

UNIVERSITATEA POLITEHNICA BUCURESTI  
FACULTATEA DE AUTOMATICA SI CALCULATOARE

# Algoritmi genetici pentru rezolvarea problemelor prin evolutie si co-evolutie

Coordonator proiect:  
Prof. dr. ing. **Adina Magda Florea**

Absolvent:  
**OSTAFIEV Sorin**

**2003**

## Cuprins

1. Introducere.....	2
2. Evoluarea sub-componentelor co-adaptate.....	3
3. Algoritmii genetici .....	4
3.1. Introducere .....	4
3.2. Principii si caracteristici de baza.....	5
4. Problema acoperirii sirurilor .....	12
5. Reprezentarea si codificarea cunostintelor .....	12
5.1. Ajustarea procesului de inductie folosind sabloane de limbaj.....	14
5.2. Transformarea formulelor din logica cu predicate de ordinul I in siruri de biti.....	17
6. Operatorii genetici .....	19
6.1. Operatorii de incrucisare si de mutatie .....	19
6.2. Operatorul de inmultire .....	23
6.3. Operatorul de selectie “sufragiu universal” .....	24
6.4. Functia de fitness .....	28
7. Teoria formarii speciilor si a niselor.....	30
8. Concluzii .....	31
 Bibliografie .....	 32
 Anexe	
A. Imagini ale aplicatiei .....	34
B. Prezentarea aplicatiei (slide-show).....	35
C. Listingul codului sursa .....	46

## 1. Introducere

Ideea de baza prezenta in aceasta lucrare se refera la aplicarea algoritmilor evolutivi pentru probleme din ce in ce mai complexe. De asemenea, notiuni clare de modularitate trebuie introduse pentru a oferi metode eficiente care sa permita solutiilor sa evolueze sub forma unor sub-componente interactive co-adaptate. Un exemplu de asemenea natura este educarea comportamentului, asa cum se poate intalni in domeniul roboticii, unde un comportament complex poate fi descompus in comportamente simple. Ceea ce face gasirea solutiilor la probleme de acest fel deosebit de dificila este faptul ca componentetele sistemului sunt puternic dependente una fata de alta. Din aceasta cauza co-adaptarea este o cerinta critica pentru evolutia lor.

Exista doua mari motive datorita carora algoritmi evolutivi clasici nu sunt intru totul adecvati pentru rezolvare unor astfel de probleme. Primul, populatia de indivizi evoluata de acesti algoritmi are o puternica tendinta de convergenta, proportional cu numarul de generatii. Al doilea, indivizii evoluati pe algoritmi traditionali reprezinta de obicei solutii complete care sunt evaluate independent. Deoarece nu sunt modelate interactii intre membrii populatiei, nu exista presiune evolutiva pentru aparitia co-adaptarii. Pentru a complica si mai mult problema, se poate sa nu stim de la inceput cate sub-componente ar trebui sa fie sau ce rol sa asociem fiecarei componente. Dificultatea apare in gasirea de solutii computationale ale paradigmei noastre evolutioniste in care componentetele evolueaza mai mult singure decat asistate. O problema importanta este identificarea si reprezentarea unor asemenea sub-componentete, furnizarea unui mediu in care ele pot interactiona si co-adapta si impartirea sarcinilor pentru rezolvarea problemei in asa fel incat evolutia sa se desfasoare fara interventia umana.

## 2. Evoluarea sub-componentelor co-adaptate

Pentru a extinde modelul computational al evolutiei trebuie sa avem in vedere anumite probleme precum interdependenta intre subcomponente, descompunerea problemei initiale, asocierea rolurilor si controlul diversitatii.

Problema descompunerii consta in determinarea numarului de subcomponente si rolul pe care-l va juca fiecare. Pentru anumite probleme, o descompunere aproximativa poate fi cunoscuta inca de la inceput. Sa consideram o problema de optimizare a unei functii cu  $k$  variabile independente. O descompunere rezonabila ar fi impartirea problemei in  $k$  sarcini, fiecare asociata unei singure variabile. Oricum, exista multe probleme pentru care avem prea putina informatie sau deloc, cu privire la numarul sau rolurile sub-componentelor.

A doua problema se refera la evolutia sub-componentelor interdependente. Daca o problema poate fi descompusa in subprobleme independente intre ele, in mod evident fiecare poate fi rezolvata independent. Din punct de vedere grafic, ne putem imagina fiecare sub-componenta progresand pentru a atinge cea mai inalta pozitie, neluand in calcul pozitiile celorlalte sub-componente. Din nefericire, multe probleme pot fi descompuse doar in sub-componente cu dependente complexe intre ele. Selectia naturala va rezulta in co-adaptare intr-un asemenea spatiu doar daca interdependentele intre sub-componente sunt modelate.

A treia problema este determinarea contributiei fiecărei sub-componente la solutia finala ca un intreg. Aceasta este des intalnita in teoria jocurilor.

A patra problema consta in mentinerea diversitatii indivizilor din populatie suficient de mult de-a lungul generatiilor pentru a fi siguri ca spatiul solutiilor a fost suficient de bine explorat. De obicei, odata ce o solutie buna a fost gasita, algoritmul genetic se termina, cel mai bun individ reprezentand solutia gasita. In paradigma co-evolutiva, cu toate ca unele sub-componente pot fi mai puternice decat altele, luate individual, alti indivizi, tinand cont de contributia lor la rezultat si luandu-i ca un tot, poti fi prezenti in solutia finala.

## 3. Algoritmii genetici

### 3.1. Introducere

Algoritmii genetici sunt proceduri de calcul robuste si adaptive modelate pe mecanismul sistemului de selectie naturala. Algoritmii genetici se comporta ca o metafora biologica si incearca sa emuleze cateva dintre procesele observate in evolutia naturala. Ei sunt vazuti ca tehnici de cautare si optimizare aleatoare dar structurate. Ei isi exprima abilitatea prin exploatarea eficienta a informatiei trecute, informatie ce va fi folosita in crearea de noi urmasi cu performante imbunatatite. Algoritmii genetici sunt executati iterativ pe o multime de solutii codificate, numita populatie, cu trei operatori de baza: *selectia/reproducerea*, *incrucisarea* si *mutatia*. Fiecare dintre aceste solutii codificate sunt referite ca indivizi ai populatiei sau cromozomi.

Pentru a gasi solutia optima a unei probleme, un algoritm genetic incepe cu o multime data (sau generata aleator) de solutii (cromozomi) si evolueaza multimi diferite dar mai bune de solutii intr-un ciclu al iteratiei. In fiecare din aceasta iteratie (numita de aici inainte "*generatie*") functia de evaluare (functie ce masoara un criteriu de fitness) determina modul in care o solutie se apropie de solutia cea mai buna si, pe baza aceste evaluari, unele dintre ele (numite cromozomi parinti) sunt alese pentru reproducere. Este de asteptat faptul ca numarul de copii reproduse dupa un individ parinte sa fie direct proportional cu valoarea sa de fitness, incorporand astfel procedura de selectie naturala, pana la anumite limite. Procedura selecteaza cromozomi mai buni (cu o valoare mai mare a fitness-ului) si ii elimina pe cei rai. De aceea, performanta unui algoritm genetic depinde foarte mult de criteriul de evaluare a valorii fitness-ului. Pe acesti cromozomi parinti selectati sunt aplicati operatori genetici si astfel se genereaza cromozomi noi (urmasi).

