Derivációs fa alapú genetikus programozás

Doktori értekezés tézisei

szerző:

Ványi Róbert

témavezető:

apl. Prof. Dr.-Ing. Kókai Gabriella

Informatika Doktori Iskola

Természettudományi és Informatikai Kar

Szegedi Tudományegyetem

Szeged, 2012

1. Bevezetés

Ez a tézisfüzet a *Derivációs fa alapú genetikus programozás* című PhD dolgozat eredményeit foglalja össze. A dolgozat a címben megjelölt, angolul ,,derivation tree based genetic programming" (röviden DTGP) néven bemutatott eljárást mutatja be.

Az összefoglaló struktúrája a dolgozat felépítését követi. A problématerület bemutatása után az evolúciós algoritmusok és a formális nyelvek releváns témaköreinek összefoglalása következik. A 4. fejezet a DTGP módszert írja le, míg az 5. fejezet az alap rendszer kiegészítéseit foglalja össze. A 6. fejezet áttekintést nyújt a DTGP alkalmazásának eredményeiről, a 7. fejezet pedig a dolgozat téziseit ismerteti.

1.1. Problématerület

A számítástudomány egy gyakori alkalmazási területe az optimalizálási problémák megoldása. Ide tartozik a minimalizálás, maximalizálás vagy általánosságban a *legjobb* megoldás megtalálása. Megkülönböztetünk *determinisztikus* és *sztochasztikus* módszereket. Bár az előbbieket általában könnyebb vizsgálni, ezek az időigényük miatt sokszor nem praktikusak. A sztochasztikus algoritmusok ezzel szemben egy bizonyos mértékű bizonytalanságot hordoznak magukban, de a gyakorlatban alkalmazott időkorlátok mellett sok esetben jobb eredményt érnek el.

Az *evolúciós algoritmusok* metaheurisztikus optimalizálási algoritmusok, melyek a *természetes szelekciót* modellezik és így hoznak létre, vizsgálnak és választanak ki *egyednek* nevezett megoldáskezdeményeket. [3] Többféle típusuk is ismert, mint az *evolúciós stratégiák*, a *genetikus algoritmusok* és a *genetikus programozás*. Ezeket különféle problémák megoldására alkalmazták a mérnöki tervezési optimalizálástól kezdve a számítógépes programok automatizált előállításáig.

A *fekete doboz paradigma* azt jelenti, hogy az optimalizálási algoritmusnak egyáltalán semmi, vagy csak nagyon kevés információja van a megoldások struktúrájáról vagy magáról a megoldástérről. Ez azt is jelenti, hogy sokszor nem garantálható, hogy az új egyedek a problémának egyáltalán megoldásai, ami rossz hatással van a keresés minőségére. Az érvénytelen egyedek csökkentik a populáció effektív méretét és a detektálásuk, kiértékelésük illetve esetleges javításuk többlet számításigényt jelent.

Egy módszer az evolúciós keresés szabályozására az evolúciós folyamatot irányító *formális nyelvtanok* alkalmazása. Ha a megoldások, illetve azok reprezentációi *formális nyelvet* alkotnak, vagyis a halmazuk leírható egy nyelvtannal, a keresés korlátozható a nyelvtan szabályait követő egyedekre. Több *nyelvtan vezérelt genetikus programozási* (röviden GGGP) módszert is definiáltak, közülük sok *környezetfüggetlen nyelvtant* használ a folyamat irányítására és *derivációkat* a genotípus reprezentálására. Megoldáskezdemények helyett derivációkon dolgozva garantálhatjuk az előállított egyedek érvényességét. [6]

1.2. A javasolt módszer

A dolgozatban bemutatott módszer *derivációs fákat* használ reprezentációként, és úgy definiálja az evolúciós operátorokat, hogy az egyedek mindig érvényes derivációs fák legyenek. Az eddigi GGGP eljárásokhoz képest előrelépést jelent a paraméterek széleskörű alkalmazása. Paramétereket a fa minden csúcsában tárolunk, és ezeket különböző célokra használjuk fel, mint például a véletlen csúcskiválasztás kiegyensúlyozottságának garantálására, a kiértékelés időigényének lineárisról logaritmikusra való csökkentésére, a keresés befolyásolására illetve szemantikai megkötések felállítására.

1.3. Publikációk

A dolgozatban bemutatott eredmények öt publikáción alapulnak, melyek időrendben a következők:

[11] R. Ványi and Sz. Zvada *Avoiding syntactically incorrect individuals via parameterized operators applied on derivation trees.* In R. Sharker, et al., editors, Proceedings of the 2003 Congress on Evolutionary Computation CEC2003, volume 4, pages 2791–2798, Canberra, 8-12 Dec 2003. IEEE Press.

[16] Sz. Zvada and R. Ványi *Improving grammar-based evolutionary algorithms via attributed derivation trees.* In M. Keijzer, et al., editors, Genetic Programming 7th European Conference, EuroGP 2004, Proceedings, volume 3003 of LNCS, pages 208–219, Coimbra, Portugal, 5-7 Apr 2004. Springer-Verlag.

[12] R. Ványi and Sz. Zvada *Syntactically correct genetic programming.* In R. Poli et al., editors, GECCO 2004 Workshop Proceedings, Seattle, Washington, USA, 26-30 Jun 2004.

[15] Sz. Zvada, G. Kókai, R. Ványi, and H.H. Frühauf *EvolFIR: Evolving redundancy-free FIR structures.* In Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007), pages 439–446. IEEE Computer Society, 5-8 Aug 2007.

[10] R. Ványi *Enforcing semantic constraints with derivation tree based genetic programming.* Abstract accepted to oral presentation at Veszprém Optimization Conference: Advanced Algorithms (VOCAL 2012), 11-14 Dec 2012.

A [11] bemutatja a DTGP módszert, az alapvető fa operátorokat és a véletlen fagenerálást. A [16] a *kiegyensúlyozott véletlen csúcs kiválasztást* építi fel és néhány paraméter alkalmazásának lehetőségét tekinti át. A [12] a DTGP-t más GGGP módszerekkel hasonlítja össze, bemutatja a *készlet keresztezést* illetve megvizsgálja az operátorok időigényét. A [15] egy valós DTGP alkalmazást mutat be, amit a Fraunhofer Alapítvány Integrált Áramkörök Intézetével közösen fejlesztettünk, és felvázolja a *szemantikai megkötések* alkalmazására tett első kísérleteket. A *szemantikailag korlátozott derivació* első formalizálását, beleértve a *disztribúciós halmazokat*, *disztribúciós függvényeket* és a *megkötött szintetizált attribútumokat*, a [10] mutatja be.

1.4. Magyar kifejezések

A dolgozat angol nyelven készült, de ebben az összefoglalóban legtöbbször az eredeti angol kifejezések magyar megfelelői szerepelnek.

Az evolúciós algoritmusok operátorait *mutációnak*, *rekombinációnak*, illetve *keresztezésnek* nevezzük, és ezek a *populációk egyedeit* módosítják. A dolgozatban definiált *készlet keresztezés* eredeti angol elnevezése pool crossover. A distribution set és distribution function kifejezéseket magyarra *disztribúciós halmaz*, illetve *disztribúciós függvény* néven fordítottuk. A forced synthesized attribute magyar megfelelője a *megkötött szintetizált attribútum.*

2. Evolúciós algoritmusok

Az *evolúciós algoritmusok* (EA) egy speciális módszert használnak az optimális megoldás megtalálására, ami a természetben megfigyelhető evolúción alapul. Evolúciós algoritmusok használatakor a keresési eljárás *párhuzamos*, vagyis több hipotézis kivizsgálása folyik egyidőben. A hipotéziseket *egyedeknek* nevezzük, és ezek alkotják a *populációt*. A kezdeti populációt véletlenszerűen hozzuk létre, majd minden hipotézist kiértékelünk a *fitness függvény* segítségével, amely megmutatja, hogy az adott hipotézis mennyire jó. Ezután a *szelekció* operátorral kiválasztunk egyedeket a *fitness érték* alapján, ezek lesznek a *szülők*. Végül az evolúciós operátorokat, mint a *mutációt* vagy a *rekombinációt*, a szülők halmazára alkalmazva a *leszármazottak* egy új populációját hozzuk létre, és a folyamatot az új populációval újrakezdjük.

