszabb, vagyis az 1-es értéket veszi fel, míg ha sikeresen végre lehetett hajtani a átmeneteket az állapotok között (amit a kromoszóma allél értékei adnak meg) akkor az adott egyed nagyobb rátermettséggel (legrosszabb esetben 2-vel) fog rendelkezni.

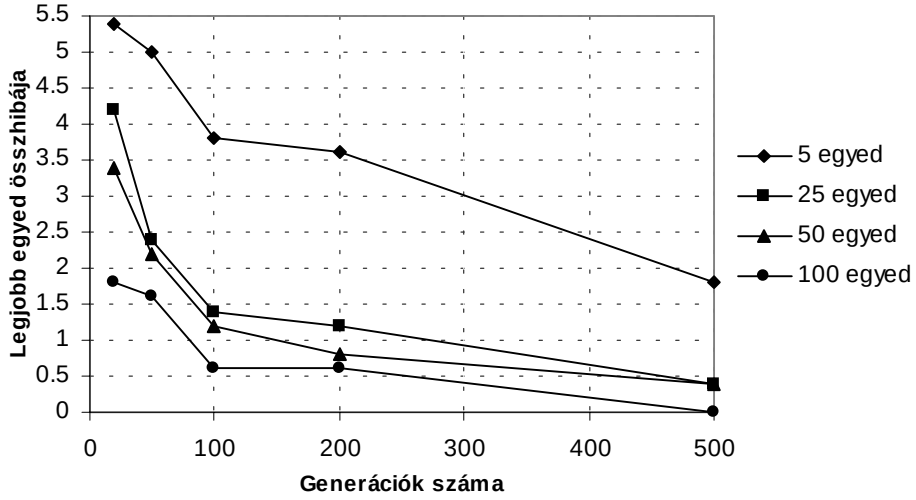
5.3.3. Dokumentálás

A megírt metaheurisztikus eljárások és a felhasznált genetikus algoritmus megegyezik azzal amit az IPD számára készítettünk és ugyanazok a paraméterek szabályozhatók a dialógusablak felületéről. Ezeken felül implementáltunk egy sajátos genetikus algoritmust (amit a *Genetic Algorithm2* rádiógomb jelöl), amely elitista verseny szelekciót használ. A módszer a következőképpen néz ki. A kezdeti véletlen populáció generálása után, az iteráció első lépéseként hajtsuk végre a keresztezést a keresztezési valószínűségeknek megfelelően, a leszármazottakat pedig adjuk hozzá a szülő populációhoz. Az így létrejött populációra alkalmazzuk a mutációt, majd a mutált egyedeket is adjuk hozzá a populációhoz. Így az eredeti populáció kibővült a gyerek-egyedekkel és a mutáns-egyedekkel. A szelekciót erre a populációra alkalmazzuk és kiválasztjuk a legjobb *n* egyedet, ahol *n* a paraméterként megadott populáció méretét jelenti. Az ily módon kiválasztott egyedek a szülők szerepét töltik be a következő generációban.

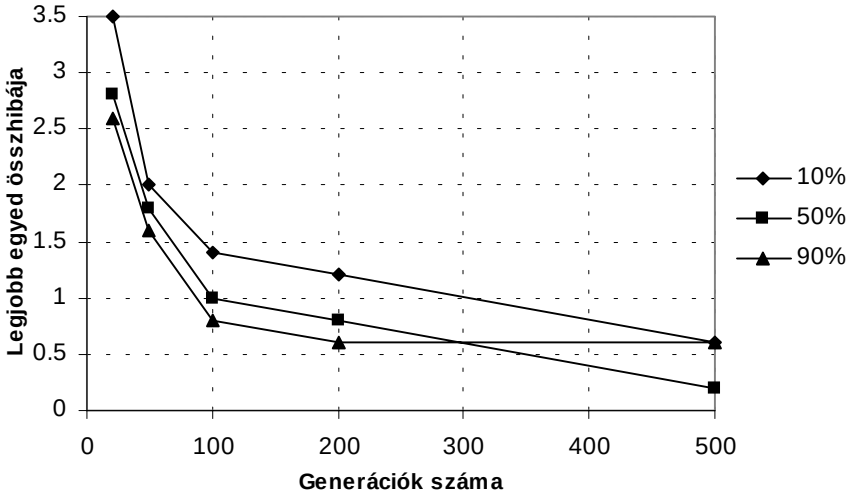
A *New Puzzle* nyomógomb segítségével a felhasználó új kezdőállapotot generálhat, de azt is megteheti, hogy a kezdeti állapotot beírja a megfelelő szerkesztő dobozba.