Algoritmii genetici conventionali iau in considerare doar valoarea fitness-ului cromozomului in cauza pentru a-i masura aptitudinea pentru selectia in generatia urmatoare, adica fitness-ul unui cromozom este o functie de o valoare a functiei de evaluare. Fitness-ul unui cromoz  $x$  este  $g[f(x)]$ , unde  $f(x)$  este functia de evaluare, iar  $g$  este o alta functie care da valoarea fitness-ului primind o valoare  $f(x)$ . De aceea, un algoritm genetic conventional nu face discriminare intre doi urmasi identici, unul produs dn parinti mai buni (cu o valoare mai

mare a fitness-ului) și alții din părinți comparabil mai slabi (cu o valoare mică a fitness-ului). În natură, în mod normal un urmaș este mai potrivit dacă strămoșii săi (părinții) sunt mai buni, cu alte cuvinte un urmaș posedă o capacitate mai mare de a se descurca mai bine în mediul său dacă aparține unei familii mai bune (strămoși bine adaptați). Deci, valoarea fitness-ului unui individ depinde și de valoarea fitness-ului strămoșilor săi, în plus față de propria valoare. Aceasta este probabil intuiția de bază pentru a da mai multă greutate cromozomilor mai bine adaptați (cu o valoare mai mare a fitness-ului) să producă mai mulți urmași pentru generația următoare.

### **3.2. Principii și caracteristici de bază**

Algoritmii genetici intenționează să mimeze câteva dintre procesele observate în evoluția naturală în următoarele feluri:

Evoluția este un proces care operează mai degrabă asupra codificării entităților biologice decât asupra fiintelor vii, mai precis evoluția are loc la nivelul cromozomilor.

Similar, algoritmii genetici operează pe codificarea soluțiilor posibile (numite cromozomi) ale problemei prin manipularea șirurilor de caractere ale unui alfabet.

Selectia naturală este legătura dintre un cromozom și performanțele evoluției sale (măsurate pe versiunea decodificată). Natura se supune principiului darwinian “supraviețuirea celui mai bun”; cromozomii cu valori mai mari ale fitnessului se vor reproduce, în medie, mai des decât cei cu valori mici ale fitnessului.

În algoritmii genetici și selecția cromozomilor încearcă să mimeze această procedură. Cromozomii mai adaptați se vor reproduce mai des în dezavantajul celor mai puțin adaptați.

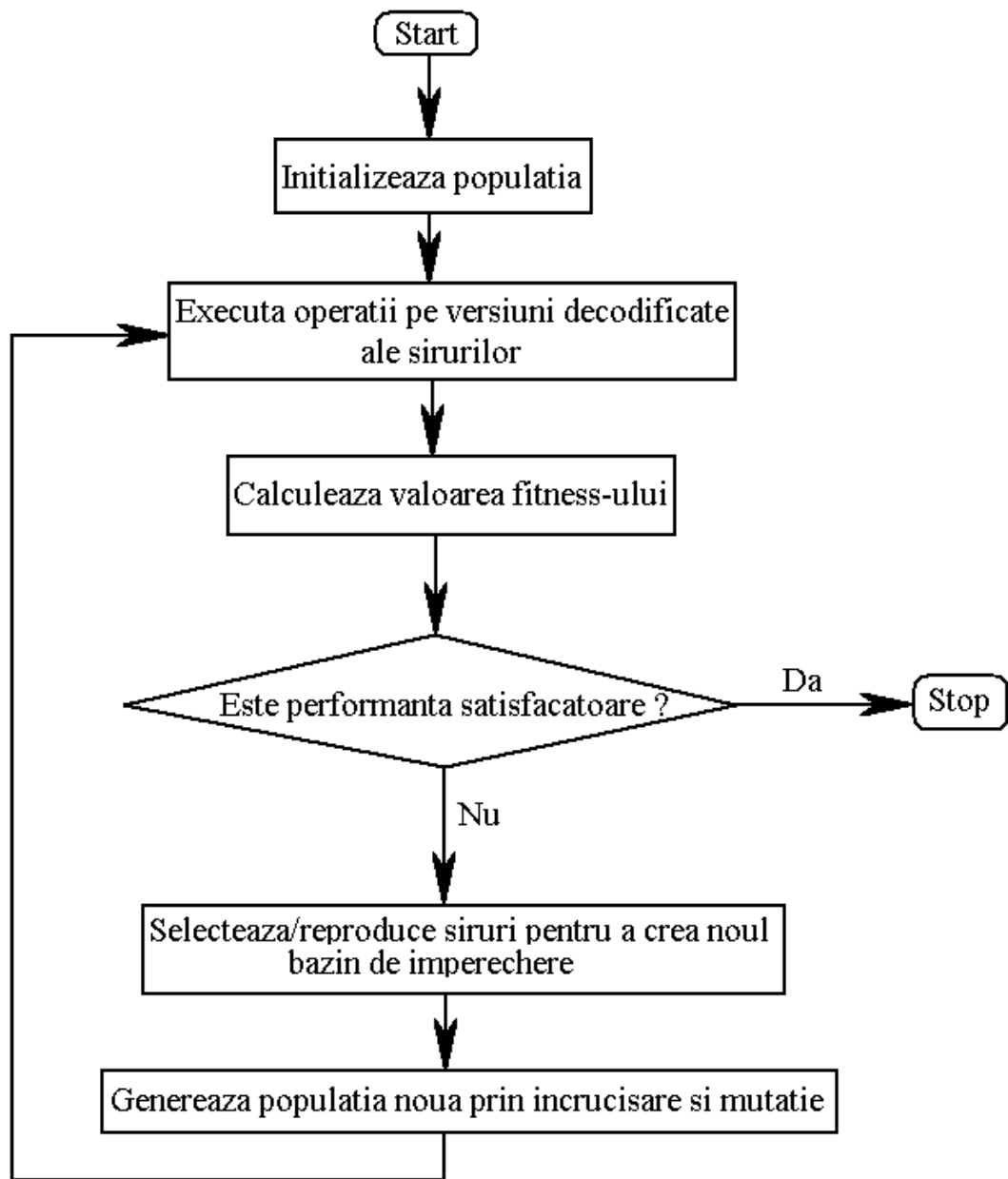
Evoluția biologică nu are memorie. Aici natura acționează ca mediu și entitățile biologice sunt modificate pentru a se adapta în mediul lor. Tot ceea ce natura cunoaște despre evoluția cromozomilor buni este conținut într-un set de cromozomi ai indivizilor curenți și în procedura de decodificare a cromozomilor.

