Automatikus tesztgenerálás formális protokollspecifikáció alapján

VINCZE GÁBOR

Budapesti Műszaki és Gazdaságtudományi Egyetem, Távközlési és Médiainformatikai Tanszék

vincze@alpha.ttt.bme.hu

Reviewed

Kulcsszavak: konformancia tesztelés, tesztgenerálás, mutáció-analízis, evolúciós algoritmusok, bakteriális algoritmus

A cikkben egy eljárást mutatunk be automatikus tesztgenerálásra a protokoll formális SDL specifikációja alapján. A protokolltesztelés a fejlesztési folyamat fontos része, ám a tesztkészletek kialakítása időigényes feladat. Ennek a fázisnak az automatizálása csökkenti a bevezetési időt, és egy komoly hibaforrást szüntet meg. Megmutatjuk, hogyan használható a mutáció-analízis egy állapottér-bejáró algoritmusból eredő tesztesetek, és a tesztkritériumok megfeleltetésének. Ezek után evolúciós algoritmusokat alkalmazunk egy optimális részhalmaz kiválasztására ebből a kezdeti tesztkészlet-halmazból. Ezeket az eljárásokat felhasználva egy teljes tesztgenerációs folyamatot építünk fel, amellyel egy protokoll formális specifikácójából tesztkészleteket kapunk.

1. Bevezetés

Ahogy a távközlési cégeknek egyre több szolgáltatást kellett nyújtaniuk, miközben hálózataik integrálására törekedtek, úgy nőtt a távközlési protokollok komplexitása. Ezzel egyidejűleg ezeknek a hálózatoknak egyre növekvő megbízhatósági követelményeknek kellett megfelelniük. Ezzel a komplexitás-növekedéssel a protokollok specifikációjához szükséges erőfeszítés súlyos teherré vált, és a megbízhatóság, valamint a gyártók termékeinek együttműködése iránti igény átfogóbb tesztelést tett szükségessé. Ezek a problémák hívták életre a formális specifikációs eljárásokat, valamint a formális tesztelési eljárásokat, amelyekkel ellenőrizni lehet, hogy egy alkalmazás a specifikációnak megfelelően működik-e.

A távközlési világban legelterjedtebben használt formális nyelvek a Specifikációs és Leíró Nyelv (Specification and Description Language, SDL [1]) a rendszerek specifikálására, amely a rendszert párhuzamosan működő kommunikáló véges automatákkal modellezi, és a Fa és Táblás Kombinált Jelölésmód (Tree and Tabular Combined Notation, TTCN [2]) a rendszerek fekete doboz jellegű ellenőrzésére.

Ma már a rendelkezésre állnak nagymértékben integrált és széles körben elterjedt fejlesztőeszközök [3], hogy segítsék a fejlesztőket a specifikációs és a vizsgálati folyamat során. Ennek ellenére a formális tesztkészletek előállítása még mindig jelentős munkát igényel, és az emberi tényező továbbra is a legdrágább, és legtöbb hiba forrása. Mivel a tesztkészleteket sokszor több százszor vagy ezerszer kell lefuttatni, a futási idő és a hardverkövetelmények szintén kulcsfontosságúak.

Ebben a cikkben bemutatunk egy módszert az automatikus tesztgenerálásra a rendszer SDL leírásából. Ennek a tesztgenerációs folyamatnak négy fő lépése van:

1) formális specifikálás SDL nyelven
2) teszteset-halmaz előállítása egy állapottér-bejáró algoritmussal
3) mutáció-analízis
4) egy optimális teszteset-részhalmaz kiválasztása

Először bemutatjuk a mutáció-analízis eljárást; ezek után megmutatjuk, hogyan alkalmazunk evolúciós algoritmusokat egy optimális teszteset-részhalmaz kiválasztására, majd végül bemutatjuk a teljes tesztgenerációs folyamatot az INRES protokoll példáján.

2. Mutáció-analízis

2.1. Áttekintés

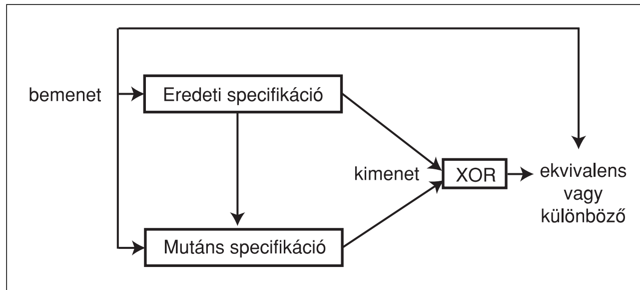
A mutáció-analízis egy fehér doboz módszer tesztesetek kialakítására, azaz a rendszer belső logikájának ismeretén alapul. A hagyományos mutáció-analízist a programkódokban található hibák felderítésére dolgozták ki, ám a mi esetünkben programok helyett specifikációkra alkalmazzuk, a megfelelő fekete doboz tesztesetek kiválasztásához.