5.3.4. Eredmények kiértékelése

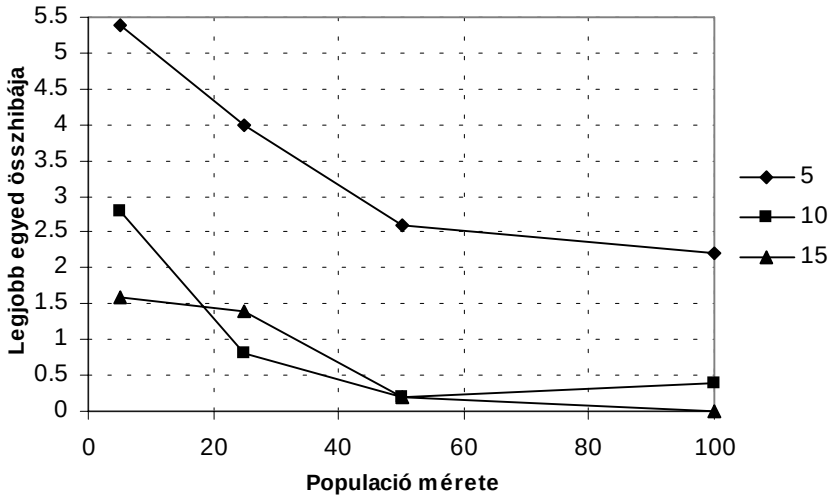
A 9-es játék eredményeinek a kiértékeléséhez ugyanazt a módszert alkalmaztuk mint az iterált fogolydilemmánál, azaz grafikonokat készítettünk. Itt nem ábrázoltuk a populáció átlagát, viszont a grafikonok két paraméter egyidejű hatását mutatják be. Ennél a játéknál az elitista verseny szelekcióval a GA gyorsabban konvergált és az eredmények is pontosabbak voltak. Következzenek az ábrák, amelyek eredményei szintén 10 próbálkozás átlagából származnak. Az összhiba az összes olyan lapocskának a távolsága a végső állapottól, amelyek nincsenek a helyükön (lásd a fenti képletet).



5.5. ábra. Az eltérő populációs méretek (5, 25, 50 és 100) eredményei a generációk függvényében. A keresztezési valószínűség 0,5, a mutációs valószínűség 0,02, a játékfa mélysége 15, elitista verseny szelekció.



5.6. ábra. A különböző keresztezési arányok (10%, 50% és 90%) eredményei a generációk függvényében. A populáció mérete 50, a mutációs valószínűség 0,02, a játékfa mélysége 15, elitista verseny szelekció.



5.7. ábra. A játékfa mélységének (5, 10 és 15) és a populáció méretének az egyidejű változtatása. A generációk száma 500, a keresztezési valószínűség 0,5, a mutációs valószínűség 0,02, elitista verseny szelekció.

5.4. Amőba

Az amőba kétszemélyes, teljes információjú, zérusösszegű, hirtelen halálú stratégiai játék. Ebben a részben az amőba egy egyszerűbb változatával foglalkozunk, a Tic-Tac-Toe-val, amelynek a szabályait a 3.2.1. szakaszban ismertettük. A 4.3. fejezetben részletesen kitértünk olyan evolúciós módszerekre, amelyek e játék számára próbáltak meg optimális stratégiákat fejleszteni.

A TTT esetében csak a genetikus algoritmust használtuk fel stratégia fejlesztésére. Mivel ez a játék az előzőknél sokkal komplexebb, így egy jóval hatékonyabb módszert igényel, amelynek a metaheurisztikus eljárások nem feleltek meg. A program az iterált fogolydilemma analógiájára készült, a felhasznált GA is ugyanazon az elveken alapszik és a paraméterek nagy része is megegyezik. A 4.3.3.1. részben leirt kromoszóma reprezentációt használtuk fel, azaz a kromoszóma 593 darab génből áll, az allél érték pedig megmutatja, hogy az adott állapotban (ami a mi esetünkben tábla konfigurációja), az aktuális játékos melyik üres pozícióra lép. Tehát a gének 1 és 9 közötti értéket vehetnek fel, ahol ez a szám az elfoglalt pozíciót jelentik a táblán (a 3x3-as négyzetrácsot egy 9 hosszúságú tömbként ábrázoljuk).