Asemănător, și algoritmii genetici operează orbi pe cromozomi. Ei folosesc doar informația funcției de evaluare care acționează ca mediu. Bazându-se pe această informație fiecare cromozom este evaluat și în timpul procesului de

selectie se da o importanta mai mare alegerii cromozomilor cu o valoare mai mare a fitness-ului.

Ca si in evolutia naturala, se introduce variatia in entitatile algortimilor genetici in timpul reproducerii. Incrucisarea si mutatia sunt operatorii de baza pentru reproducere.

Evolutia bazata pe algoritmi genetici porneste de la o multime de indivizi (multimea solutiilor initiale pentru functia de optimizat) si avanseaza din generatie in generatie prin operatii genetice. Inlocuirea unei populatii vechi cu una noua este cunoscuta sub numele de generatie atunci cand se inlocuiesc toti membrii unei populatii cu altii noi. O alta tehnica de reproducere inlocuieste unul sau mai multi indivizi la un moment dat, in loc sa inlocuiasca toata popuatiia. Dupa cum s-a mentionat mai sus, algoritmul genetic are nevoie doar de o functie de evaluare ce actioneaza ca mediu pentru a observa potrivirea sau adaptabilitatea solutiilor (cromozomilor) derivate. In Figura 1 este prezentata o diagrama schematica a structurii de baza a unui algoritm genetic:



**FIGURA 1** – Pasii de baza ai unui algoritm genetic

Un algoritm genetic consta tipic din urmatoarele componente:

O populatie de siruri sau de solutii posibile codificate (referite biologic ca cromozomi)

Un mecanism de codificare a solutiilor posibile (cel mai adesea un sir binar)

O functie de evaluare a fitness-ului cu tehnica asociata ei

Procedura de selectie/reproducere

Operatori genetici (incrucisare si mutatie)

Probabilitati de a efectua operatii genetice

Sa descriem pe scurt aceste componente:

**Populatia** -- Pentru a rezolva o problema de optimizare, un algoritm genetic porneste cu reprezentarea cromozomiala (structurala) a unei multimii de parametri  $\{x_1, x_2, \dots, x_p\}$ . Multimea de parametri va fi codificata ca un sir de lungime finita peste un alafabet de lungime finita. De obicei cromozomii sunt siruri compuse din 0 si din 1. Spre exemplu, fie multimea  $\{a_1, a_2, \dots, a_p\}$  o instanta a multimii de parametri si fie reprezentarea binara a lui  $a_1, a_2, \dots, a_p$  10110, 00100, ..., 11001. Atunci sirul 1011000100...1101 este o reprezentare cromozomiala a multimii de parametri  $\{a_1, a_2, \dots, a_p\}$ . Este evident ca numarul de cromozomi (siruri) diferiti este  $2^l$ , unde  $l$  este lungimea sirului. De fapt, fiecare cromozom este o instanta a unei solutii posibile. O multime de astfel de cromozomi intr-o generatie se numeste populatie. Marimea unei populatii poate varia de la o generatie la alta sau poate fi constanta. De obicei populatia initiala se alege in mod aleator.

**Mecanismul de codificare/decodificare** – Este mecanismul de conversie a valorilor parametrilor unei posibile solutii in siruri binare rezultand reprezentari cromozomiale. Daca solutia unei probleme depinde de  $p$  parametri si daca dorim sa codificam fiecare parametru cu un sir binar de lungime  $q$ , atunci lungimea fiecarui cromozom va fi  $p \times q$ . Decodificarea este operatiunea inversa codificarii.

**Functia de evaluare a fitness-ului si tehnica asociata ei** – Functia de evaluare a fitness-ului este aleasa in functie de problema. Alegerea se face in asa fel incat sirurile mai potrivite, mai adaptate (posibile solutii) sa aiba o valoare mai mare a fitness-ului. Aceasta este singura modalitate dupa care se alege un cromozom pentru a se reproduce in generatia urmatoare.

**Procedura de selectie/reproducere** – Procesul de selectie/reproducere copiaza siruri individuale (numite cromozomi parinti) intr-o tentativa de noua populatie (numita bazinul de imperechere) pentru operatii genetice. Numarul copiilor repoduse pentru generatia urmatoare de un individ este de asteptat sa fie direct proportional cu valoarea sa de fitness, imitand astfel procedura de selectie naturala pana la un anumit punct. Cele mai frecvente proceduri de selectie sunt selectia prin roata ruletei si selectia lineara.

Operatorii genetici se aplica pe cromozomii parinti si sunt generati noi cromozomi (numiti urmasi). In cele ce urmeaza se vor descrie cei mai des folositi operatori genetici.

**Incrucisarea** – Scopul principal al incrucisarii este schimbul de informatie intre cromozomi parinti alesi in mod aleator pentru a nu pierde nici o informatie importanta (distrugere minima a structurilor cromozomiale ce sunt selectate pentru operatii genetice). De fapt se combina materialul genetic a doi cromozomi parinti pentru a produce urmasi in noua generatie. Operatie de incrucisare poate fi vazuta ca un proces in trei pasi. Intr-o prima faza, din bazinul de imperechere sunt alese perechi de cromozomi (numite perechi pentru reproducere). Al doilea pas determina, pe baza probabilitatii de incrucisare (generandu-se un numar aleator in intervalul  $[0, 1]$  si verificandu-se daca acesta este mai mic decat probabilitatea de incrucisare) daca aceste perechi de cromozomi vor fi incrucisate sau nu. In al treilea pas se efectueaza interschimbul de segmente cromozomiale intre perechile alese. Numarul segmentelor care se vor schimba si lungimea acestora depinde de tipul de incrucisare folosit. Unele dintre cele mai folosite tehnici de incrucisare sunt incrucisarea printr-un punct, incrucisarea prin doua puncte, incrucisarea prin puncte multiple, incrucisarea uniforma, incrucisarea shuffle-exchange. Pentru a ilustra modul in care sunt schimbate segmente din cromozomii parinti, sa consideram tehnica de incrucisare printr-un punct. Fie pozitia  $k$  selectata in mod aleator uniform intre 1 si  $l-1$ , unde  $l$  este lungimea sirului (mai mare decat 1). Cele doua noi siruri sunt create prin interschimbarea tuturor caracterelor de la pozitia  $(k + 1)$  pana la  $l$ . Fie :

a = 11000 10101 01000 ... 01111 10001  
b = 10001 01110 11101 ... 00110 10100

doua siruri (parinti) selectate pentru operatia de incrucisare si fie numarul generat aleator egal cu 11 (unsprezece). Atunci, cei doi urmasi produși (prin interschimbarea tuturor caracterelor dupa pozitia 11) vor fi:

$a' = 11000\ 10101\ 01101\ \dots\ 00110\ 10100$

$b' = 10001\ 01110\ 11000\ \dots\ 01111\ 10001$

**Mutatia** – Scopul principal al mutatiei este de a introduce diversitate genetica in populatie. Cateodata ea ajuta la recastigarea informatiei pierdute in generatiile anterioare. Ca si in sistemul genetic natural, mutatia in algoritmi genetici este facuta ocazional. Se alege o pozitie aleatoare intr-un sir ales aleator si este inlocuita cu un alt caracter din alfabet. In cazul reprezentarii binare mutatia neaga valoarea bitului si este cunoscuta sub numele de mutatie de bit. De exemplu, sa presupunem ca al treilea bit al sirului  $a$  de mai sus a fost selectat pentru mutatie. Atunci, sirul transformat dupa mutatie va fi:

11100 10101 01000 ... 01111 10001

In mod normal rata mutatiei este tinuta fixa. Pentru a sustine diversitatea (care poate fi pierduta datorita incrucisarii si a ratei foarte mici a mutatiei) intr-o populatie s-a propus o tehnica numita *mutatie adaptiva*, unde probabilitatea de a efectua o mutatie este facuta sa creasca (in loc de a fi tinuta fixa) pentru a creste omogenitatea genetica intr-o populatie. Mutatia nu da intotdeauna rezultate meritorii. Valori inalte ale ratei de mutatie pot conduce cautarea genetica intr-una aleatoare. Ea poate schimba valoarea unui bit important si de aceea poate afecta convergenta rapida catre o solutie buna. Mai mult, mutatia poate chiar incetini procesul de convergenta in stagiile finale ale algoritmilor genetici. In [Bhandari si Pal, 1994] a fost propusa o noua tehnica de mutatie cunoscuta sub numele de *mutatie directionata*.

**Probabilitati pentru efectuarea operatiilor genetice** – Probabilitatea de a efectua operatia de incrucisare este aleasa intr-un asemenea mod astfel incat recombinarea sirurilor puternice (cromozomi cu valori mari ale fitness-ului) creste fara intrerupere. In general, probabilitatea de incrucisare se afla undeva intre 0.6 si 0.9.

Deoarece mutatia intervine ocazional, este clar faptul ca probabilitatea de a efectua o mutatie va fi una foarte mica. Aceasta valoare se afla undeva intre 0.0001 si 0.01.

Probabilitatile de incrucisare si de mutatie pot fi tinute fixe de-a lungul operatiilor unui algoritm genetic sau pot fi adaptate (determinate automat in functie de mediu) pentru a imbunatati performantele, daca e posibil.

Intr-un algoritm genetic standard nu se pastreaza solutia cea mai buna obtinuta pana la un moment dat, marind sansele de a pierde obtinerea celor mai

bune solutii posibil. Strategiile *elitiste* sunt cele care vin sa depaseasca acesta problema prin copierea celui mai bun membru al fiecarei generatii in generatia urmatoare. Desi aceasta strategie poate mari viteza de dominare a unei populatii de catre un sir puternic (sir cu o valoare mare a fitness-ului), ea mareste performantele unui algoritm genetic folosind inlocuirea generatiilor. Recent au fost introduse conceptele de algoritmi genetici distribuiti si de algoritmi genetici paraleli.

**Caracteristici distinctive** – Urmatoarele caracteristici faciliteaza algoritmi genetici sa-si mareasca aplicabilitatea peste cele mai multe din strategiile de optimizare si de cautare:

Algoritmi genetici lucreaza simultan pe mai multe puncte si nu pe unul singur, ceea ce ajuta la introducerea unui paralelism implicit puternic in procedura computationala. Aceasta caracteristica previne si faptul ca algoritmul genetic sa nu se “impotmoleasca” intr-un minim local.

Algoritmi genetici lucreaza cu o multime de parametri codificati, nu cu parametri insusi. De aceea, rezolutia spatiului de cautare posibil poate fi controlata prin varierea lungimii codificarii parametrilor.

In algoritmi genetici nu e necesar ca spatiul de cautare sa fie continuu (spre deosebire de metodele de cautare bazate pe calcul).

Algoritmi genetici sunt orbi. Ei folosesc doar informatia functiei de evaluare si nu au nevoie de informatii auxiliare, cum ar fi derivari din functia de optimizare.

Algoritmi genetici folosesc reguli de tranzitie probabilistice, nu deterministice.

Recent, algoritmi genetici gasesc aplicatii foarte raspandite in rezolvarea problemelor care cer cautari eficiente si practice in cercurile de afaceri, de stiinta si cele ingineresti, de exemplu: sinteza arhitecturii retelelor neuronale, problema comis-voiajorului, a colorarii grafurilor, optimizari numerice, sisteme de clasificare, recunoastere de forme, etc.

#### **4. Problema acoperirii sirurilor**

In cele ce urmeaza vom considera problema acoperirii sirurilor ca problema de referinta, in mare parte deoarece furnizeaza un mediu relativ simplu in care descompunerea si evolutia cooperativa pot fi usor explorate. Problema consta in gasirea unui set de  $N$  vectori binari care se "potrivesc" cat se poate de bine pe un alt set de  $K$  vectori binari si unde de regula  $K$  este mult mai mare decat  $N$ . Le vom numi pe acestea set de gasit si respectiv set sursa. Dandu-se faptul ca  $K$  este mult mai mare decat  $N$ , setul de gasit trebuie sa contina sabloane care se regasesc in multiple siruri din setul sursa pentru a acoperi setul sursa in mod optim, cu alte cuvinte, setul de gasit trebuie sa generalizeze. Calitatea potrivirii intre doi vectori  $x$  si  $y$  de lungime  $L$  se noteaza cu  $S$  si se determina simplu prin numararea pozitiiilor identice din cei doi vectori. Pentru a calcula calitatea potrivirii unui set  $M$ , aplicam pe rand fiecare element din el asupra setului destinatie si apoi calculam media aritmetica a acestora.

De asemena in acest caz, dimensiunea setului de gasit poate sau nu sa fie specificata de la inceput. Problema descompunerii in acest context consta in determinarea marimea setului de gasit si acoperirea fiecarui set posibil asupra setului destinatie.

In particular, algoritmul evolutiv utilizat in acest caz este un algoritm genetic. In toate cazurile, initializarea populatiilor se face aleator, se foloseste un crossover in 2 puncte la o rata prestabilita si o mutatie ce consta in schimbarea starii unui bit tot la rata prestabilita depinzand de lungimea sirurilor, iar selectia indivizilor se face proportional cu fitness-ul indivizilor.

#### **5. Reprezentarea si codificarea cunostintelor**

Sistemele bazate pe algoritmi genetici sunt capabile sa invete din instante structurate ale conceptelor. In particular, o instanta a unui concept este reprezentata ca o colectie de obiecte, fiecare din ele fiind descris de un vector de attribute. Pentru tratarea acestui fel de reprezentare este mai adecvat un limbaj

pentru descrierea de concepte in logica cu predicate de ordinul I, adica un limbaj cu variabile.