Egy mutáció-analízis rendszerben definiálni kell egy mutációs operátor készletet [4], ahol minden operátor egy atomi szintaktikai változást testesít meg. Ezeknek az operátoroknak az alkalmazása két okból praktikus. Egyrészt lehetővé teszik a hibatípusok formális leírását, másrészt lehetővé teszik a mutánsok automatikus generálását. Az operátorokat szisztematikusan alkalmazva a specifikációra egy mutánskészletet generálhatunk.

Egy mutáció-analízis rendszer 3 komponensből áll:

– az eredeti rendszer,
– a mutáns rendszer – az eredeti rendszerhez képest egy apró szintaktikai változást tartalmaz. A mutánsokat a mutációs operátorok alkalmazásával kapjuk, ahol minden operátor egy apró szintaktikai változást testesít meg,

– orákulum – egy ember, vagy a legtöbb esetben egy gép, amely megkülönbözteti az eredeti rendszert a mutánstól a környezettel való interakciói alapján.



1. ábra Mutáció-analízis

Abból a feltételezésből indulunk ki, hogy a véges automatát megalkotó olyan specifikációt készít, amely közel áll az elvárásokhoz, és ezért azok a tesztesetek, amelyek felfedik a specifikáció szintaktikai változásait hasznosak. Csak elsőrendű hibákat idézünk elő, tehát egyszerre csak egy mutációt alkalmazunk, mert azok a tesztesetek, amelyek az egyszerű változásokat detektálják, az egyszerű változások sorozataként előállított komplex változásokat is detektálják [5].

A tesztesetek akkor különböztetik meg a mutánst az eredetitől, ha az más kimenetet ad. De az operátorok által generált mutánsok egy része szemantikailag ekvivalens lehet az eredeti rendszerrel, azaz a mutáns és az eredeti rendszer pontosan ugyanazt a kimenetet adná minden lehetséges bemenetre. Ezeket a mutánsokat *ekvivalensnek* nevezzük. Az olyan rendszereket, amelyek ugyanazt a kimenetet adják minden bemenetre, mint az eredeti rendszer, de szemantikusan nem ekvivalensek azzal, *pszeudo-ekvivalenseknek* nevezzük (az ekvivalens mutánsok a pszeudo-ekvivalens mutánsok egy részhalmaza). A teszteseteknél minden ekvivalenst figyelmen kívül kellene hagyjunk, és minden nem-ekvivalenst figyelembe kellene vennünk. Ez komoly problémát okoz a mutáció analízis rendszereknél, mivel általában nem lehetséges az ekvivalensek automatikus identifikálása, és az ekvivalensek és nem-ekvivalensek megkülönböztetése emberi közreműködést igényel.

2.2. Mutációs operátorok

A mutációs operátorok kialakításánál nagyon fontos szempont, hogy amennyiben lehetséges, ne adjanak egyetlen pszeudo-ekvivalenst se, és természetesen minimalizálják az ekvivalensek számát. Az operátorok kialakításának alapelvei:

– az operátorok atomi hibákat hivatottak modellezni;
– csak elsőrendű,
– csak szintaktikailag helyes;
– és csak szemantikusan helyes mutánsokat szeretnénk generálni;
– az operátorok véges, és a lehető legkisebb számú mutánst generálják.

Öt operátor osztályt van definiálva [4] a kommunikáló kiterjesztett véges automatákhoz, attól függően, hogy az automata mely részérét módosítják: állapot-, bemenet-, kimenet-, cselekvés- és predikátum-módosító operátorok.

Minden osztálynál három típusú operátort adhatunk meg, attól függően, hogy milyen jellegű hibát reprezentálnak: növelő, csökkentő és cserélő operátorok.

2.3. Teszteset – tesztkritérium megfeleltetés

A következő algoritmus segítségével egy véges méretű, strukturálatlan, és nagymértékben redundáns tesztkészlet (amelyet például egy a rendszerspecifikáció állapotterét bejáró állapottér-bejáró algoritmussal kaphatunk) minden egyes tesztesetéhez hozzárendelhetünk egy tesztkritérium-halmazt. Ha mutációs operátorokat alkalmazunk a nem megfelelő bemenetek megfigyelésére, ennek a kezdeti tesztkészletnek szintén tartalmaznia kell nem megfelelő teszteseteket.

