1.1 Genetikus algoritmusok

A genetikus algoritmus (*genetic algorithm,* GA) iránt mutatkozó érdeklődésnek sok oka van, de egy dolog biztosan fontos szerepet játszik: bizonyos mértékig kapcsolatban áll az evolúció darwini elméletével, márpedig ennek a puszta említése is heves érzelmi reakciókat vált ki sok emberből. Komolyabbra fordítva a szót, a módszer egyik fő előnye, hogy a számítástechnikában előforduló problémák egy nagyon széles osztályára alkalmazható, ugyanakkor általában nem használ területfüggő tudást, így akkor is működik, ha a feladat struktúrája kevéssé ismert. Ebből a szempontból a problémafüggetlen metaheurisztikák csoportjába tartozik, amelyek közül a legismertebbek a **szimulált hűtés** (*simulated annealing*), a **tabu keresés** (*tabu search*) és a különböző **hegymászók** (*hill climbers*).

Bár a tanulásról szóló fejezetben kapott helyet, a módszer (mint a metaheurisztikák általában) valójában egy **globális optimalizáló**, amely a 6. fejezetben ismertetett módszerekkel rokon. Mindenhol alkalmazható, ahol a feladat sok lehetséges megoldás közül a legjobbat megkeresni, ahol az értéket egy értékelőfüggvény, másnéven **rátermettségi függvény** (*fitness function*) adja meg. A genetikus algoritmus megoldások egy **populáció**ját tartja fenn, azaz egyszerre több megoldással dolgozik. Az aktuális populációból minden lépésben egy új populációt állít elő úgy, hogy a **szelekciós operátor** által kiválasztott legrátermettebb elemeken (szülőkön) alkalmazza a **rekombinációs** és **mutációs operátorokat**. Az alapgondolat az, hogy mivel általában minden populáció ez előzőnél rátermettebb elemeket tartalmaz, a keresés folyamán egyre jobb megoldások állnak rendelkezésre. Elvben minden tanulási probléma megadható optimalizálási feladatként, így a genetikus algoritmusnak is sok alkalmazása van a gépi tanulás területén.

A fejezet felépítése a következő. Egy rövid történeti áttekintés és a rokon területek vázolása után viszonylag részletesen foglalkozunk a gráfszínezés problémájával. Ennek az az oka, hogy ezen a területen ismert egy kifejezetten sikeres genetikus algoritmus, ráadásul sok fontos technika is előkerül majd. Ezután röviden kitérünk a gépi tanulással kapcsolatos leggyakoribb alkalmazásokra, majd az algoritmusnak egy kissé általánosabb felírását adjuk meg, és ennek segítségével rámutatunk a többi metaheurisztikával fennálló rokonságra. Néhány fontosabb elméleti megközelítésre is kitérünk, inkább csak az említés szintjén. Végül azt vizsgáljuk meg, hogy a genetikus algoritmusnak mi a köze az evolúcióhoz, mennyiben használható annak modellezésére.

A korábbi fejezetekkel komoly átfedések vannak (főleg a 6. fejezettel), mégis indokolt néhány fogalmat újra elővenni, mert egyrészt a genetikus algoritmusokkal foglalkozó szakemberek (a terület sajátos történetével magyarázhatóan) biológiai ihletésű terminológiát alkalmaznak, másrészt így lehetőség nyílik az algoritmus egy önállóan is olvasható, a hegymászó típusú (vagy lokális) módszereken alapuló elemzésére.

1.1.1 Történet, irányzatok

A hatvanas években merült fel először az a gondolat, hogy az evolúcióban megfigyelhető szelekciós folyamatok mintájára olyan számítógépes modelleket lehetne létrehozni, amelyek képesek mérnöki (elsősorban optimalizálási) feladatok megoldására.

Egymástól függetlenül több próbálkozás is született. Németországban Rechenberg vezette be az **evolúciós stratégiák**nak (*Evolutionsstrategie*) nevezett módszert (Rechenberg 1973), amelyet pl. repülőgép-szárnyak valós paramétereinek az optimalizálására használt. Később Schwefel továbbfejlesztette az elgondolást. Az evolúciós stratégiák ma is a szelekció alapú heurisztikáknak egy viszonylag önállóan fejlődő ága, egy rövid bevezetőt nyújt (Bäck et al. 1991). A többi próbálkozás mind Amerikában történt. Fogel, Owens és Walsh egyszerű problémák megoldására szolgáló véges automaták automatikus kifejlesztésével kísérletezett (Fogel et al. 1966). A kiindulási automaták állapotátmenet mátrixát véletlenszerűen megváltoztatták (azaz mutációt alkalmaztak) és ha az új automata rátermettebb volt, kiválasztásra került. Az új területnek az **evolúciós programozás** nevet adták (*evolutionary*

programming), amely szintén ma is művelt terület. Egy nagyon hasonló, de sokkal frissebb terület, a **genetikus programozás** (*genetic programming*) is említést érdemel. Ez lényegében a genetikus algoritmus egy speciális alkalmazási területe, amikor is a cél meghatározott feladatokat végrehajtó számítógép programok (leggyakrabban LISP nyelven) automatikus kifejlesztése. Az első ilyen irányú próbálkozás Koza nevéhez fűződik (Koza 1992, 1994) aki ma is a terület vezető alakja.

A genetikus algoritmusok kifejlesztése Holland nevéhez fűződik. Ő és diákjai alapozták meg a University of Michigan egyetemen a területet, amely kutatás eredményeit Holland foglalta össze (Holland 1975). Az ő célja kezdetben nem optimalizáló módszer kifejlesztése, hanem a szelekció és az adaptáció számítógépes és matematikai modellezése volt. A könyvben kifejtett elvekről még szó lesz, de annyit elöljáróban megjegyezhetünk, hogy azóta számtalan kritika érte ezeket az eredményeket.

A fent említett négy fő terület gyűjtőneve **evolúciós számítások** (*evolutionary computation*). Ezek a területek a mai napig megőrizték identitásukat, de nem kizárt, hogy ennek inkább történeti mintsem lényegi okai vannak. Mostanában megfigyelhető az egyre élénkebb információcsere a területek között, és ahogy majd látszik a későbbiekben, a módszerek fő komponensei és alapelvei lényegében megegyeznek. Ettől függetlenül ebben a fejezetben a genetikus algoritmusoknál megszokott terminológiát használjuk majd.

1.1.2 A gráfszínezési probléma

Azért választottuk a gráfszínezési problémát a szemléltetés céljára, mert a hasonló típusú feladatok a genetikus algoritmusok tipikus alkalmazási területei, és lehetőségünk lesz érinteni sok fontos és gyakran használt technikát, amiknek más problémák esetében is jó hasznát lehet venni. Ezen felül erre a problémára olyan genetikus algoritmust javasoltak (Eiben, Hauw 1998) amely jobb teljesítményt nyújtott mint a leggyakrabban alkalmazott heurisztika, a DSatur (Brélaz 1979).