Az 593 darab táblakonfiguráció olyan állapotokat képviselnek, amelyek egymástól eltérnek, nem szimmetrikusak (függőlegesen tükrözött és/vagy elforgatott), nincs bennük nyerés, van legalább két üres pozíció és nem tartalmazza a kezdőállapotot. Persze ezek az állapotok csak akkor érvényesek, ha a kezdőjátékos kódja az 1-es, azaz a számítógép kezd.

Ha azt akarjuk, hogy működjön akkor is amikor az ellenfél (2-es kódszámú) kezd, akkor a kódokat fel kell cserélni: egyesből kettes lesz és kettesből pedig egyes.

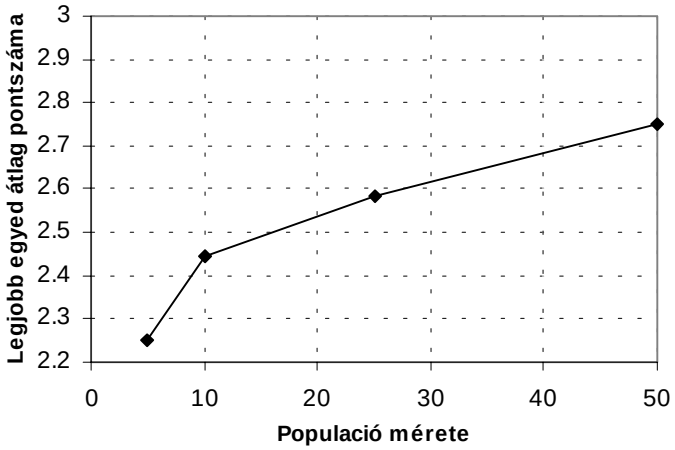
Akárcsak az IPD esetében (lásd az 5.2. szakaszt), egy egyed rátermettségének a kiszámolásához itt is bizonyos számú játékot (a *Game-Pairs* mezőből állítható) játsztattunk le a populáció többi tagjaival (lehet önjátszás is). Egy játék-pár alkalmával a két játékos felváltva kezd, egyszer az egyik lép elsőnek, majd a következő játékban az ellenfele kezd. A játékok elején a tábla celláit 0-val inicializáltuk. A kezdőjátékos a középre lép, majd felváltva lépnek oda ahova a megfelelő allél érték mutat. Ezt úgy valósítjuk meg, hogy az aktuális állapotról megállapítjuk, hogy az 593 különböző állapotot tartalmazó listában hányadik pozíción van. Ezen az indexen található gén allél értéke lesz az aktuális lépés. Minden egyes játék végén az eredmények alapján (nyerés, döntetlen vagy vesztés) jutalmazzuk a stratégiákat, a pontszámokat pedig a programból a *Win*, *Draw* és *Lose* mezők segítségével állíthatók. Az egyed rátermettsége egyenlő lesz a játékok alatt élért pontszámok összegével (kumulált pontszám). Ezt figyelembe véve, egy egyed átlagpontszáma a következő módon adható meg:

```
ha van önjátszás akkor
  átlagpontszám= összpontszám/populáció méret/játék-párok száma/2
különben
  átlagpontszám= összpontszám/(populáció méret-1)/játék-párok száma/2
```