Legyen C egy kétdimenziós, boole-algebrai értékeket tartalmazó mátrix.

0) Generáljunk egy teszteset-halmazt;
1) Alkalmazzuk egy mutációs operátort a véges automatára, hogy létrehozzuk az *i.* mutánst;
2) Futtassuk le az összes tesztesetet a mutáns specifikáción, és figyeljük meg az inkonzisztenciákat: amennyiben a teszteset az eredeti specifikációtól eltérő eredményt ad, a teszteset detektálja az adott mutánst
3) Hozzuk létre a *Ci* oszlopvektort (C mátrix *i.* oszlopát)
   – legyen *Ci[j] = 0*
   ha a *j.* teszteset nem detektálja az *i.* mutánst;
   – legyen *Ci[j] = 1*
   ha a *j.* teszteset detektálja az *i.* mutánst;
4) Ismételjük a 2-4. lépéseket, ahol *i* 1-től N-ig vesz fel értékeket, amíg létre nem hoztuk az összes lehetséges mutánst;
5) Nyerjük ki a C kritériummátrixot, ahol a sorok az eredeti halmaz teszteseteit ábrázolják, az oszlopok pedig a mutánsokat.

3. Tesztszelekció evolúciós algoritmusokkal

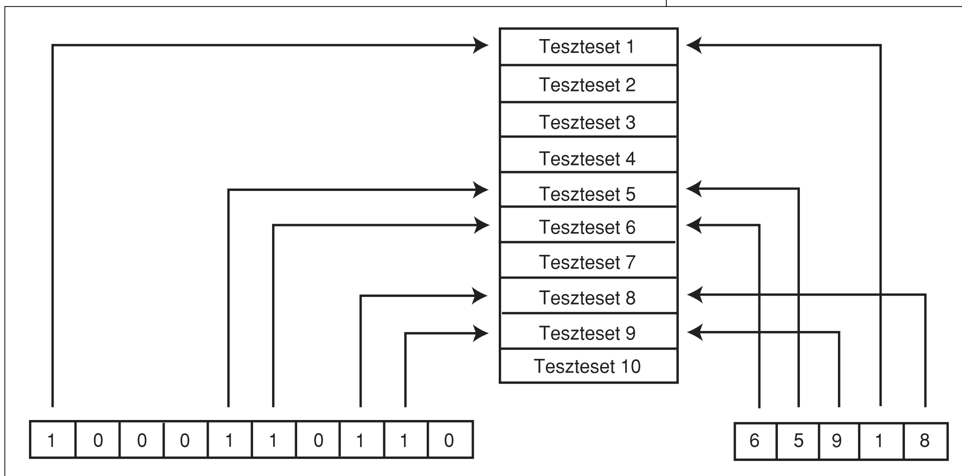
A szelekciós folyamat célja, hogy a tesztesetek egy optimális részhalmazát kapjuk a már meglévő strukturálatlan, és nagymértékben redundáns halmazból. Erre a célra három különböző „puha” algoritmust alkalmaztunk: a Genetikus Algoritmust (GA), a Pszeudo-Bakteriális Genetikus Algoritmust (PBGA), és a Bakteriális Evolúciós Algoritmust (BEA).

Az evolúciós algoritmusokra azért esett a választás, mert jó eredményeket adnak elfogadható időn belül, képesek az igen bonyolult esetek kezelésére is, és könnyen integrálhatóak a tesztgenerációs folyamatba [6].

3.1. Általános megfontolások

Egyedek: Egy *egyed* a probléma egy lehetséges megoldása, a mi esetünkben egy optimalizált tesztkészlet. Két lehetőségünk volt az egyedek ábrázolásá-

ra: egy fix hosszúságú, N bitből álló sorozat, ahol N az eredeti halmaz összes tesztkészletének száma, és egy bit értéke 1, ha az adott teszteset szerepel a tesztkészletben. Ezeket az egyedeket *bitsorozat* egyedeknek neveztük el. A másik megoldás egy változó méretű, 1 és N közötti értékeket tartalmazó halmaz, amelyben minden elem az eredeti halmaz egy tesztesetét ábrázolja. Ezeket az egyedeket *mutató-halmaz* egyedeknek neveztük el. Az utóbbi esetben természetesen lehetséges, hogy egy tesztkészlet többször tartalmazza ugyanazt a tesztesetet, ám ezek az egyedek magasabb futtatási költséggel rendelkeznek bármiféle egyéb érték nélkül, így hamar kiesnek a szelekció során. Az algoritmustól függően egyik vagy mindkét ábrázolási módot alkalmaztuk.