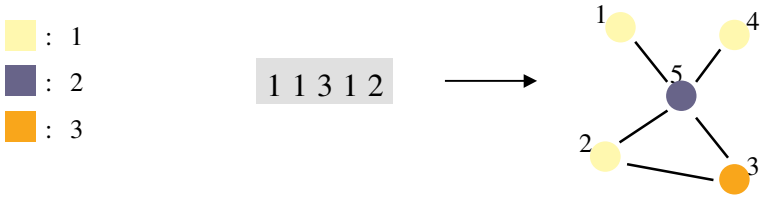
A probléma esetünkben az, hogy egy adott irányítatlan gráfhoz találnunk kell egy jó *k*-színezést, ami azt jelenti, hogy minden csúcshoz az előre adott *k* különböző szín valamelyikét kell rendelni úgy, hogy minden él két különböző színű pontot kössön össze.

Ahogy a bevezetőben már említettük, a genetikus algoritmus populációk sorozatát állítja elő operátorok segítségével. A populációk megoldásokat tartalmaznak. Itt egy megoldás a gráf egy (nem feltétlenül jó) színezése. Ahhoz, hogy ezekből a megoldásokból egyszerű operátorok segítségével könnyen újabb megoldásokat lehessen szerkeszteni, a megoldásokat **kódolni** kell. Ez többnyire azt jelenti, hogy minden megoldáshoz hozzárendelünk egy szintaktikailag jól manipulálható betűsorozatot egy **kódoló függvény** segítségével; ez lesz a megoldás **genotípusa**, míg maga a megoldás a **fenotípus**. Ez a kódolás és a hozzá tartozó operátorok a genetikus algoritmus kulcsfontosságú részei, mivel ezek határozzák meg a keresési tér topológiáját, más szóval itt dől el, hogy az algoritmus nézőpontjából mely megoldások kerülnek „közel", és melyek „távol" egymástól.

A következőkben ismertetjük a gráfszínezési probléma két lehetséges kódolását és a hozzájuk tartozó operátorokat. Az egyik kódolás magát a színezést kódolja, a másik pedig lényegében az algoritmust, amely elvégzi azt. A bevezetőben említett rátermettségi függvény a két esetben különböző lesz, így azokat is a megfelelő helyen adjuk meg. A nem tárgyalt komponenseket (mint pl. a szelekciós operátor) a későbbi fejezetekben vesszük sorra, ezek ugyanis problémafüggetlenek.

1.1.2.1 A színezés direkt kódolása

Egy adott színezés kódolása ebben az esetben nagyon egyszerű. A gráf pontjaihoz rendelt színeket (pontosabban a színekhez rendelt sorszámokat) egyszerűen felsoroljuk, ez a számsorozat lesz a kód. A dekódoláshoz természetesen tudni kell, hogy melyik csúcshoz melyik szín tartozik, tehát a felsorolásnál a pontokat egy előre rögzített sorrendben kell érinteni. A kód hossza értelemszerűen a gráf pontjainak a száma. A kódolást szemlélteti az **Hiba! A hivatkozási forrás nem található.**.

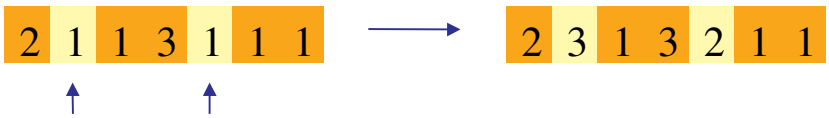


1. ábra. Adott gráf színezéseinek direkt kódolása.

Ennek a kódolásnak az az előnye, hogy a segítségével könnyű előállítani véletlenszerű színezéseket, illetve újakat régi színezések felhasználásával, hiszen az egyes betűket egymástól függetlenül lehet megadni. Egy másik tulajdonsága az, hogy a dekódolás minden ponthoz rendel egy színt, a rátermettségi függvényt tehát definiálhatjuk úgy, hogy a rosszul színezett (azaz azonos színű pontokat összekötő) élek számának a függvénye legyen: minél kevesebb a rossz él, annál nagyobb legyen a rátermettsége a kérdéses megoldásnak. Egy színezés rátermettsége ennek megfelelően lehet pl. a rossz élek számának a mínusz egyszerese, így minden jó színezés maximális rátermettségű lesz.

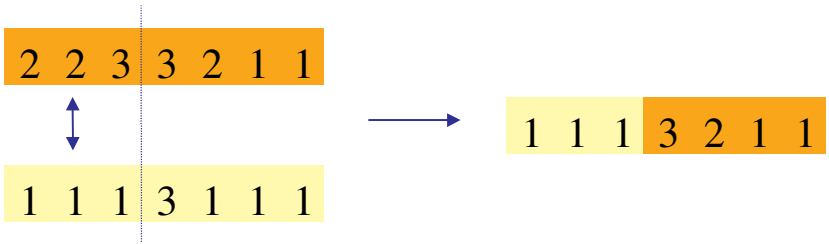
A kódokon alkalmazott operátorokat szokás két csoportba sorolni. Az elsőbe tartoznak azok, amelyek egy megoldásból állítanak elő egy újabbat, ezek a **mutáció**k. A másik csoportba tartoznak azok, amelyek több kiindulási megoldásból (szülőből) állítanak elő újabb megoldást a szülők kódjainak valamilyen kombinációja segítségével. Ezeket **rekombináció**knak nevezik. Az itt tárgyalt kódoláshoz is megadható rengeteg féle mutáció és rekombináció, itt azokat ismertetjük, amelyeket a legszélesebb körben alkalmaznak.

Mutációra a legegyszerűbb példa 2. ábra, ahol véletlenszerűen kiválasztott pozíciókat változtatunk meg. A pozíciók kiválasztására sokféle módszer alkalmazható, pl. minden pozíció egy rögzített (általában kicsi) valószínűséggel mutálódhat. Sokszor úgy állítják be ezt a valószínűséget, hogy átlagosan egy pozíció változzon meg egy kódban.



2. ábra. Mutáció operátor.

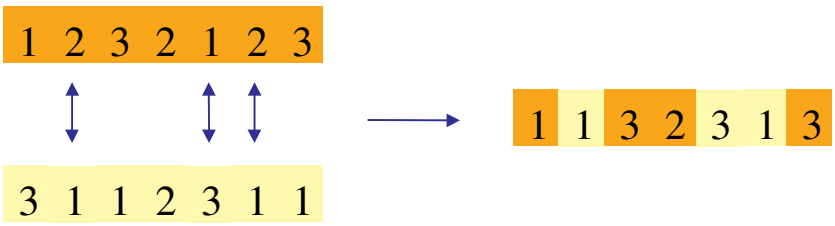
Rekombinációra két példát is adunk. Az első és egyben a legrégebbi operátor az **egypontos keresztezés** (*1-point crossover*), amit a 3. ábra szemléltet. Véletlenszerűen választunk egy keresztezési pontot, és a keletkezett fél kódokból új megoldást rakunk össze. Ez a keresztezés sokféleképpen általánosítható, pl. több szülő, vagy több keresztezési pont felhasználásával.



3. ábra. Rekombinációk: egypontos keresztezés.