A folyamat közben a populációban (amit az adott lépés végén *generációnak* nevezünk) egyre jobb egyedek jelennek meg. A folyamatot akkor állítjuk le, amikor az úgynevezett *megállási feltételt* elértük. Gyakori megállási feltétel a lépések száma, csekély változás a legjobb fitness értékben, vagy az optimum elegendően pontos megközelítése.

Az evolúciós algoritmusoknak több fajtája van. [3] A legismertebbek az *evolúciós stratégiák* (ES) és a *genetikus algoritmusok* (GA). Egy harmadik típus a *genetikus programozás* (GP), amelyet a genetikus algoritmusokból fejlesztettek ki.

2.1. Komplex struktúrák optimalizálása

A dolgozatban egy példán keresztül mutattuk be, hogyan lehet a genetikus algoritmusokat komplex struktúrák optimalizálására alkalmazni. A cél egy logikai kifejezés megtalálása volt, amely lehetőleg rövid, és az előre definiált logikai függvényt írja le. A kifejezéseket sztringekkel reprezentáltuk, egy változókat, negált változókat, konjunkció és diszjunkció operátorszimbólumokat illetve zárójeleket tartalmazó ábécé felett.

Az eredmények azt mutatták, hogy az algoritmus sok szintaktikailag helytelen egyedet állít elő, ami meggátolja a megoldás megtalálását. Lehetőség van az operátorok korlátozására, és így a helyes megoldások megtalálhatóak, de a teszteredmények szerint az érvénytelen egyedek száma továbbra is magas, a sikerességi ráta pedig alacsony marad.

A genetikus programozást arra tervezték, hogy absztrakt szintakszisfákat optimalizáljon, elsősorban olyanokat, amelyek a LISP programozási nyelv S-kifejezéseit reprezentálják. GP-t használva szintaktikailag helyes egyedek jönnek létre, de csak bizonyos korlátok mellett: a hagyományos GP megköveteli a *zártsági tulajdonságot*. [5, 9] Egyrészt ehhez *típuskonzisztencia* szükséges, ami azt jelenti, hogy minden argumentumnak és visszatérési értéknek ugyanolyan típusúnak kell lennie. Ez azért szükséges, mert az evolúciós operátorok tetszőlegesen cserélhetnek ki részfákat. Másrészt a zártsági tulajdonság magában foglalja a *kiértékelési biztonságot*, ami azt jelenti, hogy minden lehetséges részfa által reprezentált kifejezés kiértékelhető, és így minden fához fitness érték rendelhető. Tehát a hagyományos GP nem szolgáltat általános megoldást az optimalizálási folyamat szintaktikailag helyes egyedekre való korlátozására.

Azonban a GP átalakítható, hogy formális nyelvtanokat használjon az evolúciós folyamat során a szintaktikai helyesség biztosítására. Ezeket az eljárásokat nyelvtan vezérelt genetikus programozási (GGGP) algoritmusoknak nevezzük. [6] A dolgozatban javasolt módszer szintén egy GGGP eljárás, amely környezetfüggetlen nyelvtanokat használ, és ezen nyelvtanok néhány fontos tulajdonságára támaszkodik.

3. Nyelvtanok és formális nyelvek

Egy ábécé szimbólumaiból nyelvek felépítésére legtöbbet használt eszközök a Chomsky által definiált *formális nyelvtanok*. [2] A dolgozatban a *környezetfüggetlen* nyelvtanoknak kiemelt szerepük van. Egy *környezetfüggetlen nyelvtan* egy $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$ rendezett négyes, ahol $\mathcal{N}$ a *nemterminális ábécé*, Σ a *terminális ábécé* úgy, hogy $\mathcal{N} \cap \Sigma = \emptyset$, $\mathcal{P} \subset \mathcal{N} \times (\mathcal{N} \cup \Sigma)^*$ *átírási szabályok* egy véges halmaza, és $S \in \mathcal{N}$ a *kezdő szimbólum*.

Nyelvtanokkal nyelveket a levezetések (derivációk) segítségével generálhatunk. A kezdő szimbólumot egy megfelelő átírási szabállyal lecseréljük. Ezután az új szóban lehetnek nemterminális szimbólumok, melyeket újabb átírási szabályok alkalmazásával cserélünk ki. Az átírási szabályokat addig alkalmazzuk, amíg egy csak terminális szimbólumokból álló szót kapunk. Mivel általában több átírási szabály alkalmazható, egy adott nyelvtannal több szó is generálható. A legtöbb GGGP módszernél a derivációfogalma kifejezetten fontos, ezért itt megadjuk a formális definícióját is.

3.1. definíció. *(Deriváció)*
Az $(\mathcal{N} \cup \Sigma)^*$ halmaz felett bevezetünk egy $\Rightarrow_G$-vel jelölt bináris relációt, amit közvetlen deriválciónak nevezünk. Tetszőleges $\gamma, \delta \in (\mathcal{N} \cup \Sigma)^*$-ra $\gamma \Rightarrow_G \delta$ akkor és csak akkor áll fenn, ha $\exists \varphi, \psi \in (\mathcal{N} \cup \Sigma)^*$ és egy $A \to \beta$ szabály $\mathcal{P}$-ben, amelyre teljesül, hogy $\gamma = \varphi A \psi$ és $\delta = \varphi \beta \psi$. A deriváció a közvetlen deriváció tranzitív, reflexív lezártja, amit $\Rightarrow_G^*$-gal jelölünk.

A közvetlen deriváció segítségével definiálható a *derivációs szekvencia*, ami szavak egy véges $\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_n$ sorozata, melyre $\alpha_i \Rightarrow_G \alpha_{i+1}$. A derivációs szekvencia hossza a közvetlen derivációs lépések száma, azaz n.

A deriváció segítségével definiálható a generált nyelv. A G nyelvtan által generált nyelv a következő: $L(G) = \{u \in \Sigma^* \mid S \Rightarrow_G^* u\}$.

3.1. Derivációs fák

A környezetfüggetlen nyelvtanok feletti derivációkat egyszerűen reprezentálhatjuk. Tekintsük a levezetett szavak betűit csúcsoknak. Amikor egy nemterminális szimbólumot kicserélünk egy szóra, vagyis szimbólumok egy sorozatára, ezeket a szimbólumokat úgy ábrázolhatjuk, mint az eredeti szimbólum csúcsának gyerekeit egy fa struktúrában, amit *derivációs fának* nevezünk. A tényleges levezetett szó a fa *határában* található, ami a fa leveleinek sorozata balról jobbra haladva.

Fontos megjegyezni, hogy egy szó akkor és csak akkor vezethető le egy szimbólumból, ha létezik olyan derivációs fa, amelynek a gyökere az adott szimbólum, a határa pedig az adott szó. Ez a megállapítás különösen fontos a kezdőszimbólumra és a terminális szavakra, ahogyan a következő tétel kimondja.

3.2. tétel.
Egy $u \in \Sigma^*$ szóra $S \Rightarrow^* u$ akkor, és csak akkor, ha létezik $T \in \mathcal{T}(S)$, úgy hogy $fr(T) = u$, ahol $\mathcal{T}(S)$ az S gyökerű derivációs fák halmaza.

3.3. következmény.
Egy $u \in \Sigma^*$ szóra $u \in L(G)$ akkor, és csak akkor, ha létezik $T \in \mathcal{T}(S)$, úgy hogy $fr(T) = u$.

Ez a következmény azt jelenti, hogy ha derivációs fákkal dolgozva biztosítjuk, hogy az eredmények is érvényes derivációs fák legyenek, akkor a fák határaiban található szavak mindig érvényes megoldások lesznek.