2. ábra Bitsorozat és mutató-halmaz egyedek

Teszteset költsége: a vizsgálati költség az adott teszteset futtatási költségét reprezentálja, ami jelenthet végrehajtási időt, vagy hardverkövetelményeket. Legyen T = {t1,t2,…,tn} a t1,t2,…,tn teszteseteket tartalmazó készlet, és R = {r1,r2,…,rk} az általa lefedett tesztkövetelmények halmaza.

Ekkor minden teszteset-halmazhoz hozzárendeljük a c: T→R pozitív függvényt.

Egy adott T tesztkészlet futtatási költségét ekkor az alábbi függvény adja:

$$c(T) = \sum_{t \in T} c(t) \tag{1}$$

Az egyéni tesztesetek futtatási költsége lehet tetszőlegesen kijelölt, vagy a mutáció-analízis fázis során megmért érték.

Itt azt feltételezzük, hogy minden tesztkövetelmény ellenőrzése bizonyos erőforrásigénnyel rendelkezik, valamint a teszteset inicializálása is erőforrásokat igényel. Így egy teszteset költségét az alábbiak szerint kapjuk meg:

$$c(t) = c1 + c2 * L \tag{2}$$

ahol *c1* az inicializációs költség, *c2* az egyes tesztkövetelmények ellenőrzéséhez rendelhető költség, L pedig az ellenőrzött tesztkövetelmények száma.

Célfüggvény: a célfüggvény méri az egyes egyedek minőségét, ezt próbálja minimalizálni az algoritmus. A kívánt tesztkészletek eléréséhez a célfüggvénynek a következőket kell figyelembe vennie:

- A tesztkészlet futtatási költségét minimalizálni szeretnénk, a lefedett tesztkövetelmények redundanciájának minimalizálásával.
- A tesztkészlet fedje le az összes követelményt.

Célfüggvényünk az összes teszteset végrehajtási költségeinek összege, valamint egy büntető érték minden egyes lefedetlen tesztkövetelményért:

$$O = c3 * C + c4 * M \tag{3}$$

ahol *C* az egyed költsége, *M* a lefedetlen követelmények száma, *c3* és *c4* pedig súlyozó tényezők, amelyeket úgy kell megválasztani, hogy ne legyen gazdaságos elhagyni a teszteseteket lefedetlen követelmények árán.

3.2. Genetikus algoritmus

A genetikus algoritmus egy olyan optimalizációs eljárás, amely a természetben lejátszódó szelekciós folyamatokat modellezi [7]. A kanonikus GA, amelyet itt alkalmaztunk, az alábbiak szerint működik:

```
Inicializálás
Kezdeti populáció létrehozása
Kezdeti populáció kiértékelése
generáció := 0

Generációs hurok
{
      Fitness értékek számítása
      Szelekció
      Rekombináció
      Mutáció
      Új egyedek kiértékelése
      Új egyedek visszahelyettesítése

      generáció := generáció + 1
} amíg generáció < max. generáció
```

Az egyedek bitsorozatok, mivel a keresztezés alkalmazása sokkal intuitívabb volt így. Tekintsük át egyenként az algoritmus lépéseit:

Fitness: Az egyedek fitness-értékét a lineáris rangsor-alapú módszer alapján végeztük, ahol az *i.* egyed *Fi* fitness-értékét az alábbi képlet adja:

$$F_i = 2 - sp + 2 * (sp - 1) * \frac{pos(fi) - 1}{N_{ind} - 1} \tag{4}$$

Ahol *sp* a szelekciós nyomás (a mi esetünkben *sp=2*), *pos(fi)* az *i.* egyed pozíciója a célfüggvény alapján, és *Nind* a populáció mérete.

Szelekció: Az egyedeket az utódok létrehozására a Sztochasztikus Univerzális Mintavételezési módszerrel választjuk ki: leképezzük az egyedeket egy számtengelyre, ahol minden egyednek a fitness-értékének megfelelő hossz jut. Ezek után generálunk egy véletlen számot az [1..szülők_száma] intervallumban, ahol *szülők_száma* az utódok létrehozására kiválasztandó egyedek száma. Ezek után ezt az értéket eltoljuk az i*(fitness-ek összege)/(szülők_száma) értékkel, ahol i ∈ [0 .. szülők_száma – 1], és minden egyes alkalommal kiválasztjuk azt az egyedet, amelyre ez az érték mutat a számtengelyen.

Rekombináció: Itt az egyenletes keresztezési módszert alkalmazzuk: generálunk egy véletlenszerű bitmintát. Ezek után úgy állítjuk elő az utódokat, hogy a szülők bitjeit felcseréljük azokban a pozíciókban, ahol ennek a maszknak az értéke 1.