Gyakran alkalmazott operátor még az **egyenletes keresztezés** (*uniform crossover*, UX), ami egy lényegesen különböző felfogást tükröz (4. ábra). Itt a mutációhoz teljesen hasonlóan kiválasztott pozíciókon a kiindulási kódok betűket cserélnek. A mutációtól eltérően azonban

itt egy pozíció kiválasztásának a valószínűsége rendszerint 0,5, azaz átlagosan a betűk fele cserélődik ki.



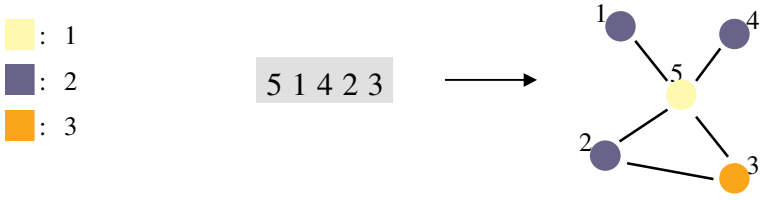
4. ábra. Rekombinációk: egyenletes keresztezés.

Az egypontos keresztezés ötlete a biológiából származik, azonban az egyenletes keresztezés már nem rendelkezik hasonló gyökerekkel. Általában is azt mondhatjuk, hogy itt az evolúciós analógiától való eltávolodás figyelhető meg, a választott operátorokat a megoldani kívánt probléma befolyásolja. Különösen igaz ez a sorrendi kódolásra és operátoraira.

1.1.2.2 A színezés sorrendi kódolása

A sorrendi kódolás (nem meglepő módon) általában valamilyen elemek sorrendjének a kódolását jelenti. Esetünkben ezek az elemek a színezendő gráf pontjai lesznek. Mielőtt elmerülnénk a részletekben, érdemes megemlíteni, hogy a sorrendi kódolásnak más, nagyon fontos alkalmazásai is vannak. Ilyen az utazó ügynök probléma, ahol (egyszerűbb esetben) egy térképen kell egy olyan körutat tervezni, amelynek a hossza minimális, de amely érinti az összes felkeresendő helyet. Itt egy lehetséges megoldás nyílván kódolható a helyek felkeresésének a sorrendjével. Ennek a problémának a genetikus algoritmussal történő megoldására irányuló törekvések termelték ki a sorrendi kódolást és az azt kezelő operátorokat.

Egy színezést természetesen nem lehet pusztán a gráf pontjainak a sorrendjével megadni. Ehhez szükség van egy egyszerű heurisztikára is, amely a pontok egy sorrendjéből kiindulva elvégzi a színezést. Erre a célra tökéletesen megfelel a következő algoritmus: vegyük sorra a pontokat, és minden pontot színezzünk arra a minimális sorszámú színre, amelynek használata az adott helyzetben lehetséges. Ha egyetlen szín sem megfelelő, hagyjuk színezetlenül a pontot. Ez az algoritmus minden pont-sorrendhez egyértelműen hozzárendel egy színezést. Ha létezik jó színezés, akkor minden pontnak lesz színe, egyébként maradnak színezetlen pontok. Az 5. ábra egy egyszerű példával szemlélteti a módszert.

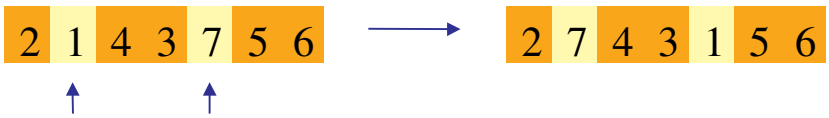


5. ábra. A dekódoló heurisztika végeredménye az adott pont-sorrendre alkalmazva a feltüntetett szín-sorszámokkal.

A fenti dekódoló algoritmus természetes módon kínál egy lehetőséget a rátermettség definíciójára: minél kevesebb a színezetlen pont, annál rátermettebb a megoldás. Ennek megfelelően legyen egy gráf pontjai egy-sorrendjének a rátermettsége a dekódoló heurisztika végrehajtása után még színezetlen pontok számának a mínusz egyszerese.

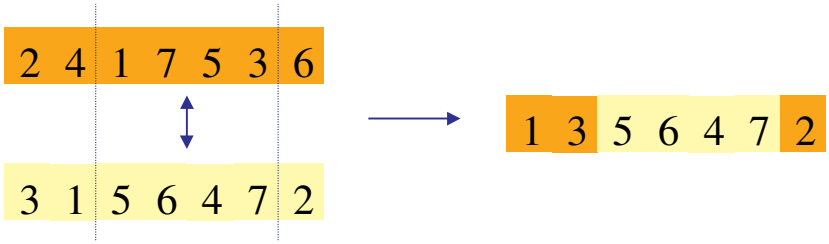
Az operátorok egy kissé bonyolultabbak mint a direkt kódolás esetében. Mutációra és rekombinációra is bemutatunk egy-egy példát, de számos más lehetőség is van ilyen

operátorok definiálására. A mutáció leggyakrabban egy véletlenszerűen kiválasztott elemi permutáció végrehajtását jelenti (6. ábra).



6. ábra. Az elemi permutáció mint sorrendi mutáció.

Több problémát okoz a rekombináció, hiszen az egyes pozíciókon található betűk erősen függnek egymástól. Ennek következtében ez előző fejezetben ismertetett egypontos keresztezés nem alkalmazható kiegészítések nélkül. Egy lehetséges sorrendi rekombináció az OX (*order crossover*), amit a 7. ábra szemléltet.



7. ábra. Az OX rekombináció.

Az OX rekombináció végrehajtása két keresztezési pont véletlenszerű kiválasztásával kezdődik. Ezután a két pozíció közötti részek kicserélődnek. A fennmaradó részek kitöltéséhez először megvizsgáljuk, hogy az eredeti szülőben az új középső részben nem szereplő pontok milyen sorrendben következnek (esetünkben ez „2 1 3”), és ezt a sorrendet követve a második keresztezési ponttól kezdve beírjuk a hiányzó pontokat.

Akár sorrendi akár másmilyen reprezentációt használunk egy probléma megoldásához, nincsen általános recept az operátorok megválasztására. Az egyes operátorok különböző feladatokon nagyon eltérő teljesítményt produkálhatnak, másfelől nincsenek általánosan alkalmazható elméleti eredmények, amelyek segítenének az operátorok megválasztásában. Minden feladathoz érdemes többfélét kipróbálni tehát.

1.1.3 Alkalmazások a gépi tanulásban

Mielőtt rátérnénk a problémafüggetlen operátorok tárgyalására, két fontos területet érdemes megemlíteni, ahol a genetikus algoritmusok alkalmazásával kapcsolatos komoly kutatások folynak. Az egyik a számítógép programok automatikus fejlesztése, a már említett genetikus programozás, a másik az **evolúciós mesterséges neuron hálók** (*evolutionary artificial neural networks*, EANN) témaköre amelyről jó összefoglalót ad (Yao 1993). Mindkét terület csak alig több mint egy évtizedes múltra tekint vissza. Ahogy már említettük, a genetikus algoritmus egy globális optimalizáló módszer, amely egy keresési tér minél rátermettebb egyedeit hivatott felfedezni. *Nagyon* röviden és tömören térjünk ki arra, hogy a fenti két területet hogyan lehet „emészthetővé” tenni a genetikus algoritmus számára, azaz milyen kódolást és operátorokat alkalmazhatunk.