3.1.1. Derivációs fák mérete

A dolgozatban adattípusként derivációs fákat alkalmazunk, ezért érdemes megvizsgálni a levezetett szavak, a derivációk és a derivációs fák mérete közötti kapcsolatot. Ha a terminális szimbólumok számát az r átírási szabály jobb oldalán $|r|_\Sigma$ jelöli, akkor a korrelációt a következő tétellel írhatjuk le.

3.4. tétel. *(A derivációhoz kapcsolódó méretek korrelációja)*
Adott egy $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$ környezetfüggetlen nyelvtan. Egy tetszőleges $\alpha_0, \alpha_1, \ldots, \alpha_k$, $\alpha_k \in \Sigma^*$ derivációs szekvenciára jelölje a hozzá tartozó derivációs fa méretét s, a levezetett terminális szó α_k hosszát pedig n. Ebben az esetben $s = \mathcal{O}(k)$. Továbbá, ha $|r|_\Sigma$ az összes alkalmazott r_i $(1 \leq i \leq k)$ szabályra átlagolva legalább 1, akkor $k \leq n$.

3.2. Attribútum nyelvtanok

Az attribútum nyelvtanok azzal terjesztik ki a környezetfüggetlen nyelvtanok koncepcióját, hogy a szimbólumokat attribútumokkal látják el és a derivációk során ezekhez értékeket rendelnek. [1]

Egy *attribútum nyelvtan* egy $AG = (G, SD, AD, \mathcal{R})$ rendezett négyes, ahol G egy környezetfüggetlen nyelvtan, SD a *szemantikai tartomány*, amely az attribútum típusokat, függvényeket és relációkat definiálja, AD tartalmazza az *attribútum leírásokat*, $\mathcal{R} = \{\mathcal{R}(p) \mid p \in \mathcal{P}\}$ pedig halmazok egy családja, amely definiálja a *szemantikai szabályokat* a G nyelvtan minden átírási szabályára.

A dolgozatban a következő attribútumokkal kapcsolatos feltételezéseket alkalmazzuk. Egy a attribútum *örökölt*, ha minden $X \to Y_1 Y_2 \ldots Y_n$ szabályra a kalkulációs séma a következő:

$$Y_i.a = f_a(X.a, Y_1.a, \ldots, Y_{i-1}.a),$$

illetve *szintetizált*, ha a kalkulációs séma a következő:

$$X.a = g_a(Y_1.a, Y_2.a, \ldots, Y_n.a).$$

Ennél formálisabb definícióra nincs szükségünk, de az olvasó fellelheti őket az irodalomban. Az örökölt attribútumokat más néven *fentről-lefelé*, míg a szintetizált attribútumokat *lentről-felfelé* attribútumoknak nevezzük.

4. Derivációs fa alapú genetikus programozás

Az érvénytelen egyedek kezelésének általános módja az, hogy a kiértékelési függvény kiszűri őket például úgy, hogy nagyon alacsony fitness értékeket rendel hozzájuk. Egy másik lehetőség, hogy egyáltalán nem engedünk meg ilyen egyedeket. Ehhez a reprezentáció vagy az evolúciós operátorok módosítására van szükség. Ezt három különböző módon érhetjük el.

Minden hipotézis engedélyezése A legegyszerűbb mód, hogy minden hipotézist elfogadunk megoldásként. Ehhez nem szükséges az evolúciós algoritmust megváltoztatni. Erre a megközelítésre egy példa a hagyományos GP, ahogyan Koza definiálta. [5]

Operátorok korlátozása Egy kézenfekvő módszer az operátorok módosítása, bár ez nem mindig egyszerű feladat. Így működik az *erősen típusos genetikus programozás*. [7]

Reprezentáció újradefiniálása A harmadik lehetőség a reprezentációk halmazának olyan definiálása, hogy zárt legyen az evolúciós operátorokra nézve. Ehhez szükségesek lehetnek apróbb változtatások az operátorokban. Ezt a megközelítést követi a DTGP.

4.1. Nyelvtan vezérelt genetikus programozás

A *nyelvtan vezérelt genetikus programozás* (*GGGP*) [6] egyre nagyobb népszerűségnek örvend. A GGGP eljárások feltételezik, hogy az érvényes egyedek halmaza egy környezetfüggetlen nyelv, és ezért környezetfüggetlen nyelvtanokat használnak az evolúciós folyamat irányítására. A reprezentációtól függően két típust különböztetünk meg.

4.1.1. Fa alapú GGGP

Az első fa alapú GGGP kísérleteket Gruau [4] végezte, aki környezetfüggetlen nyelvtanokat és derivációs fákat használt az egyedek helyességének ellenőrzésére. Az ellenőrzés után azonban a derivációs fákat törölte, majd az új egyedekre újra előállította. Whigham [13] a populáció egyedeit derivációs fákkal reprezentálta, és az operátorokat közvetlenül ezekre alkalmazta. Az ő módszerének korlátja, hogy az operátorok nem paraméterezhetőek, így csak globális változókkal lehet őket befolyásolni.

4.1.2. Lineáris GGGP

A legnépszerűbb lineáris GGGP módszer a *nyelvtani evolúció* (*GE*) [8], ami bitvektorokat használ az egyedek reprezentálására. Ez lehetővé teszi a genetikus algoritmusok területén megismert operátorok és módszerek alkalmazását.

A hipotézisek egy Σ ábécé feletti szavak, és a megoldások halmazát egy G nyelvtan írja le. Az érvényes reprezentációk, vagyis az $L(G)$ nyelv szavai leírhatóak baloldali levezetésekkel, amelyek viszont binárisan tárolt indexvektorokkal reprezentálhatóak. A nyelvtani evolúció ezeket az indexvektorokat használja egyedekként.

Azonban ennek a módszernek vannak hátrányai. Először is, mivel csak a szabályok indexeinek sorozata kerül eltárolásra, a derivációkat el kell végezni és a derivációs fákat, vagy legalább azok határát ki kell számítani ahhoz, hogy megkapjuk a tényleges megoldást. Ez jelentős időt vehet igénybe. Továbbá nincs kölcsönösen egyértelmű leképezés a bitsztringek és a derivációk között, ami befejezetlen vagy végtelen derivációkhoz vezethet.

4.2. A DTGP módszer alapjai

A derivációs fa alapú genetikus programozás egy fa alapú GGGP módszer. A reprezentációk derivációs fák egy G környezetfüggetlen nyelvtan felett. A megoldások, vagyis az $L(G)$ nyelv szavai, ezen fák határában találhatóak. Az evolúciós operátorokat úgy definiáljuk, hogy csak érvényes derivációs fákat állítsanak elő. Ennek következtében az evolúciós folyamat csak érvényes hipotéziseket generálhat. A módszer alapjai nagyon hasonlóak Whigham [13] eljárásához, viszont az adattípus az algoritmus javítására használt paramétereket is tartalmaz.

4.2.1. Derivációs fa adattípus

A derivációs fa gyakorlatilag egy hagyományos fa adattípus, általában pointer alapú implementációval. Viszont a DTGP nem csak a címkéket tárolja el minden csúcsban, hanem paraméterek formájában további információt is. Ezen paramétereknek köszönhetően az eljárás több előnnyel is rendelkezik más fa alapú GGGP módszerekkel szemben, ahogy ez a dolgozatban bemutatásra is került.