Mutáció: Minden egyedet kis valószínűséggel mutálunk, hogy egy nagymértékű változásokat is lehetővé tegyünk. Egy véletlenszerű pozíciótól egy előre meghatározott hosszúságú szegmensen minden bitet *Pm* valószínűséggel mutálunk.

3.3. Pszeudo-bakteriális genetikus algoritmus

A 90-es évek második felében kifejlesztett bakteriális algoritmusok a baktériumok evolúciós folyamatait modellezik. A legegyszerűbb bakteriális algoritmus a pszeudo-bakteriális genetikus algoritmus [8].

Az algoritmus elején létrehozunk egy véletlenszerű egyedet, amelyre alkalmazzuk a bakteriális mutációt. Az eredeti egyedről *n – 1* másolatot (klónt) hozunk létre. Ezek után véletlenszerűen kiválasztjuk a kromoszóma egy részét, amelyet minden klónnál mutálunk, de változatlanul hagyjuk az eredeti egyednél. A mutáció után kiértékeljük az összes egyedet, és a legjobb egyed mutált részét átmásoljuk a többi klónba.

Ezt a mutáció-kiértékelés-szelekció-visszahelyettesítés ciklust addig ismételjük, amíg a kromoszóma összes részét nem mutáltuk. Ezek után kiválasztjuk a legjobb egyedet, a többit pedig megszüntetjük. A ciklust addig ismételjük, amíg kielégítő eredményt kapunk, vagy elérünk egy előre meghatározott generációszámot.

Ezt az algoritmust mindkét típusú egyeddel létrehoztuk. A bitsorozat típusú egyedeknél a mutáció megegyezik a GA esetén alkalmazottal. A mutató-halmaz egyedek esetében a mutációnak lehetővé kell tennie, hogy az egyed hossza megváltozzon, mivel nincsen a priori információnk az optimális egyedhosszúságról. Így a mutáció három típusú változást idézhet elő:

– teszteset helyettesítését egy másik tesztesettel;
– egy teszteset törlését, vagy
– egy teszteset hozzáadását.

3.4. Bakteriális evolúciós algoritmus

A bakteriális evolúciós algoritmus a PBGA egy továbbfejlesztett változata, ahol a keresést egyszerre több egyeden végezzük párhuzamosan. Ezt az algoritmust a baktérium-populációk géntranszfer képessége ihlette [9].

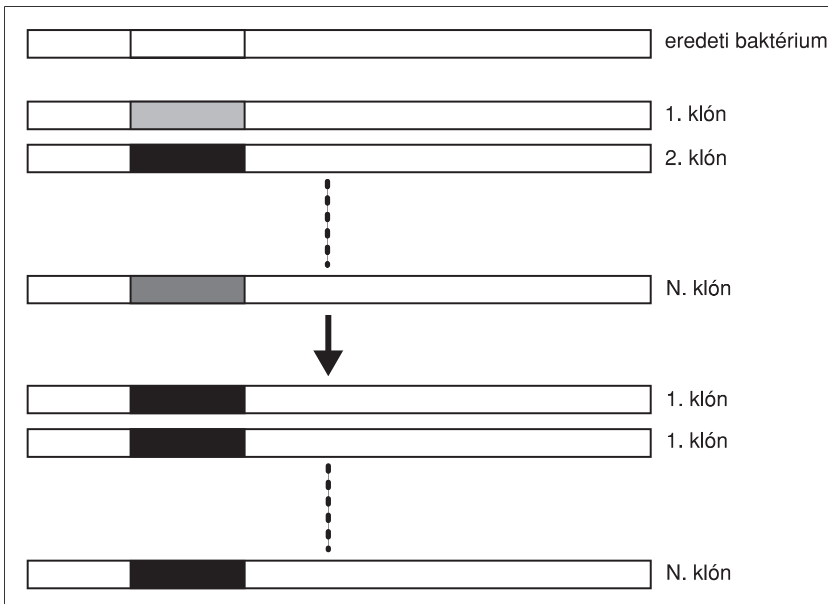
Az algoritmus az alábbiak szerint működik:

1) Létrehozunk egy *n* egyedből álló véletlenszerű populációt
2) Minden egyedre alkalmazzuk a bakteriális mutációt (a 3.3.-ban leírtak szerint)
3) *Ninf*-szer alkalmazzuk a géntranszfer műveletet, ahol *Ninf* az infekciók száma. Ennél a lépésnél egy alsó (rosszabb egyedek) és egy felső (jobb egyedek) félre osztjuk a populációt, és a felső félből az alsó félbe géneket helyezünk át.
4) A 2-4. lépéseket addig ismételjük, amíg kielégítő eredményt nem kaptunk, vagy elértünk egy előre definiált generációszámot.