A genetikus programozás esetében a keresési tér gyakran előre adott műveletekből (pl. +, -, sin, stb.), konstansokból és változónevekből összeállítható függvényekből áll, és az egyes függvények rátermettsége pedig egy „fekete dobozként” adott függvénytől való eltéréstől függ. Az ilyen feladatokat szokás **szimbólikus regresszió**nak nevezni. Az egyes függvényeket a szintaxis fájukkal kódoljuk, amelyben az operátorok leszármazottai az argumentumaik. A rekombináció a szülők véletlenszerűen kiválasztott részfáinak a cseréje, a

mutáció (amit egyébként ritkán alkalmaznak) valamely változó, operátor vagy konstans szintaktikailag megengedhető véletlenszerű módosítása.

A mesterséges neuron hálók három szinten is taníthatók evolúciós módszerekkel: a kapcsolatok súlyainak a beállítása útján, a háló struktúrájának (rétegek, neuronok, kapcsolatok) megtervezésével, és a tanítási módszer kifejlesztésével. Ezek persze kombinálhatók is. A súlyok kódolása rendszerint egyszerűen egy valós vektor, a struktúráé vagy direkt kódolás, pl. a kapcsolatmátrix, vagy szabály alapú, ahol a háló kifejődésének a módszerét, nem magát a hálót kódoljuk. A tanulási szabályok kódolása a keresési tér meghatározásától függ; gyakran egy bevett módszer (pl. backpropagation) paramétereiről van szó. Az alkalmazott operátorok ismertetését meg sem kíséreljük, mert túl sok szempontot és módszert kéne érinteni, az olvasó itt a megadott irodalomra támaszkodhat.

1.1.4 Problémafüggetlen komponensek

A gráfszínezési probléma kapcsán különböző kódolási technikákat és hozzájuk tartozó operátorokat tárgyaltunk, amelyeknek az volt a feladata, hogy ismert megoldásokból újakat állítsanak elő. A kódolás és operátorai akkor megfelelőek, ha rátermett megoldásokból kiindulva a leszármazott megoldások is hasonlóan jó tulajdonságokkal rendelkeznek. Feltéve, hogy a kódolás jó, nem mindegy, hogy az operátorainkat mely megoldásokra alkalmazzuk amikor a soron következő populációt szeretnénk előállítani. Ezeknek a „szülőknek" a kiválasztása a **szelekciós operátor**, vagy egyszerűen a **szelekció** feladata, amely már problémafüggetlen. Egészen pontosan fogalmazva a szelekció egy populációból kiválaszt a számunkra egy darab megoldást, és ezt persze annyiszor kell megismételnünk, ahány szülőre szükség van az új populáció előállításához.

A szelekció sokféle implementációjának az alapja mindig a rátermettség, de egyéb tekintetben a megvalósítások eléggé különbözőek (Goldberg 1991). Két szelekciós operátort vázolunk, amelyek közül az első, a **rátermettség-arányos szelekció** (*fitness proportionate selection*), a történetileg a legrégebbi és ez tükrözi legjobban a szelekció célját. A másik a **pár-verseny szelekció** (*binary tournament selection*), amely hatékonyabb implementációt tesz lehetővé. Végül szó esik a **rangsorolás** (*ranking*) technikájáról, amivel bármely szelekció kiegyensúlyozottabbá tehető.

1.1.4.1 Rátermettség-arányos szelekció

Mivel a módszer valószínűségi mintavételt használ, nem kerülhetjük ki a matematikai pontosságú definíciót, de a lényeg azért egyszerűen megragadható. Egy megoldás kiválasztásának a valószínűsége annál nagyobb, minél nagyobb a rátermettsége a populáció rátermettségi átlagához képest. A szelekció matematikai értelemben egy mintavétel a populációból. A populáció minden e elemére a kiválasztódás valószínűségét megadhatjuk a

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

képlettel, ahol $f(e)$ a rátermettség értéke, n a populáció elemszáma, és $f(Pop)$ a populáció tagjainak átlagos rátermettsége. A figyelmes olvasó rögtön észrevette, hogy itt feltettük, hogy a rátermettség értéke pozitív. Ez a gyakorlatban nem jelent problémát, hiszen könnyen kaphatunk alsó korlátokat a populáció tagjainak a rátermettségére, amit kivonva az egyes értékekből biztosítható a pozitivitás. Ennek a módszernek persze hatása van a kiválasztási valószínűségekre is, amit tudatosan ki is lehet használni, erre azonban nem térünk ki.

Itt érdemes megjegyezni, hogy főleg az elméleti irodalomban a rátermettség fogalmát néha ennek a kiválasztási valószínűségnek (illetve más szelekciók esetén az ott kiszámolható hasonló szerepű értéknek) a jelölésére tartják fenn, és az általunk rátermettségnek nevezett értéket célfüggvény értéknek nevezik. Ebben az értelemben a rátermettség függ a populációtól amelyben vizsgáljuk. Kétségtelen, hogy ennek a szóhasználatnak vannak előnyei, azonban a másik értelmezés sokkal gyakoribb.

1.1.4.2 Pár-verseny szelekció

A pár-verseny szelekció talán a legegyszerűbb módszer, és sok alkalmazásban lehet vele találkozni. Válasszunk ki a populációból két megoldást teljesen véletlenszerűen és a szelekció által kiválasztott elem legyen a kettő közül a rátermettebb. Ugyanez a technika általánosítható úgy, hogy nem kettő hanem több elem győztesét választjuk ki. A három elemet használó verzió pl. szintén népszerű.

1.1.4.3 Rangsorolás

A rátermettség-arányos szelekcióhoz hasonló, a rátermettség értékét közvetlen módon használó módszerek közös hátulütője, hogy túlságosan érzékenyek a rátermettség eloszlására a populációban. Ha például egy megoldás túlságosan magas értékkel rendelkezik a többihez képest, akkor szinte mindig ő lesz a kiválasztott szülő, aminek nagyon káros hatásai vannak, hiszen a keresés a változatosság gyors csökkenése miatt „beragadhat"; az aktuális legjobb megoldást tovább javítani egyre nehezebb lesz. Ezt a problémát megoldhatjuk úgy, hogy a populáció elemeit rátermettség szerint sorba rendezzük, és a rátermettség helyett valamilyen „szép", a rendezésben elfoglalt hely szerint egyenletesen növekvő függvényt használunk. Gyakori pl. a **lineáris rangsorolás** (*linear ranking*), amikor ez a „szép" függvény egy egyenes.