Limbajul de descriere al conceptelor  $L$ , folosit de sistem, este un limbaj cu predicate de ordinul I. Mai precis,  $L$  este un limbaj cu clause Horn, in care termenii pot fi variabile sau disjunctii de constante si negatia apare intr-o forma restrictiva. Disjunctia interna a fost larg folosita in invatarea conceptelor deoarece permite exprimarea compacta a ipotezelor inductive. Un exemplu de expresie atomica continand un termen disjunctiv este "forma( $x$ , patrata  $\vee$  rotunda)", care este echivalenta semantic cu "forma( $x$ , patrata)  $\vee$  forma( $x$ , rotunda)", dar este mai naturala.

In sistemul propus, o formula atomica de aritate  $m$  are forma sintactica  $P(x_1, x_2, \dots, x_m, \mathbf{K})$ , unde  $x_1, x_2, \dots, x_m$  sunt variabile, iar termenul  $\mathbf{K}$  este o disjunctie de termeni constanti,  $[v_1, v_2, \dots, v_n]$  sau negatia unei astfel de disjunctii,  $\neg[v_1, v_2, \dots, v_n]$ . Cel mult un termen poate fi o expresie disjuncta (pozitiva sau negativa), in timp ce toate celelalte trebuie sa fie variabile. Iata cateva exemple de expresii atomice bine formate:

$forma(x_1, [patrata, rotunda]), forma(x_1, \neg [triunghiulara, octogonala])$   
 $distanta(x_1, x_2, [5, 6, 7]), superior(x_1, x_2)$

De notat faptul ca expresia  $forma(x_1, \neg [triunghiulara, octogonala])$  este echivalenta semantic cu  $\neg forma(x_1, [triunghiulara]) \wedge \neg forma(x_1, [octogonala])$ . De aceea, negatia interna a unui disjunct este echivalenta cu conjunctia atomilor negati in forma clauzala. Cand o formula  $\phi$  este evaluata pe o instanta  $\xi$  a conceptului, fiecare variabila in  $\phi$  va trebui sa fie legata de un obiect ce apare in descrierea lui  $\xi$ . Apoi, predicatele ce apar in  $\phi$  sunt evaluate pe baza atributelor obiectului de care e legata variabila. Deoarece legatura intre variabilele din  $\phi$  si obiectele din  $\xi$  se poate alege in multe moduri, se spune ca  $\phi$  este adevarat pentru  $\xi$  daca si numai daca exista macar o alegere astfel incat toate predicatele ce apar in  $\phi$  sunt adevarate. Modul in care semantica predicatelor este evaluata relativ la attributele obiectului trebuie definit explicit de catre utilizator inainte de a rula REGAL pe o aplicatie specifica. Pentru a clarifica acest lucru, sa presupunem ca o instanta a conceptului este compusa din trei obiecte:

$o_1$ :<culoare = neagra, pozitie = 1, marime = 4, inaltime = 3>  
 $o_2$ :<culoare = verde, pozitie = 2, marime = 1, inaltime = 1>  
 $o_3$ :<culoare = albastra, pozitie = 3, marime = 10, inaltime = 5>

Fiecare obiect este descris de patru attribute: *culoare*, *pozitie*, *marime* si *inaltime*. Sa mai presupunem ca semantica pentru predicatele culoare( $x_1$ ,

[galbena, verde]),  $distanta(x_1, x_2, [2, 3, 4])$ ,  $mai\_mare(x_1, x_2)$  si  $inalt(x_1)$  este definita de expresiile:

$membru(culoare(x_1), \mathbf{K})$ ;  
 $membru(|pozitie(x_1) - pozitie(x_2)|, \mathbf{K})$ ;  
 $marime(x_1) > marime(x_2)$ ;  
 $inaltime(x_1) > 6$ .

Atunci, predicatul  $culoare(x_1, [galbena, verde])$  va fi adevarat cand variabila  $x_1$  va fi legata de obiectul  $o_2$  si va fi fals pentru orice alta alegere a legaturii (termenul  $\mathbf{K}$  este legat la [galbena, verde]). Asemnator,  $distanta(x_1, x_2, [2, 3, 4])$  este adevarat doar cand  $x_1$  este legat de  $o_3$  si  $x_2$  de  $o_1$  (termenul  $\mathbf{K}$  este legat la [2, 3, 4]),  $mai\_mare(x_1, x_2)$  este adevarat pentru trei alegeri ale legaturilor  $\langle x_1=o_1, x_2=o_2 \rangle, \langle x_1=o_3, x_2=o_1 \rangle, \langle x_1=o_3, x_2=o_2 \rangle$  si  $inalt(x_1)$  este mereu fals.

In cele ce urmeaza se va prezenta modul in care descrierea conceptelor in limbajul  $L$  poate fi reprezentata pe un sir de biti de lungime fixa.

### 5.1. Ajustarea procesului de inductie folosind sabloane de limbaj

Totii algoritmi de invatare folosesc intr-un fel sau altul constrangeri, fie ele implicate sau explicite. O metoda larg raspandita in sisteme care integreaza inductia si deductia in structuri bazate pe logica predicatelor de ordinul I consta in folosirea unor constructii metasemantice pentru definirea unui set de formule admisibile ce urmeaza a fi explorate. Algoritmul propus limiteaza spatiul ipotezelor folosind un sablon de limbaj. Neformal, un sablon de limbaj este o formula  $\Lambda$  apartinand limbajului  $L$ , astfel incat fiecare descriere de concept conjunctiva admisibila poate fi obtinuta din  $\Lambda$  prin stergerea unor constante din disjunctiile interne care apar in ea.

Inainte de a da o definitie formală a lui  $\Lambda$ , sa reamintim cateva notiuni logice introduse mai sus, pentru a defini conceptul de *forma completa* pentru un termen disjunctiv si pentru un predicat. Dandu-se o formula  $\varphi(x_1, x_2, \dots, x_n)$ , a evalua valoarea de adevar a formulei intr-un univers  $U$  inseamna sa se caute o legatura intre variabilele  $x_1, x_2, \dots, x_n$  si niste constante  $a_1, a_2, \dots, a_n$ , definite in  $U$ , astfel incat  $\varphi(a_1, a_2, \dots, a_n)$  sa fie adevarat. De notat faptul ca in invatarea supervizata a conceptelor fiecare instanta de invatare constituie un univers specific. Putem da urmatoarea definitie:

Definitia 2: Fie un predicat  $P(x_1, x_2, \dots, x_n, \mathbf{K})$ , unde  $\mathbf{K}$  este o disjunctie pozitiva de constante  $[v_1, v_2, \dots, v_n]$ . Se spune ca  $P$  este in *forma completa* daca multimea  $[v_1, v_2, \dots, v_n]$  este in asa fel aleasa incat  $P$  poate fi satisfacut pentru orice legare a variabilelor  $x_1, x_2, \dots, x_n$ .