A dolgozatban használt jelöléseket az 1-es táblázat mutatja be. Meg kell jegyeznünk, hogy a csúcs és a fa jelölések sok esetben felcserélhetőek. Például a fa méretét jelölheti $T.size$, de ezt általában paraméterként tároljuk el a fa N gyökerében, ezért $N.size$-zal is jelöljük.

szimbólum	**jelentés**	**megjegyzés**
$T, T_1, T_2, \ldots$	fa, részfa	$T_1, T_2, \ldots$ a T egy részfáját jelöli
$N, N_1, N_2, \ldots$	csúcs	$N_1, N_2, \ldots$ az N gyerekei
$N[T_1, \ldots, T_n]$	fa	N gyökér $T_1, \ldots, T_n$ részfákkal
$N.label$	címke	N csúcs címkéje
$T.label$	címke	T fa gyökerének címkéje
$N.param$	paraméter	N csúcs paramétere
$T.param$	paraméter	T fa gyökerének paramétere

1. táblázat. A derivációs fa adattípus részeinek jelölése

4.2.2. Paraméterezett derivációs fák

Az evolúciós folyamat közben az operátorok a derivációs fák több tulajdonságát is felhasználhatják. Mivel ezeknek az ismételt kiszámítása nagy munkaigényű lehet, ezért a DTGP az értékeket paraméterként eltárolja a részfák gyökereiben.

A műveletek részfákat módosítanak, és az a célunk, hogy ezek a változtatások csak a csúcsok egy korlátozott halmazát érintsék, ezért megköveteljük, hogy egy fa tulajdonságai csak a részfáinak tulajdonságaitól függjenek. Ez azt jelenti, hogy ezeket a tulajdonságokat *lentről felfelé* definiáljuk, így minden p tulajdonságra és minden $T = N[T_1, \ldots, T_n]$ fára a kalkulációs séma $T.p = f_p(N, T_1, \ldots, T_n)$.

A paraméterek csúcsokban történő eltárolásának hátránya, hogy ezeket az evolúciós folyamat során aktualizálnunk kell. Viszont mivel a paramétereket *lentről felfelé* definiáljuk, könnyű belátni, hogy a derivációs fán belül egy részfa megváltoztatása csak a részfa gyökerétől a derivációs fa gyökerébe vezető úton lévő csúcsokra van hatással. Ezért a paramétereket egy egyszerű algoritmussal frissíthetjük, ami csak logaritmikus időigényű a fa méretére nézve. Az algoritmus a megváltoztatott részfa csúcsából indul ki és az őseit aktualizálja, amíg el nem éri a fa gyökerét.

4.3. Evolúciós operátorok

Az evolúciós operátorok a szokásos faoperátorokon alapulnak, bár szükség van néhány módosításra annak biztosításához, hogy csak érvényes derivációs fák jöhessenek létre. A DTGP-hez több különböző mutáció és kereszteződés operátort definiálhatunk. Ezek az operátorok két alapműveletet használnak: a *véletlen fagenerálást* és a *véletlen csúcskiválasztást*. Az operátorokat úgy terveztük meg, hogy figyelembe vegyék a környezetfüggetlen nyelvtan által definiált megkötéseket, de az evolúciós algoritmus bonyolultságát ne növeljék jelentősen.

4.3.1. Véletlen fagenerálás

A véletlen fagenerátor egy alapvető része a DTGP-nek, mert ez az egyetlen komponens, amelynek információja van a keresési térről, egy környezetfüggetlen nyelvtan formájában. Az alapötlet az, hogy egy adott nemterminális szimbólumból kiindulva vegyük az alkalmazható szabályokat, válasszunk egyet véletlenszerűen, alkalmazzuk, majd vegyük a nemterminális szimbólumokat a létrehozott fa határában. Az algoritmus addig folytatódik, amíg van nemterminális szimbólum a fa határában.

A fa méretének korlátozásához egy min_p nevű konstanst rendelünk minden p szabályhoz, hogy tudjuk, mekkora a legkisebb előállítható fa, amennyiben az adott szabállyal kezdjük a derivációt. A min_p konstans kiszámítása igényel valamennyi időt, de csak egyszer kell elvégezni a számítást, mégpedig az evolúciós folyamat elindítása előtt.

4.3.2. Véletlen csúcskiválasztás

Minden operátornak ki kell választania a derivációs fa egy vagy több csúcsát. A komplex adatstruktúra miatt ez nem triviális feladat, különösen azért, mert nem minden csúcs lehetséges jelölt a kiválasztásra. Egy vektorból véletlenszerűen kiválasztani egy elemet egyszerű, de az összes csúcs vektorba rendezése csak a kiválasztás miatt nem hatékony megoldás. Ezért a DTGP-hez egy, a keresési fa kiválasztási módszerén alapuló véletlen csúcskiválasztót definiáltunk, ami logaritmikus időigényű. Ez azt jelenti, hogy a fában bejárunk egy utat a gyökérből lefelé, miközben véletlenszerűen választunk az aktuális csúcs, illetve a csúcs gyerekei közül. Ha magát a csúcsot választottuk, a keresés befejeződik.

Azonban ha minden gyereknek ugyanaz a súlya, a véletlen csúcskiválasztó kiegyensúlyozatlan lesz, mert a kisebb részfákban található csúcsoknak nagyobb lesz a kiválasztási valószínűsége. Ezért minden X csúcsnál a részfa méretét is eltároljuk az $X.size$ nevű paraméterben, és ezt kiválasztási súlyként alkalmazzuk. Így az $X \rightsquigarrow X'$ lépés valószínűsége a következőképpen definiálható:

$$P(X \rightsquigarrow X') = \begin{cases} \frac{1}{X.size} & \text{ha } X' = X, \\ \frac{X'.size}{X.size} & \text{ha } X' \text{ az } X\text{-nek egy gyereke.} \end{cases}$$

Belátható, hogy ezzel a definícióval a fa összes csúcsát ugyanakkora valószínűséggel választjuk ki. Továbbá, mivel a kiválasztási útvonal a gyerekeken keresztül megy, és mindig lefelé halad, ezért a maximális úthossz a fa magasságával egyenlő, ami a csúcsok számát tekintve logaritmikus. Mivel az $X.size$ értéke csak az $X'.size$-től függ, ahol X' az X gyereke, ezért $X.size$ lentről-felfelé definiált tulajdonságnak tekinthető, és mint ilyen, paraméterként eltárolható a csúcsokban, így további számítás nélkül elérhető a véletlen csúcskiválasztáskor. Ez a módszer általánosítható úgy, hogy különböző kiválasztási súlyokat engedjünk meg minden csúcsra.

4.3.3. Derivációs fa mutáció

Az előzőekben definiált véletlen fagenerálással és véletlen csúcskiválasztással egy egyszerű mutáció könnyen definiálható. Először véletlenszerűen kiválasztunk egy csúcsot, majd a csúcs alatti részfát kicseréljük egy véletlenszerűen generált új részfára. A generált részfát korlátozhatjuk a kiválasztott csúcs aktuális attribútumaival. Például néha ésszerű egy adott mélységű részfát egy azonos mélységű részfával kicserélni, viszont általában nincs erre vonatkozó megkötés, csak az új részfa méretére adott globális korlát. Fontos megjegyezni, hogy ezek a mutáció operátorok csak lokális változásokat eredményeznek, és több paraméterrel szabályozhatóak. A mutáció költsége főként a véletlen fagenerálásból és a véletlen csúcskiválasztásból tevődik össze. Néhány további számítás szükséges az attribútumok újraszámításához a részfa beillesztése után, ami logaritmikus időigényű, ahogy azt a 4.2.2 fejezetben említettük.

4.3.4. Derivációs fa keresztezés

A derivációs fákra alkalmazható keresztezés a mutációnál egyszerűbben definiálható, mivel nincs szükség részfa generálásra. Csupán két csúcsot kell kiválasztani, és az ezek alatti részfákat megcserélni. Egy nehézség azonban jelentkezik, ugyanis a kiválasztott részfáknak azonos címkéjű gyökérrel kell rendelkezniük. Ha nem ez az eset áll fenn, a keresztezés kihagyható vagy újra meg lehet próbálni a kiválasztást. Megtehetjük azt is, hogy az első szülőben kiválasztunk egy csúcsot, és ezután a második szülőben egy speciális kiválasztással egy ugyanolyan címkéjű csúcsot keresünk, de ez a módszer nem hatékony. Az is lehetséges, hogy a keresztezés operátort újradefiniáljuk úgy, hogy az egész populáción működjön, ne csak szülőpárokon. Ez a megközelítés lehetővé teszi, hogy kicseréljünk részfákat akkor is, ha az egyszerű véletlen csúcskiválasztást alkalmazzuk, és a kiválasztott csúcs címkéjét nem határozzuk meg előre.