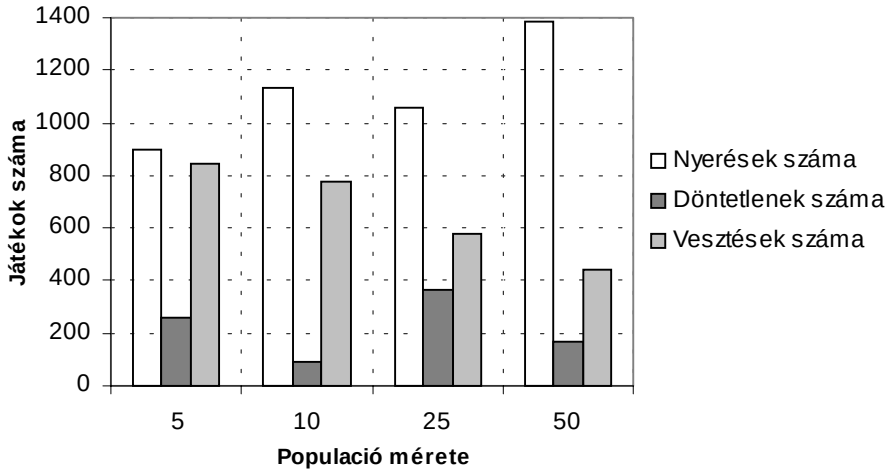
Ha a GA-val megkerestünk egy stratégiát, akkor a *Strategy vs RANDOM* nyomógomb aktívvá válik, majd ezt lenyomva letesztelhetjük a stratégia hatékonyságát a véletlen stratégiával szemben. Az alkalmazás felkínálja azt a lehetőséget is, hogy a felhasználó játszódhasson a frissen előállított stratégia ellen, beállítva a *Computer moves First* jelölődoboz segítségével a kezdőjátékost (ha a doboz ki van jelölve akkor a gép lép elsőnek).

Az eredmények aránylag szerény számú próbálkozásokból származnak, ugyanis a feladat komplexitásából származó erőforrás-igény nagyon nagy. A TTT esetében a verseny szelekció hatékonyabbnak bizonyult mint a rulettkerék módszer. Az elitizmus nem sokat javított és mivel az algoritmuson sokat lassít, így nem is használtuk. A grafikonokon az egyszeri keresés eredményei jelennek meg. Ennél a játéknál a populáció mérete talán fontosabb paraméter mint a generáció-szám. A nagyobb populációval egy iteráció alatt a stratégiák több játékot játszanak le, ami által az egyedek tanulnak. Az 5.8. ábra ezt szemlélteti. Megjegyezzük, hogy az elérhető maximális összpontszám ebben az esetben 3, mert egy egyed 3 pontot kap, ha egy játékot megnyert, 2-öt ha döntetlent játszott és 1-et, ha veszített. Az így megkapott stratégiát tesztnek vetettük alá, ahol 2000 darab játékot játszott le

(ahol felváltva lépett elsőnek illetve másodiknak) a RANDOM, azaz a véletlen stratégiával. Az eredmények (5.9. ábra) azt mutatják, hogy az adott egyed az evolúciós folyamatok során képes volt bizonyos lépések betanulására és nagyon sok esetben nyert, vagy döntetlent játszott. A populációk számának növelésével tovább növelhető a stratégia hatékonysága (lásd a 4.3.3.4. részt).



5.8. ábra. Az átlagpontszámok változása a populáció méretének függvényében. Az iteráció szám 100, a keresztezési valószínűség 0,05, a mutációs valószínűség 0,08, a játék-párok száma 2, verseny szelekció.



5.9. ábra. Az 5.8. ábrán levő egyed által elért eredmények a RANDOM stratégiával szemben. A lejátszott játék-párok száma 1000.

6. Következtetések és javaslatok

A genetikus algoritmusok hatékony globális optimalizálóknak bizonyultak. Mint sztochasztikus algoritmusok jól alkalmazhatók NP-teljes feladatokra és nagy keresési térrel rendelkező problémákra (függvényoptimalizálás, gépi tanulás, mesterséges neuronhálók fejlesztése, kombinatorikus problémák stb.). Robusztus struktúrájukból kifolyólag képesek olyan nehéz feladatokat is megoldani, ahol nem rendelkezünk terület-specifikus leírással. Mivel a GA probléma-független, így alkalmazható játékelméleti problémákra is, ahol nagy a keresési tér és nincs konkrét terület-specifikus ismeret.

A dolgozat során számos olyan módszert, eljárást mutatunk be, ahol evolúciós technikákat alkalmaztunk játékok stratégiáinak a fejlesztésére. A kísérletekre használt játékok eltérő tulajdonságúak, különböző játék-osztályokat képviselnek. A GA képes volt viszonylag rövid idő alatt intelligens magatartásformákat fejleszteni az iterált fogolydilemma számára. A GA által megkapott stratégia jobb eredményeket ért el mint bármely más módszer. Ez annál is fontosabb, mert az IPD számos társadalmi interakció szimulálására használt játék. A kirakós játékra nem létezik egy kimondott megoldási stratégia, így a GA ebben az esetben is bebizonyította hasznosságát. A TTT bonyolultabb játékot képvisel, így ebben az esetben a GA erőforrás-igényessége negatívan hatot. A jó stratégia megkeresése sok tanulást (játékot) igényel, ami lassú lehet, még speciális technikák használatával is.