Definitia 2 stabileste faptul ca un predicat  $P$  care contine un termen disjunctiv in forma completa este adevarat pentru orice instanta a setului de invatare. Considerand din nou predicatul *culoare*( $x, \mathbf{K}$ ), acesta va fi in forma completa daca  $\mathbf{K}$  este multimea tuturor culorilor posibile. Pentru a distinge predicatul in forma completa de celelalte, acestea vor fi scrise cu caractere cursive.

Definitia 3: Un sablon de limbaj  $\Lambda$  este o formula a limbajului  $L$  care contine cel putin un predicat in forma completa.

Scopul unui sablon  $\Lambda$  este de a defini problema invatarei pentru algoritmul propus. Predicatul ce apar in  $\Lambda$  si nu sunt in forma completa au rolul de constrangere si trebuie sa fie satisfacute de o alegere specifica a legarii variabilelor din  $\Lambda$ . Dimpotriva, predicatul in forma completa (adevarate prin definitie) sunt folosite pentru a defini spatiul de cautare care caracterizeaza problema de invatare. Stergerea unei constante dintr-un termen complet pozitiv ce apare intr-un predicat  $P$  face ca predicatul  $P$  sa fie mai specific. Mai general, orice formula obtinuta prin renuntarea la unele constante din  $\Lambda$  este mai specifica decat insusi  $\Lambda$ . Dandu-se un sablon  $\Lambda$ , spatiul de cautare explorat de algoritm este restrans la multimea  $H(\Lambda)$  a formulelor care pot fi obtinute prin stergerea unor constante din termenii completi ce apar in  $\Lambda$ .

Este usor de observat faptul ca, avand o formula conjunctiva  $\Lambda$ , ea poate fi scrisa ca o conjunctie de doua subformule:  $\Lambda_n$ , compusa din predicatul necomplet si  $\Lambda_c$ , compusa din predicatul complet. Sablonul trebuie declarat la inceput prin specificarea separata a lui  $\Lambda_n$  si a lui  $\Lambda_c$ . In particular, formula  $\Lambda_n$  poate fi goala, in timp ce  $\Lambda_c$  trebuie sa contina intotdeauna cel putin un predicat in forma completa.

Datorita faptului ca multimea de constante necesare sa completeze o expresie disjunctiva poate fi una foarte mare (chiar infinita), s-a introdus un simbol constant special "\*" in  $\Lambda$ , cu rol de wildcard. Fie  $\mathbf{K} = [v_1, v_2, \dots, v_n]$  un termen complet; semnificatia constantei "\*" este definita implicit ca  $* = \neg [v_1, v_2, \dots, v_n]$ . Cu alte cuvinte, inseamna "orice alta valoare care nu este mentionata explicit in termenul disjunctiv in care apare". Semantica asociata simbolului "\*" este locala termenului disjunctiv in care apare si este definita static la declararea sablonului. Dupa aceasta isi mentine semnificatia initiala in toti termenii disjunctivi derivati

din acelasi termen in forma completa. Un exemplu de sablon de limbaj se gaseste in Figura 1, alaturi de doua formule apartinand spatiului de ipoteze asociat.

Sablonul $\Lambda$
$\Lambda = \text{greutate}(x, [3, 4, 5]) \wedge \text{culoare}(x, [\text{rosie}, \text{albastra}, *]) \wedge$ $\wedge \text{forma}(x, [\text{patrata}, \text{triunghiulara}, \text{circulara}, *]) \wedge$ $\wedge \text{distanta}(x, y, [1, 2, 3, 4, 5, *])$
$\Lambda_n = \text{greutate}(x, [3, 4, 5])$
$\Lambda_c = \text{culoare}(x, [\text{rosie}, \text{albastra}, *]) \wedge \text{forma}(x, [\text{patrata}, \text{triunghiulara}, \text{circulara}, *]) \wedge$ $\wedge \text{distanta}(x, y, [1, 2, 3, 4, 5, *])$
$\Phi_1 = \text{greutate}(x, [3, 4, 5]) \wedge \text{culoare}(x, [\text{rosie}]) \wedge \text{forma}(x, \neg [\text{patrata}, \text{circulara}]) \wedge$ $\wedge \text{distanta}(x, y, [1, 2])$
$\Phi_2 = \text{greutate}(x, [3, 4, 5]) \wedge \text{culoare}(x, \neg [\text{rosie}]) \wedge \text{distanta}(x, y, [1, 2])$

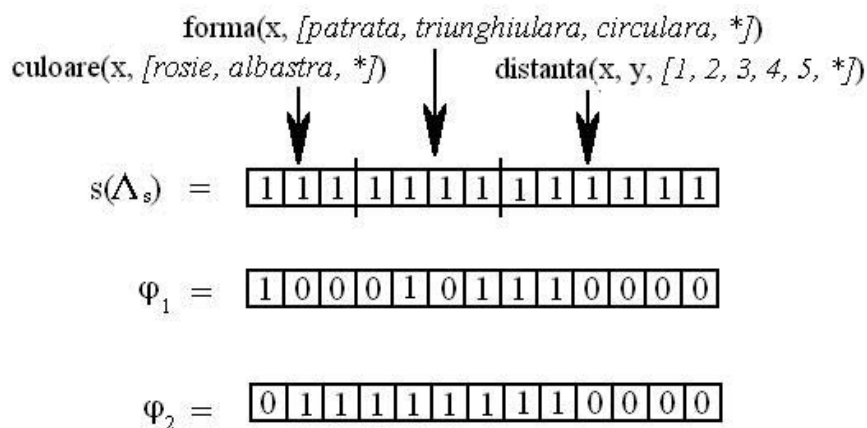
**FIGURA 2** – Exemplu de sablon de limbaj  $\Lambda$  cu subformulele sale  $\Lambda_n$  si  $\Lambda_c$ . Formulele  $\phi_1$  si  $\phi_2$  apartin spatiului  $H(\Lambda)$ . Toate formulele se considera implicit a fi cuantificate existential.

Este demn de mentionat faptul ca folosirea constantei wildcard “\*” duce natural la introducerea de termeni disjunctivi negati. De exemplu, predicatul  $\text{forma}(x, \neg [\text{patrata}, \text{circulara}])$ , care apare in  $\phi_1$ , este doar o rescriere a predicatului  $\text{forma}(x, [\text{triunghiulara}, *])$ , unde  $*$  =  $\neg [\text{patrata}, \text{triunghiulara}, \text{circulara}]$ , dupa definitia din sablon. Mai general, “\*” poate fi intotdeauna eliminata din orice ipoteza inductiva prin introducerea unui termen negat. In sfarsit, se mai observa faptul ca se poate renunta la predicatele complete care apar in orice ipoteza inductiva fara a i se modifica semnificatia, fiind o tautologie. De exemplu, in formula  $\phi_2$  s-a renuntat la predicatul complet  $\text{forma}(x, [\text{patrata}, \text{triunghiulara}, \text{circulara}, *])$ .