1.1.5 Az algoritmus általános szerkezete, implementáció

A genetikus algoritmus minden komponensét érintettük már, és mindegyikre szerepelt több példa is. Itt összefoglaljuk ezeket a komponenseket, és a fontosabb paraméterek leggyakrabban használt értékeit is megadjuk azok számára, akik maguk is szívesen kísérleteznének az algoritmussal. Ezen kívül lehetőség lesz összevetni egymással a fontosabb metaheurisztikákat is. Az általános váz a következőképpen adható meg:

```
1.Hozzuk létre a P0 kezdeti populációt
2.t := 0
3.while not Kilépés(Pt)
4.           Pt' := UjElemek(Pt)
5.           Pt+1 := UjPopuláció(Pt',Pt)
6.           t := t+1
```

Az eddigiekhez képest ez nem sok újdonságot tartalmaz. A kezdő populáció feltöltése leggyakrabban véletlen elemekkel történik, de léteznek eredmények, amik azt támasztják alá, hogy érdemes lehet valamilyen egyszerű heurisztika segítségével viszonylag jó teljesítményű megoldásokból kiindulni. A populációk elemszáma a futás során változatlan. Konkrétan az 50-100 körüli értékek a tipikusak, de a genetikus programozásban gyakran több ezres populációkkal dolgoznak. Általában érdemes először kisebb populációkkal próbálkozni.

A `Kilépés(`P_t`)` függvény sokszor nem is függ a populációtól, mint ahogy a jelölés sejtetné, a tipikus módszer a már kiértékelt megoldások számának a vizsgálata. Az algoritmus futása ebben az esetben akkor áll le, ha egy előre adott mennyiségű különböző lehetséges megoldás rátermettségét kiszámoltuk. Ezt az indokolja, hogy a futási idő java részét általában a megoldások kiértékelése adja, így ez a legfontosabb szempont. A megengedett kiértékelések konkrét száma függ a problémától, leggyakrabban 1000 és 500000 között van, extrém esetekben több is lehet. Nemritkán ilyen extrém esetnek számítanak a gépi tanulás körébe tartozó problémák.

Az `UjElemek(`P_t`)` függvény feltölti a P_t' populációt új elemekkel. Egy új megoldás előállítása legtöbbször úgy történik, hogy a szelekció segítségével szülőket választunk, majd a rekombináció ezekből egy utódot hoz létre. Erre aztán alkalmazható a mutáció, majd az új elem rátermettségének a meghatározása következik. A P_t' populáció elemszáma nem feltétlenül azonos P_t elemszámával. Ha mégis azonos, az `UjPopuláció(`P_t`',`P_t`)`

függvény egyszerűen a P_t' populációt adja. Ebben az esetben a genetikus algoritmus **generációs** (*generational*). Ha azonban P_t' elemszáma kisebb, a megfelelő mennyiségű elemet törli a régi populációból, és a helyükre P_t' elemei kerülnek. Abban a szélsőséges esetben, amikor P_t' mindössze egy elemű, a genetikus algoritmust **helybeni**nek (*steady state*) nevezzük. Az elemek törlésére a szelekcióhoz hasonló operátorok alkalmazhatók, csak éppen fordított előjellel. A legtöbb implementációban a régi populáció legrátermettebb elemét is az új populációba helyezik akár ki lett választva akár nem, feltéve ha az új populációban egyébként minden elem rosszabb lenne. Ekkor az algoritmus **elitista** (*elitist*).

1.1.6 A genetikus algoritmus és a metaheurisztikák

Az itt említett algoritmusok részletes tárgyalása a 6. fejezetben található. Az alábbiak megértéséhez elegendő az itt megadott néhány mondatos ismertető, amely a genetikus algoritmusok irodalmában megszokott terminológiát és a fent megadott algoritmus-vázlat eljárásneveit használja.

A genetikus algoritmus és a többi metaheurisztika, esetünkben a szimulált hűtés, a tabu keresés és a sztochasztikus hegymászó, figyelemreméltóan sok közös tulajdonsággal rendelkezik. Az algoritmus mindegyiknél problémafüggetlen, de szükség van a keresési tér elemeinek, azaz a lehetséges megoldásoknak valamilyen kódolására és az azon értelmezett kereső operátorokra. A kódolás és az operátorok kulcsszerepet játszanak a keresés sikerességében, és egyben a probléma-specifikus tudás beépítésére is ezek biztosítanak lehetőséget. Közös tulajdonság még, hogy mindegyik módszer **globális**, azaz tartalmaz valamilyen technikát a lokális optimumokba való „beragadás" ellen. A lokális optimum olyan megoldás, amelytől kicsit eltávolodva mindig rosszabb minőségű megoldásokat kapunk, nagyobb távolságra azonban jobbak is léteznek, ahol a távolság fogalmát természetesen az alkalmazott kódolás és az operátorok határozzák meg. Ezen kívül maga a problémafüggetlen keresés folyamata is nagyon hasonló az egyes módszerek esetében; a fenti algoritmus-vázlat fogalmaiban megadható mindegyik heurisztika. Vegyük sorra, hogy az egyes heurisztikák esetében hogyan kell értelmezni az `UjElemek(`P_t`)` és az `UjPopuláció(`P_t'`,`P_t`)` függvényeket (a `Kilépés(`P_t`)` függvény lehet ugyanaz mint a genetikus algoritmus esetében).

A szimulált hűtés esetében a szokásos analógia a fémek edzésének a folyamata, amelynek során a fém lassú hűtés során közel optimálisan rendezett atomi szerkezetet vesz fel. Itt a populáció egyelemű és a kereső operátor a mutáció. Az `UjPopuláció(`P_t'`,`P_t`)` függvény azonban némileg eltér a genetikus algoritmusétól: Ha az új megoldás rosszabb, mint a régi, akkor is elfogadjuk egy bizonyos valószínűséggel, amit a **hőmérséklet** paraméter szabályoz. Ha a hőmérséklet nulla, csak akkor fogadjuk el, ha nem rosszabb mint a régi megoldás. A hőmérséklet a keresés folyamán fokozatosan csökken.

A tabu keresés szintén egyelemű populációt használ, és a kereső operátor szintén a mutáció. A különbség itt is az `UjPopuláció(`P_t'`,`P_t`)` függvényben van. A tabu keresés során egy tabu listát tartunk fenn, amely a legutóbb megvizsgált néhány megoldásból áll. A tabu lista mérete az algoritmus paramétere. Az új populáció, azaz az új aktuális megoldás kiválasztásához először megnézzük, hogy a mutációval létrehozott új elem szerepel-e a tabu listában. Ha igen, akkor nem fogadjuk el, egyébként pedig ha nem rosszabb mint a régi megoldás, akkor elfogadjuk. A régi megoldás a tabu listára kerül, és a tabu lista legrégebbi eleme törlődik.

A sztochasztikus hegymászó a legegyszerűbb. A populáció itt is egyelemű és a kereső operátor pedig a mutáció. Az új megoldás felváltja a régit, ha ugyan olyan rátermett vagy rátermettebb. Érdekes, hogy míg ezek a módszerek egymástól független természeti folyamatoknak a modelljei, és meglehetősen önálló területet alkotnak, a sztochasztikus hegymászó lényegében egy fix nulla hőmérsékleten futó szimulált hűtés, vagy éppen nulla hosszuságú tabu listát használó tabu keresés sőt egyelemű populációt használó elitista genetikus algoritmus. Az egyes metaheurisztikák tehát a hegymászó különböző általánosításaiként is felfoghatók. Másrészről ezeknek a módszereknek a különböző

kombinációi (vagy stílszerűen: hibridjei) is használatosak, pl. tabu listát alkalmazó szimulált hűtés, csökkenő mértékű mutációt alkalmazó genetikus algoritmus, stb.