4.3.5. Készlet keresztezés

A keresztezés jelentősége, hogy a jó egyedek részeikkel hozzájárulnak az új generáció sikerességéhez. Ezt nem csak úgy érhetjük el, hogy veszünk két szülőt, és belőlük két leszármazottat hozunk létre. A keresztezést definiálhatjuk úgy, hogy az egész populáción működjön, akárcsak a globális ES rekombináció. A dolgozatban erre az ötletre építve egy új operátort definiáltunk a derivációs fákra *készlet keresztezés* néven.

Ez az operátor a következőképpen működik. Az első lépésben kiválaszt egy-egy részfát, eltávolítja ezeket a szülőkből és felveszi őket a *készletbe*, a kiválasztott csúcs címkéje szerint csoportosítva. A második lépésben ezeket a részfákat visszailleszti a szülőkbe véletlenszerű sorrendben.

4.3.6. Operátor költségek

Úgy tűnhet, hogy a fa műveletek, mint a részfa kivágása vagy beillesztése miatt az operátorok költsége lényegesen magasabb a bitvektorokra alkalmazott operátorokénál. Azonban pointer alapú implementációt használva a részfa kivágása és beillesztése konstans idő alatt elvégezhető a láncolt listákhoz hasonlóan.

Az operátor költségeket az általános esetre vizsgáltuk a dolgozatban, ezeket a 2. táblázat foglalja össze. A vizsgált operátorok a mutáció (MUT)

és a négy féle keresztezés: standard (XO), szimbólumonként súlyozott (XO-W), újrapróbálkozás (XO-R) és a készlet keresztezés (POOL). Az RNS és OP oszlopok mutatják a véletlen csúcskiválasztás és az operátor alkalmazásának időigényét. A megoldás méretét, vagyis a szó hosszát n-nel jelöljük és a 3.4 tétel alapján a fa méretét $\mathcal{O}(n)$-nek tekintjük.

	RNS	OP	frissítés	siker	méret
MUT	$\mathcal{O}(\log n)$	$\mathcal{O}(\lvert\mathcal{P}\rvert r)$	$\mathcal{O}(\log n)$	100%	$\mathcal{O}(1)$
XO	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$1/\lvert\mathcal{N}\rvert^a$	$\mathcal{O}(1)$
XO-R	$\mathcal{O}(\lvert\mathcal{N}\rvert \log n)^a$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	100%	$\mathcal{O}(1)$
XO-W	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\mathcal{N}\rvert \log n)$	100%	$\mathcal{O}(\lvert\mathcal{N}\rvert)$
POOL	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\approx 100\%^b$	$\mathcal{O}(1)$

[a] a nemterminálisok eloszlásától függően
[b] ha a populáció $\mathcal{N}$-hez viszonyítva nagy

2. táblázat. A DTGP operátorok költségeinek összegzése

Ez az elemzés a DTGP módszer (és más fa alapú GGGP módszerek) enyhe hátrányát mutatja a sztring alapú GGGP módszerekkel szemben, mivel az utóbbiak legtöbbször konstans időigényű operátorokat használnak, és a sikeres műveletek aránya 100%. Viszont az elemzés a DTGP más fa alapú módszerekkel szembeni előnyét is megmutatta. A paraméterek csúcsokban való eltárolásával a DTGP lineárisról logaritmikusra tudja csökkenteni az időigényt, mivel elkerüli, hogy a véletlen csúcskiválasztás során minden csúcsot be kelljen járni.

Az operátorok költségének tárgyalásánál a kiértékelés költségét is figyelembe kell vennünk. Először a genotípust, esetünkben a derivációs fát, le kell képeznünk a fenotípusra a fa határának leolvasásával, majd ezután a fenotípust ki kell értékelnünk a fitness függvény segítségével. Fontos megjegyezni, hogy a fenotípus mindig szintaktikailag helyes hipotézis, ezért nincs szükség szintaktikai ellenőrzésre vagy korrekcióra. A fa határának leolvasásához általában a fa bejárása szükséges, ami összevethető a lineáris GGGP-knél végrehajtott derivációval. Tehát a genotípus-fenotípus leképezés költsége mindkét megközelítés esetén $\mathcal{O}(n)$. Bizonyos problémákra azonban a DTGP alkalmazhat paramétereket a fenotípus előállításához vagy a fitness függvény kiszámításához, és így csökkentheti a kiértékelés költségét akár $\mathcal{O}(1)$-re. Egy ilyen továbbfejlesztés nem lehetséges olyan

algoritmusokkal, amelyek nem tárolják el a derivációkat, mint például a lineáris GGGP módszerek.

4.4. DTGP példa

A DTGP működésének szemléltetéséhez, és a tulajdonságainak elemzéséhez egy példát mutattunk be és értékeltük ki. A példa egy logikai regressziós probléma volt, amit a genetikus algoritmus tesztelésére is használtunk. A példához használt, az érvényes logikai kifejezéseket generáló környezetfüggetlen nyelvtan nagyon egyszerű, és az irodalomban gyakran ismertetett.

4.4.1. Eredmények

Az alapértelmezett beállításban a populáció mérete 1000 volt, és 100 lépést végeztünk el a mutáció és készlet keresztezés operátorokkal. A kezdeti populációban a fák magassága nem lehetett nagyobb 15-nél. A véletlen csúcskiválasztásnál minden csúcsnak ugyanaz volt a súlya, kivéve természetesen a leveleket. A mutációnál a véletlenszerűen generált részfák magasságát 10-re korlátoztuk. A fitness számításhoz a kifejezést mind a $2^5 = 32$ lehetséges kiértékeléssel kiszámítottuk, és összehasonlítottuk a célfüggvénnyel, amit egy 32-bites szám reprezentált. Minden találatot 1000 ponttal jutalmaztunk, majd a kifejezés méretét levontuk a pontszámból, így elméletileg a maximális fitness 32000-1. Az egyik futás alatt a 40. generációra az algoritmus talált egy kifejezést, amely pontosan leírja a függvényt, majd a megoldáshoz tartozó derivációs fa méretét tovább tudta optimalizálni 532-ről 247-re.

Az eredmény hasonló az általános GA algoritmus biztonságos mutációval elért eredményéhez. Viszont a GA eljárás egy 10000 egyedből álló populációt használt, amelyek közül sok érvénytelen volt. A DTGP algoritmusnak 1000 egyed elég volt. Az algoritmus által talált legjobb megoldás egy 93 szimbólum hosszú kifejezés. Az általános beállításokkal 100 külön tesztet futtattunk le, ezek közül 38 elért egy 31000 feletti fitness értéket, vagyis mind a 32 bit helyes volt, ami sikeres futást jelent. A medián azonban 31000 alatt volt, és a legrosszabb esetben a fitness kicsivel 27000 alatt maradt, ami csak 27 helyes bitet jelent.

4.4.2. Paraméter beállítások

A DTGP algoritmusnak sok beállítható paramétere van. A legfontosabb paraméterek az operátor alkalmazási ráta, a lépések száma és a populáció mérete. A DTGP algoritmus csúcskiválasztási paramétereit és a véletlen fagenerálás korlátait is be lehet állítani. A DTGP eljárást több beállítással teszteltük, az eredmények összefoglalása alább olvasható:

Operátor alkalmazási ráta A [14] cikkben bemutatott eredményeket megerősítve a különböző alkalmazási rátákkal kapott teszteredmények megmutatták, hogy a mutáció a DTGP legfontosabb operátora, bár a keresztezés magas alkalmazási rátája is javítja az algoritmust. Továbbá azt is bizonyítottuk, hogy ha a populáció elég nagy, a készlet keresztezés sikertelenségének valószínűsége elhanyagolhatóan kicsi.