## 5.2. Transformarea formulelor din logica cu predicate de ordinul I in siruri de biti

O ipoteza inductiva  $\phi$ , apartinand spatiului  $H(\Lambda)$ , corespunde unei reguli de implicatie  $\phi \rightarrow h$ ,  $h$  fiind numele conceptului tinta. Dupa cum se va arata in cele ce urmeaza, formulele din  $H(\Lambda)$  pot fi usor reprezentate ca siruri de biti de lungime fixa. Se poate observa faptul ca doar predicatele complete din  $\Lambda_c$  trebuie sa fie procesate, in timp ce subformula  $\Lambda_n$  trebuie sa apara, implicit, in orice ipoteza inductiva, astfel ca ea nu mai are nevoie sa fie reprezentata. Transformarea unei formule  $\Lambda_c$  intr-un sir de biti de lungime fixa  $s(\Lambda_c)$  este imediata, fiecare constanta ce apare intr-un termen in forma completa fiind asociat unui bit in  $s(\Lambda_c)$ . Pastrand adiacenti bitii corespunzatori literalilor care sunt adiacenti in sablon, fiecare predicat in forma completa va corespunde unui substring specific si, de aceea, decodificarea unui sir de biti este directa.

Reprezentarea semantica a "genelor" in sirul de biti sa facut dupa cum urmeaza: daca bitul corespunzator unui termen  $v$  dat intr-un predicat  $P$  este positionat pe 1, atunci  $v$  apartine disjunctiei interne curente a lui  $P$ , pe cand, daca este positionat pe 0, nu apartine acelei disjunctii a lui  $P$ . Deoarece eliminarea unui termen dintr-o disjunctie interna a unui predicat introduce constrangeri, adica il face mai specific, operatia de transformare a unui 1 in 0 intr-un sir de biti se va numi specializare. Dintr-un motiv contrar, operatia de transformare a unui 0 in 1 intr-un sir de biti se va numi generalizare. Exemple de siruri de biti care codifica formulele din Figura 1 sunt date in Figura 2.



**FIGURA 2** – Sirurile de biti corespunzatoare sablonului si formulelor si descrise in Figura 1. Fiecare termen, inclusiv ‘\*’, care apare intr-o disjunctie interna are un bit corespondent in sirul de biti.

Sirul de biti caracterizeaza in mod unic o ipoteza inductiva  $\varphi$  si se constituie in informatia procesata de algoritmul genetic. De fapt, in REGAL un individ este reprezentat de o structura mai complexa care contine si informatii procesate de procedura de evaluare. In particular, acesta contine un sir de biti  $F$  care specifica multimea de exemple ce satisfac  $\varphi$ , clasa  $h$  asignata formulei  $\varphi$  si valoarea de fitness  $f$ . In faza de evaluare se incearca potrivirea fiecarui individ nou generat cu toate exemlele de invatare si se umple sirul de biti  $F$ . Dupa aceasta se asigneaza clasa  $h$ : daca exista un singur concept tinta, clasa este univoc determinata, altfel sistemul asigneaza conceptul care are mai multi reprezentanti in instantele de invatare satisfacute de  $\varphi$ . In final este evaluata valoarea fitness-ului,  $f$ , tinand cont de calsa  $h$  asignata.

## 6. Operatorii genetici

În secțiunea ce urmează se vor descrie operatorii genetici și funcția de evaluare a indivizilor (fitness) folosiți în REGAL. Sistemul folosește operatori de selecție, de încrucișare, de mutație și de înmulțire. Cele mai relevante diferențe față de un algoritm genetic clasic se află în operatorii de selecție și de înmulțire; operatorii de încrucișare și de mutație sunt mult mai apropiați de cei clasici.

### 6.1. Operatorii de încrucișare și de mutație

Operatorul de încrucișare este special proiectat pentru problema în discuție și constă din: încrucișare din două puncte, încrucișare uniformă, încrucișare prin generalizare și încrucișare prin specializare. Acest operator se aplică între doi indivizi din populație (adică siruri de biți din reprezentarea a două formule  $\varphi$  și  $\psi$ ).

Incrucisarea din două puncte funcționează după cum urmează: dându-se două siruri de biți (siruri părinți), încrucisarea produce urmași selectând aleator două puncte în sirul de biți și interschimbând apoi biții care se află între aceste două puncte. În acest fel se schimbă o secvență aleatoare de biți între doi indivizi.

Operatorul de încrucișare uniformă alege mai întâi în mod aleator un set de poziții de biți și apoi produce urmași prin interschimbarea valorilor biturilor corespunzătoare din cei doi părinți.

Incrucisarea prin generalizare și prin specializare necesită explicații suplimentare. După cum s-a descris în Figura 2, sirul  $s(\Lambda_s)$  este împărțit în subsiruri, fiecare din ele corespunzând unui predicat specific  $P$ . În ambele tipuri de încrucișare, o mulțime  $D$  de predicate este selectată aleator din  $\Lambda_s$ . Incrucișarea prin specializare (generalizare) funcționează după cum urmează:

Subsirurile din sirurile părinți  $s_1$  și  $s_2$ , corespunzătoare predicatelor neselectate în mulțimea  $D$ , sunt copiate neschimbate în urmașii corespunzători  $s_1'$  și  $s_2'$ .

Pentru fiecare predicat  $P \in D$ , un nou subsir  $s_i'$  este generat prin operația SI (SAU) logic între biții corespunzători subsirurilor  $s_1(P_i)$  și  $s_2(P_i)$ . Apoi se copiează subsirurile  $s_i'$  atât în  $s'$  cât și în  $s_2'$ .

Multimea D se creeaza asociind fiecarui predicat R o probabilitate egala de a fi selectat. Dupa cum s-a explicat in sectiunea !!!, transformand un 0 in 1 intr-un sir de biti se adauga un termen unei disjunctii interne ce apare in ipoteza inductiva corespunzatoare, aceasta devenind mai generala, adica mai putin constransa. De aici si notiunea de incrucisare prin generalizare, deoarece unramssi care se genereaza sunt mai generali decat parintii lor. Printr-un rationament asemanator se poate explica incrucisarea prin specializare.