1.1.7 Elméleti kutatások

A genetikus algoritmusok viselkedésének a magyarázatára szolgáló elméletek Holland munkájára építenek elsősorban (Holland 1975). Ez sok szempontból érdekes következményekkel jár és különösen a tudományszociológia iránt érdeklődők számára valóságos kincsesbánya. Holland célja ugyanis, ahogy arra már felhívtuk a figyelmet, *komplex adaptív rendszerek* tanulmányozása volt, nem pedig optimalizáló módszerek kifejlesztése. Az érdekelte, hogy a szelekció milyen módon befolyásolja a populációkban bizonyos típusú egyedek eloszlását, és a folyamat hogyan függ a rátermettségtől. Elméleti alapfeltevéseit is ez befolyásolta elsősorban. Ugyanakkor a genetikus algoritmusokat már a kezdetektől fogva, mintegy magától értetődően optimalizálásra is használni kezdték, és természetes módon a Holland által javasolt elméleti keretet és alapfogalmakat alkalmazták az optimalizálási folyamat jellemzésére. Ennek a legnagyobb hatású példája Goldberg építőkocka hipotézise (building block hypothesis) (Goldberg 1989), amely az *optimalizálási folyamat* egy elméleti modelljére tesz javaslatot Holland fogalmaira építve. Később egyre világosabbá vált, hogy az építőkocka hipotézis különböző okokból (amelyekre nemsokára visszatérünk) nem alkalmas arra a célra amelyre bevezették. A már-már válságosnak tűnő helyzetet annak a felismerése követte, hogy itt két különböző dologról van szó; a Holland által bevezetett elméleti keret pedig nem alkalmas, legalábbis változtatás nélkül, az optimalizációs célra alkalmazott, és a Holland által bevezetett kanonikus genetikus algoritmustól gyakran sok és lényeges szempontból eltérő algoritmusok kezelésére (Whitley 1993). Az elmélet alkalmazására azonban továbbra is rengeteg kísérlet születik, sokan még mindig Holland és Goldberg munkájára építenek az optimalizálási folyamat elemzésekor a kritikák, a szinte nyilvánvaló alapvető problémák és meggyőző kísérleti eredmények ellenére. Még érdekesebb ez annak a fényében, hogy az evolúciós stratégiák területe, amelyről korábban már volt szó, és amely egyébként ha egyáltalán valamiben, akkor csak terminológiájában, hagyományos alkalmazási területeiben és az egyes problémák preferált kódolásában különbözik a genetikus algoritmusoktól, teljesen különböző elméleti keretben dolgozik, aminek a magyarázatát talán a különböző gyökerekben kell keresnünk.

Mivel eddig a szokásos módon a genetikus algoritmusok optimalizációs célra történő alkalmazására helyeztük a hangsúlyt, nem követjük azt a gyakorlatot, hogy ezek után egy ettől különböző dolog elméleti tárgyalását részletezzük, helyette csak vázoljuk a háttérben meghúzódó alapfeltevéseket, majd néhány fontosabb irányvonalat is megemlítünk.

1.1.7.1 A kanonikus genetikus algoritmus és az optimalizálás

A kanonikus genetikus algoritmus, amire Holland eredményei vonatkoznak, a 0 és 1 bitekből álló, fix hosszúságú kódolást használ, azaz minden megoldásnak egy adott hosszú bitfüzér felel meg, pl. 100110. Az alkalmazott szelekció a rátermettség-arányos szelekció, az algoritmus generációs és *nem* elitista. Az alkalmazott operátorok a gráfszínezés direkt kódolásánál ismertetett egypontú keresztezés és mutáció. A mutáció valószínűsége nagyon kicsi.

Az elméleti tárgyaláshoz Holland **sémákat** (*schemata*) definiál, amik lényegében azok a kód-típusok amiknek speciális jelentőséget tulajdonított. Ilyen séma pl. az 010**; olyan kódok halmazát jelenti, amik 010-val kezdődnek, és az utolsó két betűjük tetszőleges. Világos, hogy vizsgálható az, hogy egy populációban mekkora egy adott séma aránya ill. átlagos rátermettsége. Holland azt vizsgálta, hogy az egymás után következő populációkban mekkora az egyes sémák aránya a rátermettségük függvényében, és adott sémára ennek a növekedésnek a várható értékére egy alsó korlátot is megadott, ami a **séma tétel** néven vált ismertté. Nem meglepő módon ez a tétel azt jósolja, hogy az átlagosnál rátermettebb sémák aránya közel exponenciálisan növekszik amíg ez az arány viszonylag kicsi. Ez a tétel akkor

értelmes, ha a populáció elég nagy méretű, és a rátermettség szórása az egyes sémákon belül elég kicsi.

Az építőkocka hipotézis a séma tételre épül, és a keresztezés szerepét hangsúlyozza. A modell szerint, mivel az átlagot meghaladó rátermettségű sémák aránya növekszik, és a keresztezés pedig éppen a sémák összekombinálására való, az optimalizáció során a populáció elemei olyan sémák kombinációi lesznek, amely sémák önmagukban átlagon felüli rátermettségűek. A modell jóslata tehát, hogy ha az optimális megoldást felépítő sémák átlagon felüliek, akkor a genetikus algoritmus könnyen megtalálja az optimumot, és fordítva: ha van olyan megoldás, amely ugyan kis rátermettségű, de az őt felépítő sémák rátermettek, akkor a probléma félrevezető lesz.

A baj az, hogy a modell alapfeltevései megegyeznek a séma tételével, és sajnos ezek a feltevések a gyakorlatban nem teljesülnek. A konkrét alkalmazásoknál a rátermettség legtöbb esetben időben nem változó, statikus. Az elsődleges szempont a hatékony implementáció, és a rátermettség-kiértékelések számának a függvényében a minél gyorsabb optimalizáció, nem pedig dinamikus, komplex adaptív rendszerek elméleti igényű modellezése. Éppen ezért a populáció nem elég nagy, így az egyes sémák nagy hibával vagy egyáltalán nem is reprezentálódnak. Ha reprezentálódnak is, akkor a populációban megtalálható séma előfordulások általában nem képviselik a séma valódi átlagos rátermettségét, mert vagy az egész keresési tér fölött a séma rátermettsége nagy szórást mutat vagy túl kevés az előfordulás, és kis szórás esetén is komoly eltérés mutatkozhat. A populáció méretének a növelése ugyan javítja a teljesítményt, de óriási költségnövekedéssel jár, így praktikus szempontból ez sem oldja meg a problémát. Párhuzamos implementáció is alkalmazható lenne, azonban a jelenleg elterjedt, üzenetek segítségével kommunikáló processzorokból álló párhuzamos architektúrákon a párhuzamosítás nem elég hatékony. Mindent összevetve az elmélet gyakorlati implikációi csekélynek mondhatók.