A lépések száma Mivel az algoritmus által talált legjobb megoldás átlagos fitness értéke minden lépéssel fokozatosan javul, meg lehet próbálni az algoritmust tovább futtatni, bár, ahogy a tesztek megmutatták, a javulás mértéke egyre csökken. Ez főleg a populáció csökkenő diverzitásának köszönhető, és a dolgozatban megállapításra került, hogy két külön teszt futtatásával jobb eredményeket lehet elérni, mint egy teszt dupla hosszúságú futtatásával. Például a dolgozatban használt beállításokkal kétszer 100 lépéssel jobb eredményt kaptunk, mint egyszeri 200 lépéssel.

Populáció mérete A megfelelő populációméret fontos az evolúciós algoritmusok számára, mert a populáció diverzitásának megőrzéséhez szükség van egy bizonyos mennyiségű egyedre. Továbbá egy nagy populáció a keresési tér nagyobb részét fedi le. Viszont a populáció méretének növelése csak egy bizonyos szintig javít az eredményeken, ezért néha, akárcsak a lépések számánál, a populáció méretének növelése helyett jobb, ha több külön tesztet futtatunk. A dolgozatban bemutatott teszt 500-as populációmérettel érte el a legjobb eredményeket.

A dolgozatban a keresési aszimmetriát is vizsgáltuk. A módszer enyhe tendenciát mutat nagy fák generálása felé. Ez gyakori jelenség a genetikus programozásban, amit a szakirodalom *bloat*-nak nevez. [9]

5. A DTGP továbbfejlesztése

A DTGP által alkalmazott fa struktúra várhatóan aszimptotikusan nem nagyobb, mint a lineáris GGGP struktúrái, de mégis nagyobb és bonyolultabb adattípus. Szerencsére ez az adattípus lehetőséget nyújt az algoritmus továbbfejlesztésére. A dolgozatban bemutatott továbbfejlesztéseket három csoportba sorolhatjuk: Paraméterek lehetőségeinek kihasználása, szemantikai megkötések alkalmazása és a fagenerálás véletlenszerűségének javítása.

5.1. Paraméterek felhasználása

Paramétereket alkalmazhatunk a részfák különféle tulajdonságainak eltárolására. A dolgozatban ilyen információt a következő célokra használtunk:

A fa határának futásidőben történő előállítása A fa határát paraméterként eltárolhatjuk a gyökérben. Ez egy lentről felfelé definiált paraméter, ezért a DTGP könnyen tudja kezelni, és a fa határát konstans idő alatt elérhetővé teszi. Meg kell jegyeznünk azonban, hogy ez a paraméter egyedenként $\mathcal{O}(n \log n)$ tárhelyet igényel, $\mathcal{O}(n)$ helyett.

A hipotézis futásidőben történő kiértékelése Egy egyed kiértékeléséhez a fa határa gyakorlatilag irreleváns, amíg a reprezentált hipotézis kiértékelhető. Ha a hipotézist tömören tudjuk reprezentálni, akkor paraméterként használhatjuk, így a fitness számításnál konstans idő alatt elérhető lesz.

Futásidőben történő fitness számítás Bizonyos esetekben lehetséges, hogy részben vagy akár teljesen kiszámítsuk és eltároljuk a fitness érték részfához tartozó részét, például a részfa által reprezentált részhipotézishez tartozó költséget.

Operátor befolyásolás A véletlen csúcskiválasztáshoz bármilyen kiválasztási súlyt lentről felfelé definiált tulajdonságként paraméter formájában eltárolhatunk, és alkalmazhatjuk a véletlen csúcskiválasztás befolyásolására. A dolgozatban három példát mutattunk be: magasságküszöb szerinti kiválasztás, a szélesség-magasság aránytól függő súlyozás, és egy, a magassággal exponenciálisan csökkenő súlyozás.

5.2. Szemantikai megkötések

A DTGP-ben használt lentről felfelé definiált paramétereket tekinthetjük egy attribútum nyelvtan szintetizált attribútumainak. Ezért ezeket szintén *szintetizált attribútumoknak* vagy röviden *attribútumoknak* nevezzük. Az a attribútum lehetséges értékeinek halmazát V_a-val jelöljük. Ha másként nem jelezzük, az attribútumok egy adott szabályra értelmezhetők, mert még ha ugyanazt az információt is tartalmazzák minden csúcsban, a definíciójuk különböző lehet minden szabályra. Az attribútumokat a gyerek csúcsok attribútumaiból számítjuk ki, vagyis $T.a = f_a(T_1.a, \ldots, T_n.a)$.

Szemantikai megkötések alkalmazása azt jelenti, hogy a véletlen fák felépítésekor előre definiált szemantikai információt kell továbbítanunk lefelé a részfáknak, és esetleg aktualizálnunk is kell a folyamat közben. Ezt a folyamatot *disztribúciós halmazok* és *disztribúciós függvények* segítségével írhatjuk le, amelyek meghatározzák, hogy a szemantikai információt miképp osztjuk el a részfák között.

5.1. definíció. *(Disztribúciós halmaz)*
Adott egy szintetizált attribútum a, amit a következő függvény definiál:

$$T.a = f_a(T_1.a, \ldots, T_n.a)$$

A $v_0 \in V_a$ attribútum értékhez és az $n \in \mathbb{N}$ nemnegatív egészhez tartozó disztribúciós halmaz vektorok egy $D_a(v_0, n)$ halmaza, amit a következőképpen definiálunk:

$$D_a(v_0, n) = \{(v_1, \ldots, v_n) \in (V_a)^n \mid f_a(v_1, \ldots, v_n) = v_0\}$$

Tehát a disztribúciós halmaz az összes olyan értékvektor halmaza, ami az előre definiált értéket szintetizálja.

5.2. definíció. *(Disztribúciós függvény)*
Adott egy a szintetizált attribútum. Definiáljuk a $\mathcal{D} \subseteq V_a \times \mathbb{Z}^+$ halmazt, mint a legnagyobb olyan halmazt, amire minden $(v_0, n) \in \mathcal{D}$ párra a $D_a(v_0, n)$ disztribúciós halmaz nem üres.

Egy disztribúciós függvény egy $Dom(\hat{f}_a) = \mathcal{D}$ felett definiált véletlen $\hat{f}_a$ függvény, amelyre

$$\hat{f}_a(v_0, n) = \vec{v}, \text{ ahol } \vec{v} \in D_a(v_0, n).$$

Vagyis az $\hat{f}_a(v_0, n)$ a $D_a(v_0, n)$ egy véletlenszerűen kiválasztott eleme.

Disztribúciós halmazok és disztribúciós függvények segítségével definiálhatunk olyan attribútumokat, amelyek szemantikai megkötéseket tesznek a derivációs fákra. Ezeket *megkötött szintetizált attribútumoknak* nevezzük, és a szintetizált attribútumokkal ellentétben ezeket nem a részfák attribútumai alapján számítjuk ki, hanem az értékeket a disztribúciós függvényekkel a szülőktől a gyerek csúcsokba adjuk tovább.

Meg kell jegyeznünk, hogy van hasonlóság az örökölt és a megkötött szintetizált attribútumok között. A szemantikai információt mindkét esetben *fentről lefelé* továbbítjuk, de a megkötött szintetizált attribútumokat *lentről felfelé* definiáljuk.