Ennél az algoritmusnál módosítanunk kellett az egyedek felépítésén, hogy jól elhatárolt géneket tartalmazzanak, mivel a géntranszfer-műveletnél szükség van egy mérőszámra, ami azt mutatja meg, mennyire „jó” egy gén. A mutató-halmaz egyedeket egy előre meghatározott számú génre osztottuk, amelyek változó számú tesztesetet tartalmazó csoportok. A gén jóságának két különböző verzióját használtuk:

Első változat

Ennél az implementációnál egy gén jóságát az határozza meg, hogy átlagosan milyen költséggel fed le egy tesztkövetelményt.



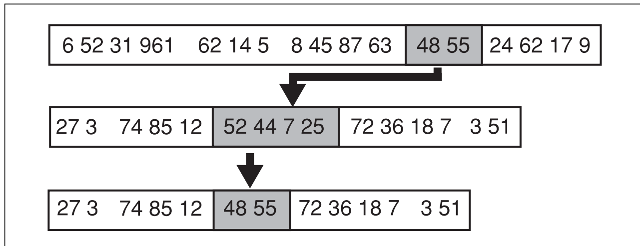
3. ábra
A pszeudo-bakteriális genetikus algoritmus

Így ezt a következőképpen számítjuk:

$$F=\frac{\sum_{i\in I} C_i}{R} \quad (5)$$

ahol F a gén jósága, C_i a tesztesetek költsége, I a gén teszteseteinek halmaza, és R a gén által lefedett tesztkövetelmények száma.

A géntranszfer-művelet során a felső fél egy baktériumából a legjobb génnel helyettesítjük az alsó fél egy baktériumának legrosszabb génjét *(4. ábra)*.



4. ábra Géntranszfer 1

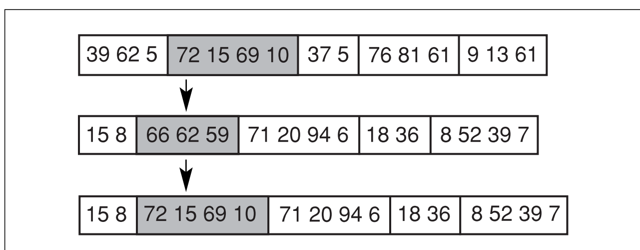
Második változat

Ennél a megközelítésnél annyi részre osztjuk a tesztkövetelményeket, ahány gént tartalmaz a baktérium. A célunk az, hogy minden gén a tesztkövetelmények egy meghatározott részét fedje le. Egy gén jóságát ugyanúgy határozzuk meg, mint a célfüggvényt az előző esetekben, de a lefedetlen tesztkövetelményeket csak a gén által lefedett intervallumon vesszük figyelembe. A gén jóságát az alábbi képlet adja *(5. ábra)*.

$$F = c1 * C + c2 * M_i \quad (5)$$

ahol F a gén jósága, C a gén költsége, Mi a gén által lefedett halmazon kihagyott tesztkövetelmények száma, $c1$ és $c2$ pedig súlyozó tényezők.

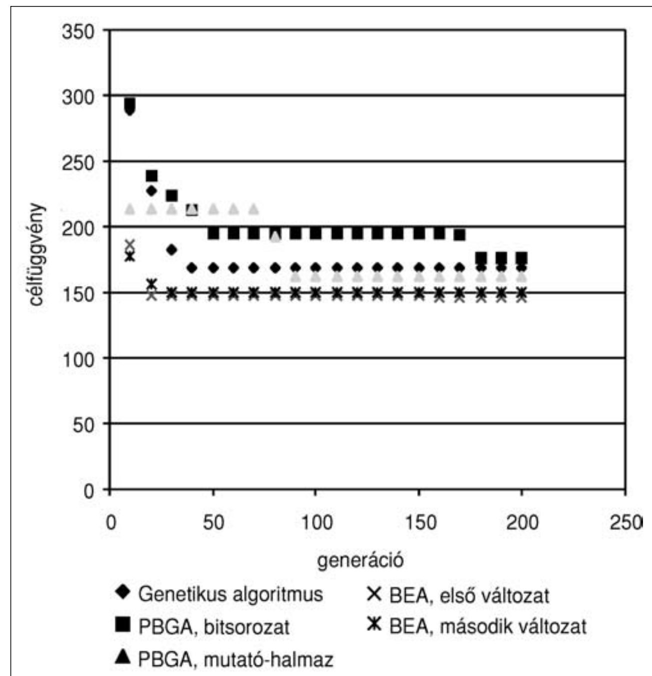
A géntranszfer során egy a felső félből vett forrásbaktériumból kiválasztunk egy véletlenszerű gént, és ha jobb az alsó félből vett célbaktérium ugyanazon pozíciójú génjénél, akkor helyettesítjük vele:



5. ábra Géntranszfer 2

3.5. Algoritmusok összehasonlítása

Hogy összehasonlíthassuk ezen algoritmusok hatékonyságát a teszteset-szelekció során, egy fiktív, 100 tesztesetet tartalmazó halmazon futtattuk őket (amint azt később látni fogjuk, az INRES protokoll kezdeti tesztkészlete csak 41 tesztesetet tartalmaz, ami túl kevés, hogy különbségek mutatkozzanak ezen algoritmusok konvergenciájában).



6. ábra Algoritmusok konvergenciája

A különböző algoritmusok konvergenciája a *6. ábrán* látható.

4. Automatikus tesztkészlet-generálás

Bemutatjuk a teljes tesztkészlet-generálási eljárást. Ezt a folyamatot a jól ismert INRES mintaprotokoll példájával illusztráljuk:

0. Létrehozunk egy formális SDL protokollspecifikációt. Erre a célra kiforrott eszközök állnak rendelkezésre [3]. A *7. ábra* mutatja az INRES protokoll SDL specifikációjának rendszer-áttekintő részét.
1. Az SDL specifikáción lefuttatunk egy állapottér-bejáró algoritmust, amely egy nagymértékben redundáns, strukturálatlan tesztkészletet eredményez.
2. A mutáció-analízis segítségével meghatározzuk a tesztkövetelmények mátrixát erre a teszteset-halmazra. Az *1. táblázat* az INRES rendszer SDL specifikációjának állapottér-bejárásából eredő 41 tesztesetből álló teljes tesztkészletét mutatja, az egyes tesztesetek költségével, valamint a teljes tesztkészlet költségével, ahol a tesztesetek költségét (2) szerint számítottuk, c1=20 és c2=5 értékekkel.
3. Kiválasztjuk a tesztesetek optimális részhalmazát a halmazból a fent bemutatott evolúciós algoritmusok egyikével. Ez egy olyan tesztkészletet eredményez, amely minimális redundanciával és végrehajtási költséggel lefedi az összes tesztkritériumot. A *2. táblázat* az INRES protokoll optimalizált tesztkészletét mutatja.

(Megjegyzés: Ebben az esetben a tesztesetek kiválasztása elég egyszerű, és bár nem feltétlenül van így nagyon nagy tesztkészletek esetében, minden evolúciós algoritmus ugyanazt a megoldást találta meg néhány generáció alatt.)

Teszteset	Lefedett tesztkritériumok	Teszteset költsége
inres01	48	260
inres02	19	115
inres03	36	200
inres04	21	125
inres05	44	240
inres06	34	190
inres07	46	250
inres08	21	125
inres09	27	155
inres10	60	320
inres11	11	75
inres12	46	250
inres13	89	465
inres14	59	315
inres15	58	310
inres16	49	265
inres17	17	105
inres18	46	250
inres19	47	255
inres20	66	350
inres21	21	125
inres22	65	345
inres23	24	140
inres24	82	430
inres25	25	145
inres26	26	150
inres27	78	410
inres28	29	165
inres29	71	375
inres30	30	170
inres31	36	200
inres32	34	190
inres33	66	350
inres34	62	330
inres35	35	195
inres36	88	460
inres37	37	205
inres38	39	215
inres39	84	440
inres40	41	225
inres41	48	260
Teljes tesztkészlet költsége:		**10145**

1. táblázat Kezdeti teszteset-halmaz

Teszteset	Lefedett tesztkritériumok	Teszteset költsége
inres10	60	320
inres13	89	465
inres14	59	315
inres23	24	140
inres27	78	410
inres28	29	165
Teljes tesztkészlet költsége:		**1815**