További kutatás tevékenységét képviselheti egy olyan alkalmazás implementálása, amelyben a TTT számára fejlesztünk ki stratégiát az együtt-fejlődés segítségével, illetve a neuronhálók felhasználásával. Mivel ezek a módszerek elsősorban egyszerű játékokra voltak kifejlesztve, így bonyolultabb játékok (pl. a Go-Moku) esetében a bemutatott genetikus minimax eljárás ajánlott.

Szakirodalom

[Alander00a] Jarmo T. Alander: An Indexed Bibliography of Genetic Algorithms and Simulated Annealing: Hybrids and Comparisons, Report 94-1-SA, Department of Information Technology and Production Economics, University of Vaasa, Finland, 2000.

[Alander00b] Jarmo T. Alander: An Indexed Bibliography of Genetic Algorithms and the Traveling Salesman Problem, Report 94-1-TSP, Department of Information Technology and Production Economics, University of Vaasa, Finland, 2000.

[Alander01] Jarmo T. Alander: An Indexed Bibliography of Genetic Algorithms and Neural Networks, Report 94-1-NN, Department of Information Technology and Production Economics, University of Vaasa, Finland, 2001.

[Allis94] Louis Victor Allis: Searching for Solutions in Games and Artificial Intelligence, Ph.D. Thesis, Rijksuniversiteit Limburg te Maastricht, Maastricht, The Netherlands, 1994.

[Angeline92] Peter J. Angeline, Jordan B. Pollack: The Evolutionary induction of subroutines, in Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society (Bloomington, Indiana), Lawrence Erlbaum, Hillsdale, NJ, pp. 236-241, 1992.

[Angeline93a] Peter J. Angeline, Jordan B. Pollack: Learning with a Competitive Population, LAIR Tech Report # 92-pa-compete, Submitted to IJCAI ’93, 1993.

[Angeline93b] Peter J. Angeline, Jordan B. Pollack: Competitive environments evolve better solutions for complex tasks, in Stephanie Forrest (Ed.), Proceedings of the 5th International Conference on Genetic Algorithms (Urbana-Champaign, IL, 1993), Morgan Kaufmann, pp. 264-270, 1993.

[Angeline94] Peter J. Angeline, Jordan B. Pollack: Coevolving high-level representations. in Christopher G. Langton (Ed.), Artificial Life III: Proceedings of the 3rd Artificial Life Meeting, Addison-Wesley, Reading, MA, pp. 55-71, 1994.

[Axelrod84] Robert Axelrod: The Evolution of Cooperation, Basic Books, New York, 1984.

[Axelrod87] Robert Axelrod: The Evolution of Strategies in the Iterated Prisoner's Dilemma, in Lawrence Davis (Ed.), Genetic Algorithms and Simulated Annealing, Pitman, London/Morgan Kaufmann, Los Altos, CA, pp. 32-41, 1987.

[Beaufils97] B. Beaufils, J. Delahaye, P. Mathieu: Our meeting with gradual, a good strategy for the iterated prisoner's dilemma, in Christopher G. Langton and Taksunori Shimohara

(Eds.), Artificial Life V: Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems (Nara, Japan, 1996), MIT Press, Cambridge, MA, pp. 202-209, 1997.

[Campbell02] Murray Campbell, A. Joseph Hoane Jr., Feng-hsiung Hsu: Deep Blue, Artificial Intelligence 134 57-83, 2002.

[Chellapilla99a] Kumar Chellapilla, David B. Fogel: Evolution, Neural Networks, Games, and Intelligence, Proceedings of the IEEE 87(9) 1471-1496, 1999.

[Chellapilla99b] Kumar Chellapilla, David B. Fogel: Evolving Neural Networks to Play Checkers without Expert Knowledge, IEEE Transactions on Neural Networks 10(6) 1382-1391, 1999.

[Clark97] David Clark: Deep thoughts on Deep Blue, IEEE Expert: Intelligent Systems and Their Applications 12(4) 31, 1997.