1.1.7.2 Újabb megközelítések

A fentiek miatt mostanában más szempontok is szerepet játszanak a genetikus algoritmusok elméleti tárgyalásaiban. Az egyik érdekes szemléletmód az operátorok szerepét az új megoldások előállításában és nem a populációban szétszórt különböző részek összekombinálásában látja. Így egyrészt megszűnik az éles különbség a keresztezés és a mutáció között: „rehabilitálódik" a mutáció, másrészt az evolúciós módszerek és a metaheurisztikák is egységes keretben tárgyalhatók. A fő eszköz a keresési tér elemei közötti távolságdefiníció, amelynek az alapjai a kódolás és az operátorok. Durván szólva, ha egy megoldásból az operátorok „könnyen", azaz nagy valószínűséggel elő tudnak állítani egy másik megoldást, akkor ez a két megoldás „közel" van egymáshoz, egyébként „távol". Konkrét kódolásra és operátorokra pontosan definiálható ilyen távolság. Így a tér struktúrát kap, amit **rátermettség-tájkép**nek (*fitness landscape*) neveznek. Az elnevezést az indokolja, hogy ha a keresési tér kétdimenziós valós intervallum, akkor a rátermettséget ábrázolva gyakran egy tájképszerű ábrát kapunk dombokkal, völgyekkel, márpedig a távolság definíció segítségével tetszőleges keresési téren beszélhetünk ilyen „domborzati viszonyokról". (A hasonlat nem biztos hogy szerencsés, mert az absztrakt sokdimenziós terek gyakran egészen sajátosan, a józan észnek ellentmondóan viselkednek, de ennek a kifejtése nem feladatunk.)

A rátermettség-tájkép egyik alkalmazása a távolság-rátermettség korreláció (*fitness distance correlation*, FDC) számítás (Jones, Forrest 1995). Itt egyszerűen a keresési térből vett nagyszámú minta alapján megvizsgáljuk, hogy van-e korreláció, és ha igen, milyen korreláció van a minta elemeinek a globális maximum helytől való távolsága, és a rátermettségük és a globális maximum különbsége között. A hipotézis az, hogy a feladat pontosan akkor könnyű a genetikus algoritmus számára, ha pozitív korreláció figyelhető meg, tehát a rátermettség függvény „szépen", közel monoton nő a globális maximum felé haladva, az operátorok által definiált távolság fogalmaiban persze. A módszer még viszonylag új, de az eddig vizsgált problémák esetében, ha a megfelelő távolságfüggvénnyel alkalmazták, helyesen jósolta meg a genetikus algoritmus teljesítményét.

A **tenyésztő genetikus algoritmus** (*breeder genetic algorithm*, BGA) elmélete az állattenyésztők által is használt kvantitatív genetikai statisztikai modellre épül (Mühlenbein et al. 1994). Maga az algoritmus egyébként a neve ellenére „sima" genetikus algoritmus, amely a **levágó szelekció**t (*truncation selection*) alkalmazza: a populáció felső valahány (ált. 10-50) százalékából véletlenszerűen választ szülőt.

Említést érdemelnek még a tisztán elméleti jelentőségű egzakt modellek, amelyek Markov láncokat használva adják meg a populációban ez egyes megoldások előfordulási valószínűségét (Vose 1998). Ezen kívül érdekesek még az operátorok adaptív módosításának a hatásait, lehetőségeit vizsgáló kutatások, amelyek konkrét gyakorlati jelentőséggel bírnak (Bäck 1996).

1.1.8 Miért genetikus az algoritmus?

Ahogy az olvasó mostanára már tapasztalhatta, a genetikus algoritmusok elméletében és gyakorlatában használt fogalmak majdnem mind a biológiából származnak. Természetes a kérdés, hogy a névadók játékosságán kívül mi indokolja ezt, hiszen sok fogalomra bevett matematikai elnevezés is használható volna.

Először ismételjük át a fontosabb biológiai ihletésű elnevezéseket. Egy megoldani kívánt probléma lehetséges megoldásainak egy gyűjteménye a populáció (nem halmaza, mert az ismétlődés megengedett). A megoldásokhoz genotípust rendelünk, amelyből egy dekódoló algoritmus állítja vissza a fenotípust, azaz magát a megoldást. A genotípus rendszerint valamilyen betűsor, amelynek pozíciói (indexei) a **gének**, maguk a betűk az **allélok**. A megoldások populációjára szelekciót alkalmazunk, és a szelektált megoldások genotípusain operátorokat alkalmazunk, mint pl. a keresztezés vagy a mutáció.

A szakirodalomban a genetikus algoritmust a természetes szelekción alapuló, a darwini elméletből merítő módszerként kezelik. Amellett fogunk érvelni, hogy a genetikus 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. A természetes szelekció gondolatának egyik inspirálója éppen a mesterséges szelekció volt (Darwin 1859); úgy tűnik, a genetikus algoritmus, mint ötlet, nem a darwini gondolatokból, hanem azokkal közös tőről fakad. Nem állítjuk azonban azt, hogy az algoritmust nem is lehet az evolúció modellezésére használni, mint ahogy azt sem, hogy nem léteznek olyan feladatok és implementációk, amelyek közelebb állnak a természetes evolúcióhoz. A megjegyzéseink a genetikus algoritmusok használatának általános gyakorlatára vonatkoznak

A fajok evolúciója és a genetikus algoritmus közti fő különbség, hogy míg az előbbinél a legfontosabb szempont az adaptációra való képesség, azaz a valamilyen szempontból folyton változó, bonyolult környezethez való alkalmazkodás, addig az utóbbinál a lényeg az optimalizáció, azaz egy rögzített szempontból a legjobb megoldás megtalálása. A rátermettség fogalmának a használata is zavarba ejtő lehet egy biológus számára, hiszen az evolúció elméleteiben ez a tényleges utódok számának a függvénye, tehát valamilyen *megfigyelt* érték. Ez azért van így, mert a környezet és a populáció dinamikájának és komplexitásának köszönhetően nagyon nehéz megjósolni azt, hogy egy adott egyed vagy annak közeli rokonai hány utódot fognak létrehozni. A genetikus algoritmusoknál az utódok száma egyrészt egy *előírt* értéktől függ, amit (elég helytelenül) rátermettségnek neveznek, másrészt attól, hogy ez az érték mekkora a populációban a többi egyed esetében. A rátermettségnek nevezett érték a legtöbb esetben időben nem változik, és az utódok száma semmilyen bonyolultabb módon nem függ a populáció elemeinek tulajdonságaitól.

Érdekes ebbe a megvilágításba helyezni az ivaros szaporodás (azaz a több szülőt igénylő rekombinációs operátorok, pl. keresztezés) és a populáció szerepét. Világos, hogy adaptív szempontból a populáció kulcsszerepet játszik: az ivaros szaporodással karöltve biztosítja, hogy a gének különböző alléljai megmaradjanak, mint egyfajta memóriában, és a különböző allélok előfordulási gyakorisága a környezet változásával egyensúlyt tarthat. Nem világos azonban, még az elméleti szakemberek erőfeszítéseinek a figyelembevételével sem, hogy a

fenti vonásoknak mi a szerepük a genetikus algoritmus működésében; egyáltalán értelmes-e időben nem változó optimalizálási problémák esetén a számítástechnikai szempontból költséges populáció és az ivaros operátorok fenntartása. Jónéhány munka számol be arról, hogy populáció nélkül (azaz egy elemű populációval) és ivartalan operátorokkal (azaz egy szülőt igénylő operátorokkal, pl. mutáció) jobb eredményt lehet elérni, mint ezek alkalmazásával (Juels, Wattenberg 1994), (Yaguira, Ibaraki 1996). Ahogy korábban említettük, az egyelemű populációt ivartalan operátorokkal alkalmazó algoritmusokat szokás sztochasztikus hegymászóként emlegetni.