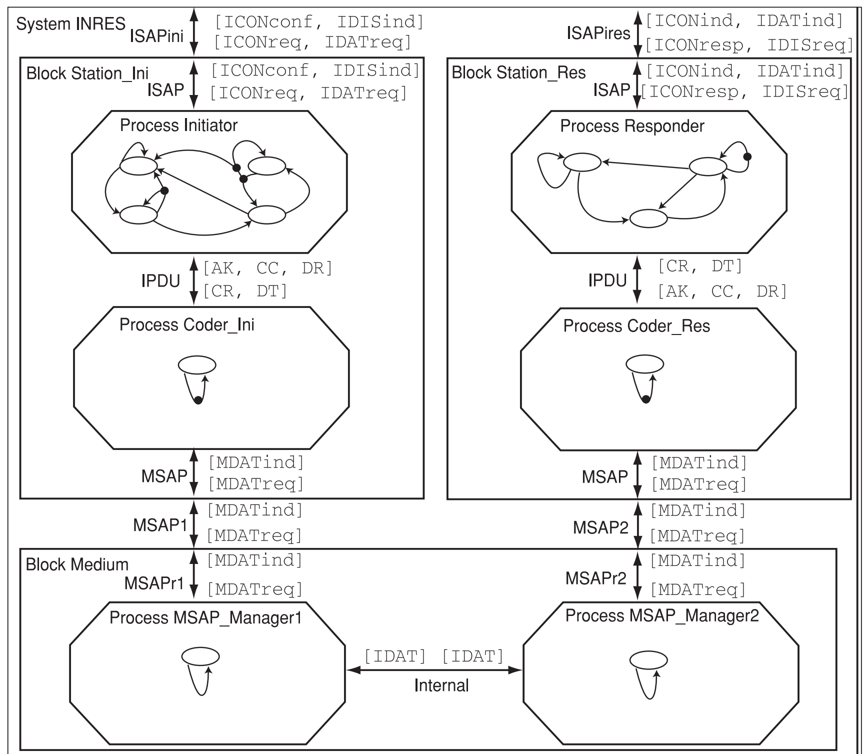
2. táblázat Optimalizált teszteset-halmaz

5. Konklúzió

Itt egy teljes automatikus tesztgenerálási módszert mutattunk be, amely a rendszer SDL specifikációjából állít elő egy tesztkészletet. Csupán egy egyszerű példával illusztráltuk az eljárást, de a mutáció-analízis bizonyítottan jól alkalmazható valós problémákra [4], és az evolúciós algoritmusok kifejlesztése mögötti motiváló erő kifejezetten a rendkívül komplex problémák kezelése volt.

A konformancia-vizsgálat a távközlési protokollok fejlesztési folyamatának kulcsfontosságú része. Mivel a tesztkészletek előállítása időigényes folyamat, az automatikus tesztgenerálás egyre fontosabb szerepet játszik a fejlesztési folyamatban. Ez a tesztkészlet-generációs eljárás könnyen implementálható, és működőképes megoldást kínál a való életbeli távközlési protokollok automatikus tesztgenerálására, nagymértékben lerövidítve ezzel a fejlesztési folyamatot.

További kutatások tárgyát képezheti, hogy milyen állapottér-bejáró algoritmusokat érdemes alkalmazni a legkedvezőbb kezdeti tesztkészlet kialakításához. A mutáció-analízis szintén egy gyorsan fejlődő terület, itt is lehetőség nyílhat a tesztesetek eddiginél gyorsabb és hatékonyabb azonosítására. A szelekciós folyamatban alkalmazott algoritmusok körét érdemes lehet tovább bővíteni, az újabb algoritmusok hatékonyságát megvizsgálni.



7. ábra Az INRES SDL specifikáció

Irodalom

[1] ITU-T. Z.100 ajánlás (1992):
Specification and Description Language (SDL)

[2] CCITT. X.292 ajánlás (1992):
The Tree and Tabular Combined Notation (TTCN)

[3] Telelogic Tau, http://www.telelocig.com

[4] Black P. E., Okun V., Yesha Y. (2000):
Mutation Operators for Specifications.
In The Fifteenth IEEE International Conference on Automated Software Engineering,
Proceedings ASE 2000, pp.81–88.

[5] Gábor Kovács, Zoltán Pap, Gyula Csopaki (2002):
Automatic Test Selection based on CEFSM,
Acta Cybernetica 15, pp.583–599.

[6] B. Kotnyek, T. Csöndes:
Heuristic methods for conformance test selection.

[7] J. H. Holland (1992):
Adaptation in Nature and Artificial Systems:
An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence, MIT Press, Cambridge

[8] M. Salmeri, M. Re, E. Petrongari, G. C. Cardarilli (1999):
A Novel Bacterial Algorithm to Extract the Rule Base from a Training Set,
Dept. of Electronic Engineering, University of Rome

[9] N. E. Nawa, T. Furuhashi (1999):
Fuzzy System Parameters Discovery by Bacterial Evolutionary Algorithm,
IEEE Tr. Fuzzy Systems 7, pp.608–616.