[Davis87] Lawrence Davis, Martha Steenstrup: Genetic Algorithms and Simulated Annealing: An Overview, in Lawrence Davis (Ed.), Genetic Algorithms and Simulated Annealing, Pitman, London/Morgan Kaufmann, Los Altos, CA, 1987.

[Davis91] Lawrence Davis: Handbook of Genetic Algorithms, Van Nostrand Reinhold, 1991.

[DeJong75] Kenneth A. De Jong: An Analysis of the Behavior of a Class of Genetic Adaptive Systems, Ph.D. Thesis, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, 1975.

[DeJong88] Kenneth A. De Jong: Learning with Genetic Algorithms: An Overview, Machine Learning 3 (2-3) 121-138, 1988.

[Epstein92] Susan L. Epstein: The Role of Memory and Concepts in Learning, Minds and Machines 2 239-265, 1992.

[Epstein94] Susan L. Epstein: Toward an Ideal Trainer, Machine Learning 15(3) 251-277, 1994.

[Faybish99] Itamar Faybish: Applying the genetic algorithm to the game of Othello, M.Sc. Thesis, Vrije Universiteit Brussel, Computer Science Department, Brussels, Belgium, 1999.

[Fekete99] Fekete István, Gregorics Tibor, Nagy Sára: Bevezetés a mesterséges intelligenciába, LSI Oktatóközpont, Budapest, 1999.

[Ferrer95] Gabriel J. Ferrer, Worthy N. Martin: Using genetic programming to evolve board evaluation functions. in Proceedings of the 1995 IEEE Conference on Evolutionary Computation, Perth, Australia, 1995.

[Fogel93a] David B. Fogel: Evolving Behaviors in the Iterated Prisoner's Dilemma, Evolutionary Computation 1(1) 77-97, 1993.

[Fogel93b] David B. Fogel. Using Evolutionary Programming to Create Neural Networks that are Capable of Playing Tic-Tac-Toe, in Proceedings of the IEEE International Conference on Neural Networks (San Francisco, 1993), IEEE Press, Piscataway, NJ, pp. 875-880, 1993.

[Fogel00] David B. Fogel: Evolutionary Computation: Toward a New Philosophy of Machine Intelligence, IEEE Press, New York, 2000.

[Fogel66] L. J. Fogel, A. J. Owens, M. J. Walsh: Artificial Intelligence through Simulated Evolution, John Wiley and Sons, New York, 1966.

[Fraser94] Adam P. Fraser: Genetic Programming in C++, Technical Report 040, Cybernetics Research Institute, University of Salford, UK, 1994.

[Freisleben94] Bernd Freisleben, Hartmut Luttermann: Learning to Play the Game of Go-Moku: A Neural Network Approach, Australian Journal of Intelligent Information Processing Systems 3(2) 52-60, 1996.

[Generation03] generation5.org, 2003:

a) www.generation5.org/aisolutions/tspapp.shtml.
b) www.generation5.org/ga_math.shtml.
c) www.generation5.org/aisolutions/ga00.shtml.
d) www.generation5.org/ga.shtml.
e) www.generation5.org/coevolution.shtml.
f) www.generation5.org/ipd.shtml.

[Goldberg88] David E. Goldberg, John H. Holland: Genetic Algorithms and Machine Learning, Machine Learning 3 (2-3) 95-99, 1988.

[Goldberg89] David E. Goldberg: Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, 1989.

[Hillis92] Daniel W. Hillis: Co-Evolving Parasites Improve Simulated Evolution as an Optimization Procedure, in Christopher G. Langton, Charles Taylor, J. Doyne Farmer and Steen Rasmussen (Eds.), Artificial Life II, Addison-Wesley, Reading, MA/Santa Fe Institute Studies, New Mexico, pp. 313-324, 1992.

[Holland92] John H. Holland: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology Control, and Artificial Intelligence, First MIT Press Edition, Cambridge, MA, 1992.

[Hong98] Tzung-Pei Hong, Ke-Yuan Huang, Wen-Yang Lin: A Genetic Minimax Game-Playing Strategy, in Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, pp. 690-694, 1998.

[Hong02] Tzung-Pei Hong, Ke-Yuan Huang, Wen-Yang Lin: Applying genetic algorithms to game search trees, Soft Computing 6(3-4) 277-283, 2002.