5.3. definíció. *(Szemantikailag korlátozott deriváció)*
Adott $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$ nyelvtanra, az $\alpha_0, \alpha_1, \ldots, \alpha_n$, $\alpha_0 = S$ derivációs szekvencia szemantikailag korlátozott egy a megkötött szintetizált attribútummal, ha minden i lépésre $\alpha_{i-1} = \varphi A \psi$, $\alpha_i = \varphi\beta\psi$ és a $p : A \to \beta$ szabályra a következő teljesül:

$$D_{a,p}(A.a) \neq \emptyset,$$

és ha a β szó $k > 0$ nemterminálist tartalmaz, melyek $B_1, B_2, \ldots B_k$, akkor

$$(B_1.a, B_2.a, \ldots, B_k.a) \doteq \hat{f}_{a,p}(A.a),$$

ahol a $\doteq$ az egyik lehetséges értékkel való egyenlőséget jelöli.

A szemantikailag korlátozott deriváció során, amikor az $A \to \beta$ szabályt alkalmazzuk, vesszük az A-val címkézett csúcs a attribútumának v_0 értékét. Először ellenőrizzük a $D_{a,p}(v_0)$ halmazt, hogy az értéket szét lehet-e osztani. Ha a $D_{a,p}(v_0)$ üres, akkor az adott szabály nem alkalmazható. Ha találunk alkalmazható szabályt, akkor az $\hat{f}_{a,p}$ disztribúciós függvény segítségével meghatározzuk a β nemterminális szimbólumaihoz tartozó attribútum értékeket.

5.2.1. Véletlen fagenerálás

Véletlen fa generálásakor úgy kell részfákat generálnunk, hogy az attribútumaik a teljes fának megfelelő értéket szintetizálják. Ezért amikor a megfelelő szabály alkalmazásával a részfák gyökereit létrehozzuk, a disztribúciós függvényt használjuk a szemantikai megkötések részfák gyökerei közötti szétosztására.

A szemantikailag korlátozott derivációegy nehézsége, hogy zsákutcába futhat, vagyis egy részleges derivációs fához vezethet, aminek egy vagy több nemterminális csúcsában üres disztribúciós halmazok vannak. Ez a lehetőség probléma-specifikus, és ezt a disztribúciós függvények definiálásakor figyelembe kell vennünk.

5.2.2. Keresztezés

A standard keresztezés két fát választ ki, és mindkettőben egy azonos címkével rendelkező csúcsot keres. Megkötött szintetizált attribútumok alkalmazása esetén azonban az attribútumok értékeinek is egyezniük kell, különben konfliktus alakulna ki.

Ennek a problémának a megoldására módosítanunk kell az operátort úgy, hogy ne csak a címkék, hanem a megkötött szintetizált attribútumok is egyezzenek, amikor két részfát megcserélünk. A standard keresztezéssel ez nehézkes, de a készlet keresztezést módosíthatjuk úgy, hogy kezelje a megkötött szintetizált attribútumokat is. Ehhez az egyes készleteket nemcsak nemterminálisokkal, hanem a megkötött szintetizált attribútum értékeivel is meg kell jelölni.

Előfordulhat, hogy a készlet keresztezés sem működik. Ez akkor történhet meg, amikor egy készlet elemeinek száma pontosan egy. Ennek a valószínűsége nő a lehetséges készletek számával, és csökken a populáció méretének növelésével. Ezért a készlet keresztezés nem alkalmazható, ha túl sok nemterminális–attribútumérték kombináció létezik, hacsak a populáció méretét nem növeljük.

5.2.3. A szemantikai megkötések korlátai

Amikor szemantikai megkötéseket definiálunk, figyelembe kell vennünk a sikertelen derivációkat is. Ha a megkötés nagyon szigorú, nehézzé válik a helyes megoldások generálása. Ez történik, amikor a célfüggvényt szemantikai megkötésként használjuk a logikai regressziós problémán. A sikertelen derivációk elkerülése érdekében a derivációs fák magasságát növelnünk kell, de ezzel a fák mérete exponenciálisan növekszik. Tehát szemantikai megkötés bevezetése csak egy bizonyos mértékig használható. Továbbá könnyű belátni, hogy a készlet keresztezés a megkötések szigorításával egyre kevésbé hatékony, mivel a készletek száma is exponenciálisan nő.

5.3. Véletlenszerűsítés

A derivációs fa alapú genetikus programozás, mint bármely más sztochasztikus algoritmus, nagyban függ a véletlenszerűsítés (randomizáció) minőségétől, mivel ez befolyásolhatja a keresést, és meghatározhatja, hogy a keresési tér mely részeit vizsgáljuk nagyobb valószínűséggel. A dolgozatban a következőket vizsgáltuk:

Kiválasztás A kiválasztás operátort illetően a DTGP nem különbözik más evolúciós algoritmusoktól. Ezek több különböző kiválasztási módszert is definiálnak, melyek közül bármelyiket alkalmazhatjuk a DTGP-vel is.

Véletlen csúcskiválasztás a csúcsok súlyai alapján működik, amit paraméterként tárolunk. Az általános súly a levelekre 0, a többi csúcsra pedig 1. A dolgozatban két további példát mutattunk be a logikai regressziós problémán. A három tesztelt súlyozás közül a standard súllyal értük el a legjobb eredményt.

Véletlen szabály kiválasztás A véletlen szabály kiválasztás optimalizálásához bizonyos tulajdonságok alapján a szabályokhoz súlyokat rendelhetünk. A dolgozatban három lehetőséget teszteltünk, ezek a *szabály minimum értéke*, amit a min_p konstansban tárolunk, a *nemterminálisok száma* a jobb oldalon és a *rákövetkező szabályok száma*. Az eredmények azt mutatták, hogy a minimum érték szolgáltatja a legjobb eredményt.

Részfa korlát elosztás Miután egy szabályt kiválasztottunk és alkalmaztunk, vagyis új csúcsokat szúrtunk be a derivációs fába, a részfa korlátait újra kell számolnunk, és továbbadnunk a véletlen fagenerátornak minden nemterminális csúcsra. Ennek elvégzésére különböző stratégiák léteznek, a dolgozatban a következőket említettük: a standard eljárás, ami *véletlen* elosztást használ, a min_p-vel *súlyozott* és a *szekvenciális*. Azonban a tesztek nem mutattak lényeges javulást sem a logikai regressziós problémával, sem az integer generálással, ezért a gyakorlatban elég a standard disztribúciós stratégiát alkalmazni.

6. DTGP alkalmazások

A dolgozatban a DTGP bemutatására példaként a logikai regressziós problémát használtuk, de más feladatokra is alkalmaztuk a módszert, és röviden elemeztük őket. Az első példa a *6-Multiplexer* probléma volt, ami a genetikus programozás egy standard példája. A második példa az *utazóügynök probléma* volt, amit gyakran használnak optimalizálási módszerek tesztelésére. A harmadik példa a DTGP egy gyakorlati alkalmazása volt *FIR (finite input response) filterek* optimalizálására.

6.1. Multiplexer

A multiplexer egy n címbittel és 2^n adatbittel rendelkező logikai kapu, ami a címbitek által kiválasztott adatbitet adja vissza kimenetként. A dolgozatban három lehetőséget mutattunk be a 6-Multiplexer probléma (2 címbit és 4 adatbit) megoldására DTGP alkalmazásával. Az eredmények megmutatták, hogy a DTGP algoritmus képes megoldani a 6-Multiplexer problémát 88%-os sikeraránnyal, és ha szemantikai megkötéseket is alkalmazunk, az eredmény még jobb. Ez megmutatta a DTGP egyértelmű előnyét más fa alapú GGGP módszerekkel szemben.

6.2. Utazóügynök probléma (TSP)

Az utazóügynök problémára lefuttatott különböző tesztek eredményei azt mutatták meg, hogy a DTGP algoritmus alkalmazható a TSP útvonalának optimalizálására, bár a talált megoldás nem mindig optimális. Mivel az utazóügynök probléma megoldásai utak vagy körök egy gráfban, ezek belső struktúrája nagyban különbözik a környezetfüggetlen nyelvek szavaitól. Ezért egy szintaktikailag megkötött optimalizálási algoritmus alkalmazása nem a legjobb választás, és a szakirodalomban is csak nagyon kevés olyan példát találhatunk, amiben a hagyományos vagy nyelvtan vezérelt genetikus programozást a TSP probléma megoldására alkalmazták.