1.1.9 Összefoglalás

A genetikus algoritmus egy olyan problémafüggetlen metaheurisztika, amely általános keresési terekben végez kevés tudást igénylő optimalizálást. Maga a módszer nagyon hasonlít az állattenyésztők módszerére. Megoldás populációk sorozatát állítja elő különböző operátorok segítségével, amelyek közül a legfontosabbak a rátermett szülők kiválasztását végző szelekció, és a szülőkből új megoldásokat előállító mutáció és rekombináció. A genetikus algoritmus alkalmazásához szükség van keresési teret adó megoldások kódolására, ahol a kód analóg az örökítőanyaggal. A kódolás és az operátorok kiválasztása lehetőséget ad terület specifikus ismeretek indirekt használatára.

Sok alkalmazási területe ismert az algoritmus problémafüggetlen természetéből adódóan. Tipikusak a kombinatórikus, NP-teljes problémák, mint amilyen a gráfszínezés, és a gépi tanulás területéről származó alkalmazások mint pl. számítógép programok automatikus fejlesztése (genetikus programozás) és a mesterséges neurális hálózatok evolúciója.

Végül néhány, általánosan elfogadott, főleg intuitív alapokon álló elv arra vonatkozóan, hogy mikor érdemes a genetikus algoritmus alkalmazásával kísérletezni: ha a probléma keresési tere kezelhetetlenül nagy, nem ismert a struktúrája, nem áll rendelkezésre terület specifikus tudás, nem ismert egzakt, gyors algoritmus, nincs szükség a pontos globális optimumra, csak egy stabil, jó közelítés kell, ill. ha a kiértékelő függvény (rátermettség) pontatlan, zajos.

1.2 Irodalomjegyzék

(Bäck 1996) Thomas Bäck. *Evolutionary algorithms in theory and practice*. Oxford University Press, New York, 1996.

(Bäck 1997) Thomas Bäck, editor. *Proceedings of the Seventh International Conference on Genetic Algorithms*, San Francisco, California, 1997. Morgan Kaufmann.

(Bäck et al.1991) Thomas Bäck, F. Hoffmeister, and H. Schwefel. A survey of evolution strategies. In Belew and Booker editors. *The Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1991.

(Belew, Vose 1997) R. K. Belew and M. D. Vose, editors. *Foundations of Genetic Algorithms IV*. Morgan Kaufmann, 1997.

(Darwin 1859) C. R. Darwin. *The Origin of Species*. Dent, London, 1859.

(Davidor et al. 1994) Y. Davidor, Hans-Paul Schwefel, and R. Männer, editors. *Parallel Problem Solving from Nature 3*, volume 866 of Lecture Notes in Computational Science. Springer-Verlag, 1994.

(Ebeling et al. 1996) W. Ebeling, I. Rechenberg, H.-P. Schwefel, and H.-M. Voigt, editors. *Parallel Problem Solving from Nature - PPSN IV*, volume 1141 of Lecture Notes in Computational Science. Springer-Verlag, 1996.

(Eiben, Hauw 1998) A. E. Eiben and J. K. van der Hauw. Graph coloring with adaptive genetic algorithms. *Journal of Heuristics*, 4(1), 1998.

(Eshelman 1995) L. J. Eshelman, editor. *Proceedings of the Sixth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1995.

(Fogel 1995) David B. Fogel. *Evolutionary Computation*. IEEE Press, 1995.

(Fogel et al. 1966) L. J. Fogel, A. J. Owens, and M. J. Whals. *Artificial Intelligence through Simulated Evolution*. Wiley, 1966.
(Forrest 1993) S. Forrest, editor. *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1993.
(Goldberg 1989) David E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, 1989.
(Goldberg 1991) David E. Goldberg. A comparative analisys of selection schemes used in genetic algorithms. In Rawlins , editor. *Foundations of Genetic Algorithms*. Morgan Kaufmann, 1991.
(Holland 1975) John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.
(Jones, Forrest 1995) T. Jones and S. Forrest. Fitness Distance Correlation as a Measure of Problem Difficulty in Genetic Algorithms. In (Eshelman 1995).
(Juels, Wattenberg 1994) A. Juels and M. Wattenberg. Stochastic hillclimbing as a baseline method for evaluating genetic algorithms. Technical report, UC Berkeley, 1994.
(Koza 1992) John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.
(Koza 1994) John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, 1994.
(Männer, Manderick 1992) R. Männer and B. Manderick, editors. *Parallel Problem Solving from Nature 2*. Elsevier Science Publishers/North Holland (Amsterdam), 1992.
(Michalewicz 1996) Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 3rd edition, 1996.
(Mitchel 1996) Melanie Mitchell. *An Intruduction to Genetic Algorithms*. MIT Press, 1996.
(Mitchell et al. 1994) M. Mitchell, J. H. Holland, and S. Forrest. When will a genetic algorithm outperform hillclimbing? In J. D. Cowan et al., editors, *Advances in Neural Information Processing Systems 6*. Morgan Kaufmann, 1994.
(Mühlenbein et al 1994) Heinz Mühlenbein and Dirk Schlierkamp-Voosen. Theory and application of the breeder genetic algorithm. In Jacek M. Zurada, Robert J. Marks, and Charles J. Robinson, editors, *Computational Intelligence Imitating Life*. IEEE Press, 1994.
(Radcliff 1994) N. J. Radcliff. The algebra of genetic algorithms. *Annals of Maths and Artificial Intelligence*, 1994.
(Rechenberg 1973) I. Rechenberg. *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologische Evolution*. Fromman-Holzboog, Stuttgart, 1973.
(Schwefel 1995) Hans-Paul Schwefel. *Evolution and Optimum Seeking*. Wiley, 1995.
(Vose 1998) Michael D. Vose. *The simple genetic algorithm: foundations and theory*. MIT Press, 1998.
(Whitely 1993) L. D. Whitley, editor. *Foundations of Genetic Algorithms II*. Morgan Kaufmann, 1993.
(Whitley, Vose 1995) L. D. Whitley and M. D. Vose, editors. *Foundations of Genetic Algorithms III*. Morgan Kaufmann, 1995.
(Yaguira, Ibaraki 1996) Mutsunori Yagiura and Toshihide Ibaraki. Genetic and local search algorithms as robust and simple optimization tools. In Osman and Kelly editors. *Meta-Heuristics: Theory and Application*. pages 63—82, Kluwer Academic Publishers,.1996
(Yao 1993) Xin Yao. A review of evolutionary artificial neural networks. *International Journal of Intelligent Systems*, 8(4):539--567, 1993.