[Janikow93] Cezary Z. Janikow: A Knowledge-Intensive Genetic Algorithm for Supervised Learning, Machine Learning 13 (2-3) 189-228, 1993.

[Jelasity99] Jelasity Márk: Genetikus algoritmusok, Mesterséges intelligencia-ban, szerkesztő Futó Iván, Aula Kiadó, Budapest, 549-568 old., 1999.

[Kim02] Kyung-Joong Kim, Sung-Bae Cho: Checkers Strategy Evolution with Speciated Neural Networks, in M. Ishizuka and A. Sattar (Eds.), Proceedings of the 7th Pacific Rim International Conference on Artificial Intelligence: Trends in Artificial Intelligence (Tokyo, Japan, 2002), Springer-Verlag, Berlin Heidelberg, p. 599, 2002.

[Kirkpatrick83] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi: Optimization by Simulated Annealing, Science 220 671-680, 1983.

[Kis99] Kis Tamás: Fejlett kereső algoritmusok, Mesterséges intelligencia-ban, szerkesztő Futó Iván, Aula Kiadó, Budapest, 243-279 old., 1999.

[Kojima98] Takuya Kojima: Automatic Acquisition of Go Knowledge from Game Records: Deductive and Evolutionary Approaches, Ph.D. Thesis, University of Tokyo, Japan, 1998.

[Konidaris02] George Konidaris, Dylan Shell, Nir Oren: Evolving Neural Networks for the Capture Game, SAICSIT Postgraduate Research Symposium, 2002.

[Koza92] John R. Koza: Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, MA, 1992.

[Michalewicz96] Zbigniew Michalewicz: Genetic Algorithms + Data Structures = Evolution Programs, 3rd Revision edition, Springer-Verlag, Berlin-Heidelberg-New York, 1996.

[Military01] US Military: GA Archives, 2001: www.aic.nrl.navy.mil/galist/.

[Mitchell96] Melanie Mitchell: An Introduction to Genetic Algorithms, MIT Press, Cambridge, MA, 1996.

[Nagy99] Nagy Sára: Kétszemélyes játékok, a Mesterséges intelligencia-ban, szerkesztő Futó Iván, Aula Kiadó, Budapest, 215-242 old., 1999.

[Neumann44] John von Neumann, Oskar Morgenstern: Theory of Games and Economics Behavior, Princeton University Press, Princeton, New Jersey, 1944.

[Pécsy98] Pécsy Gábor: Genetikus algoritmusok a képfeldolgozásban, Szakdolgozat, Eötvös Loránd Tudományegyetem, Természettudományi Kar, Programtervező-matematikus Szak, Budapest, 1998.

[Pollack97] Jordan B. Pollack, Alan D. Blair, Mark Land: Coevolution of a backgammon player, in Christopher G. Langton and Taksunori Shimohara (Eds.), 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.

[Pollack98] Jordan B. Pollack, Alan D. Blair: Co-Evolution in the Successful Learning of Backgammon Strategy, Machine Learning 32(3) 225-240, 1998.

[Rosin95] Christopher D. Rosin, Richard K. Belew: Finding opponents worth beating: Methods for competitive co-evolution, in Proceedings of the 6th International Conference on Genetic Algorithms (ICGA 95), 1995.

[Rosin97a] Christopher D. Rosin, Richard K. Belew: New methods for competitive coevolution, Evolutionary Computation 5(1) 1-29, 1997.

[Rosin97b] Christopher D. Rosin: Coevolutionary Search Among Adversaries, Ph.D. Thesis, University of California, San Diego, CA, 1997.

[Ross02] Don Ross: Game Theory, in Edward N. Zalta (Ed.), Stanford Encyclopedia of Philosophy, Spring 2002 Edition: http://plato.stanford.edu/entries/game-theory.

[Shannon50] C. E. Shannon: Programming a computer to play chess, Philosophy Magazine 41 256-275, 1950.

[SMAC99] SMAC Team: PRISON Project, 1999:

a) www.lifl.fr/IPD/ipd.html.

b) www.lifl.fr/IPD/references/from_lifl/ep98/html.

[Whitley93] Darrell Whitley, Stephen Dominic, Rajarshi Das, Charles W. Anderson: Genetic Reinforcement Learning for Neurocontrol Problems, Machine Learning 13 (2-3) 259-284, 1993.