6.3. FIR filterek

A DTGP algoritmust alkalmaztuk *FIR filterek* optimalizálására egy közös projektben a Fraunhofer Társaság Integrált Áramkörök Intézetével. [15] Ennek az alkalmazásnak az összefoglalója is megtalálható a dolgozatban.

7. Összefoglalás

A dolgozatban egy új *nyelvtan vezérelt genetikus programozási* (GGGP) eljárás, a *derivációs fa alapú genetikus programozás* (DTGP) került definiálásra és kiértékelésre. Ez a módszer egy előre megadott környezetfüggetlen nyelvtan feletti derivációs fákat használ az egyedek reprezentálására, és ezekre a fákra alkalmaz genetikus programozást. Ennélfogva a *derivációs fa alapú GGGP*-k kategóriájába sorolható, és más nyelvtan vezérelt eljárásokhoz hasonlóan garantálni tudja, hogy az előállított egyedek az adott nyelvtan szempontjából mindig *szintaktikailag helyesek*.

A lineáris GGGP megközelítésekkel összehasonlítva a DTGP adattípusa nagyobb, bár legtöbbször nem aszimptotikusan, de mindenképp összetettebb. Ez a dolgozat bemutatta, hogyan lehet az evolúciós operátorokat helyesen és hatékonyan definiálni úgy, hogy ne csak a derivációs fák legyenek helyesek, hanem az időbonyolultság is logaritmikus maradjon az esetek többségében.

Azt is bemutattuk, hogyan lehet paramétereket alkalmazni az algoritmus továbbfejlesztésére. Ennek egy fontos alkalmazása a fent említett véletlen csúcskiválasztás. A lentről-felfelé paraméterek arra is alkalmasak, hogy a fitness számításhoz használható információt tároljunk el. Némely esetben a fenotípus, vagy akár a fitness érték is kiszámítható paraméterként, és így a kiértékelés időigénye konstans, egy további, logaritmikus idejű művelettel a paraméterek frissítésére.

A lentről-felfelé paraméterek értékeit előre is megadhatjuk, mint *szemantikai korlátokat*. A disztribúciós halmazok és disztribúciós függvények használatával ezeket az értékeket fentről lefelé haladva lehet továbbadni a részfáknak a véletlen fa generálása során. Ezt az eljárást *szemantikailag korlátozott derivációnak* nevezzük. A szemantikai korlátok alkalmazása egy jelentős előrelépés a korábbi GGGP eljárásokhoz képest, mert eddig ilyen korlátozásokat csak a fitness függvénybe lehetett beépíteni. Ebben az esetben olyan egyedek is létrejöttek, amelyek nem elégítették ki a szemantikai korlátokat, és ezeket később a fitness érték alapján kellett kiszűrni.

A dolgozatban a DTGP módszer a logikai regressziós problémán került részletes bemutatásra. Ezen kívül, a DTGP algoritmust teszteltük a 6-Multiplexer problémán, valamint bemutattuk, hogyan lehet alkalmazni az utazóügynök problémára. A módszer *FIR (finite input response) filterek* optimalizálására való gyakorlati alkalmazását szintén összefoglaltuk.

A dolgozat megállapításainak összefoglalása:

I. Tézis A derivációs fa alapú genetikus programozás, ahogy ebben a dolgozatban definiáltuk, egy specializált evolúciós algoritmus, ami különböző optimalizálási problémák megoldására alkalmazható a fekete doboz elv alkalmazásával, miközben az előállított egyedek szintaktikai helyessége is garantált.

- **a**. A jól definiált véletlen fagenerátorral a DTGP érvényes derivációs fákat állít elő, miközben a szükséges méretkorlátozást is betartja.
- **b**. Az operátorok alkalmazása előtt a DTGP logaritmikus időben ki tud választani egy véletlen csúcsot a derivációs fában, miközben biztosítja, hogy a kiválasztás valószínűsége minden csúcsra ugyanaz. Továbbá a kiválasztható csúcsok halmaza korlátozható, és szükség esetén nem uniform szelekciós súly is alkalmazható.
- **c**. A gyengén teljesítő hagyományos keresztezés helyettesíthető a készlet keresztezéssel, ami ugyanolyan időigényű, de legtöbbször 100%-os hatékonysággal működik.

II. Tézis A derivációs fák kibővített adattípusának köszönhetően, megfelelően definiált paramétereket alkalmazva, a DTGP viselkedése alakítható, és az algoritmus továbbfejleszthető.

- **a**. Megfelelő információ csúcsokban történő eltárolásával bizonyos esetekben a fitness függvény konstans időben kiértékelhető. A paraméterek frissítése további számítást igényel az operátorok alkalmazása után, de ez nem növeli a teljes időbonyolultságot.
- **b**. Paraméterek használatával a véletlen csúcskiválasztás, és így az evolúciós operátorok, valamint a fitness kiértékelés is befolyásolhatóak.
- **c**. Megkötött szintetizált attribútumok segítségével szemantikus korlátok helyezhetők az algoritmusba.

III. Tézis A DTGP alkalmazható különféle optimalizálási problémákra, különösen akkor, ha a megoldásoknak olyan struktúrájuk van, amely egy környezetfüggetlen nyelvtannal leírható.

Hivatkozások

[1] H. Alblas. Introduction to attribute grammars. In *Proceedings of the International Summer School on Attribute Grammars, Applications and Systems (SAGA'91)*, volume 545 of *LNCS*, pages 1–16. Springer Verlag, 1991.

[2] N. Chomsky. On certain formal properties of grammars. *Information and Control*, 2:137–167, 1959.

[3] A. E. Eiben and J. Smith. *Introduction to Evolutionary Computing*. Springer-Verlag, 2003.

[4] F. Gruau. On using syntactic constraints with genetic programming. In P. J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 19, pages 377–394. MIT Press, Cambridge, MA, USA, 1996.

[5] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[6] R. I. McKay, N. X. Hoai, P. A. Whigham, Y. Shan, and M. O'Neill. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, 11(3/4):365–396, Sept. 2010. Tenth Anniversary Issue: Progress in Genetic Programming and Evolvable Machines.

[7] D. J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, May 1995.

[8] M. O'Neill and C. Ryan. *Grammatical Evolution - Evolving programs in an arbitrary language.*, volume 4 of *Genetic Programming*. Kluwer Academic Publishers, 2003.

[9] R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[10] R. Ványi. Enforcing semantic constraints with derivation tree based genetic programming. Abstract accepted to oral presentation at Veszprém Optimization Conference: Advanced Algorithms (VOCAL 2012), 11-14 dec 2012.

[11] R. Ványi and S. Zvada. Avoiding syntactically incorrect individuals via parameterized operators applied on derivation trees. In R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon, editors, *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, volume 4, pages 2791–2798, Canberra, 8-12 dec 2003. IEEE Press.

[12] R. Ványi and S. Zvada. Syntactically correct genetic programming. In R. Poli et al., editors, *GECCO 2004 Workshop Proceedings*, Seattle, Washington, USA, 26-30 jun 2004.

[13] P. A. Whigham. Grammatically-based genetic programming. In J. P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41, Tahoe City, California, USA, 9July 1995.

[14] S. Zvada. *Attribute Grammar Based Genetic Programming.* Cuvillier Verlag, 2010.

[15] S. Zvada, G. Kókai, R. Ványi, and H. H. Frühauf. EvolFIR: Evolving redundancy-free fir structures. In *Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007)*, pages 439–446. IEEE Computer Society, 5-8 aug 2007.

[16] S. Zvada and R. Ványi. Improving grammar-based evolutionary algorithms via attributed derivation trees. In M. Keijzer, U.-M. O'Reilly, S. M. Lucas, E. Costa, and T. Soule, editors, *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 208–219, Coimbra, Portugal, 5-7 apr 2004. Springer-Verlag.