In cele ce urmeaza vom nota cu **E**, respectiv cu **C**, multimea exemplorlor pozitive, respectiv negative, ale conceptului tinta, si cu E si C cardinalitatea lor. Fie o pereche de siruri ( $s_1, s_2$ ), generata de procedura de imperechere. Incrucisarea se va aplica cu o probabilitate pc. Apoi se va selecta stochastic modul de incrucisare tinand cont de caracteristicile celor doua siruri. Probabilitatile conditionale  $p_u$  pentru incrucisarea uniforma,  $p_{2pt}$  pentru incrucisarea prin doua puncte,  $p_s$  pentru incrucisarea prin specializare si  $p_g$  pentru incrucisarea prin generalizare se asociaza dupa cum urmeaza:

$$\begin{aligned} p_u &= (1 - a \cdot f_n) \cdot b \\ p_{2pt} &= (1 - a \cdot f_n) \cdot (1 - b) \\ p_s &= a \cdot f_n \cdot r \\ p_g &= a \cdot f_n \cdot (1 - r) \end{aligned} \quad (5.1)$$

In expresia (4.1) a si b iau valori in intervalulu [0, 1] si sunt parametri ajustabili,  $f_n$  este valoarea medie normalizata a fitnessului celor doua formule  $\varphi_1$  si  $\varphi_2$ , definite de sirurile  $s_1$  si  $s_2$ :

$$f_n = \frac{f(\varphi_1) + f(\varphi_2)}{2f_{Max}} \leq 1 \quad (5.2)$$

si r este raportul:

$$r = \frac{[n^+(\varphi_1) + n^-(\varphi_1) + n^+(\varphi_2) + n^-(\varphi_2)]}{(E + C)^2} \quad (5.3)$$

unde  $n^+(\varphi)$  si  $n^-(\varphi)$  reprezinta numarul instantelor pozitive, respectiv negative, acoperite de  $\varphi$ .

Intr-un caz specific, valoarea medie normalizata  $f_n$  determina probabilitatea de selectie intre incrucisarea uniforma si cea prin doua puncte,  $p_u + p_{2pt} = 1 - af_n$ , pe de o parte, sau intre incrucisarea prin generalizare si cea prin

specializare,  $p_s + p_g = af_n$ , pe de alta. Cand  $s$  si  $s_2$  au valori mici ale fitness-ului, adica inca nu sunt ipoteze inductive bune, sansa de a aplica primele doua tipuri de incrucisari este mai mare pentru a exploata putere lor mai mare de explorare. Dimpotriva, valori mari ale fitness-ului privilegiaza folosirea incrucisarii prin specializare si prin generalizare pentru a rafina ipoteza inductiva. Alegerea intre incrucisarea uniforma si cea prin doua puncte este controlata static de parametrul  $b$ , pe cand alegerea intre incrucisare prin generalizare si cea prin specializare este controlata de parametrul  $r$  care, la randul sau, este evaluat in timp real pe baza exemplurilor acoperite de  $s_1$  si de  $s_2$ . Astfel, cand acestea acopera multe exemple, este preferata incrucisarea prin specializare pentru a mari sansele de a crea un urmas mai consistent, altfel incrucisarea prin generalizare primeste o sansa mai mare.

In ceea ce priveste operatorul de mutatie, acesta este identic cu cel folosit in algoritmi genetici clasici. El este aplicat urmasilor cu probabilitatea  $p_m = 0.00001$  si poate afecta orice bit din  $s(\Lambda_s)$ .

**Exemplu de operatii genetice – Pentru a clarifica modul in care operatorii genetici functioneaza vom da un exemplu. Fie  $\Lambda$  sablonul de limbaj din Figura 1 si formulele:**

$$\begin{aligned} \Phi_1 &= \text{greutate}(x, [3, 4, 5]) \wedge \text{culoare}(x, [\text{rosie}, \text{albastra}]) \wedge \\ &\wedge \text{forma}(x, \neg [\text{patrata}, \text{circulara}]) \wedge \\ &\quad \wedge \text{distanta}(x, y, [1, 2]) \\ \Phi_2 &= \text{greutate}(x, [3, 4, 5]) \wedge \text{culoare}(x, \neg [\text{albastra}]) \wedge \\ &\wedge \text{distanta}(x, y, [1, 2, 3]) \end{aligned}$$

Atunci, aceste formule pot fi transformate, prin sablonul , in indivizii:

$$\begin{aligned} i_1 &= 1100101110000 \\ i_2 &= 1011111111000 \end{aligned}$$

In Tabelul 1 este descris efectul operatorilor de incrucisare asupra acestor doi indivizi.

	Parinti	Formule
	1100101110000	culoare(x, [rosie, albastra]), forma(x, $\neg$ [patrata, circulara])
	1011111111000	distanta(x, y, [1, 2]) culoare(x, $\neg$ [albastra]), distanta(x, y, [1, 2, 3])

Xover	Urmasi	Formule
$u_{2,6,7,9}$	<u>1000111110000</u> <u>1111101111000</u>	culoare(x,[rosie]), forma(x,[triunghiulara, circulara, *]), distanta(x, y, [1,2]) culoare(x, [patrata, triunghiulara, *]), distanta(x, y, [1, 2, 3])
$2p_{5-8}$	0100 <u>111110000</u> 1011 <u>101111000</u>	culoare(x, [albastra]), distanta(x, y, [1, 2]) culoare(x, [rosie, *]), forma(x, [patrata, triunghiulara, *]), distanta(x, y, [1, 2, 3])
$s_{1-3,4-7}$	<u>1000101110000</u> <u>1000101111000</u>	culoare(x, [rosie]), forma(x, [triunghiulara, *]), distanta(x, y, [1, 2]) culoare(x, [rosie]), forma(x, [triunghiulara, *]), distanta(x, y, [1, 2, 3])
$g_{4-7,8-13}$	110 <u>1111111000</u> 101 <u>1111111000</u>	culoare(x, [rosie, albastra]), distanta(x, y, [1, 2, 3]) culoare(x, [rosie, *]),distanta(x, y, [1, 2, 3])

**Tabelul 1** – Exemplu de operatii de incrucisare.  $u$  si  $2p$  semnifica incrucisarea uniforma si cea prin doua puncte, pe cand  $s$  si  $g$  insemna incrucisare prin specializare si prin generalizare. Indicii operatorilor exprima care dintre biti au fost alesi de operatori.

Operatorii din Tabelul 1 sunt scrisi in forme de genul  $s_{1-3,4-7}$ , unde  $s$  exprima operatorul, in acest caz cel de incrucisare prin specializare, iar indicii denota bitii afectati de operatorul de incrucisare. Se observa ca in cazul incrucisarii prin doua puncte sunt afectati biti consecutivi, iar in cazul incrucisarii prin specializare si prin generalizare bitii afectati corespund subsirurilor predicatelor alese. De exemplu,  $s_{1-3,4-7}$ , corespunde operatorului de incrucisare prin specializare aplicat primelor doua predicate din sablomul de limbaj.

## 6.2. Operatorul de inmultire

Dupa cum se va explica mai jos, operatorul de selectie folosit de REGAL favorizeaza ipoteze ce acopera mai multe instante in setul de invatare. Daca o instanta anume nu e acoperita inca, va fi foarte probabil ca acesta sa genereze dinamic indivizi care sa o acopere. Aceasta facilitate este oferita de operatorul de inmultire, care actioneaza ca o functie care, primind o instanta pozitiva, intoarce o formula ce o acopera. Descrierea abstracta a algoritmului de inmultire este urmatoarea:

### ***Inmultire*** ( $\xi$ )

Fie  $\xi \in \mathbf{E}$  o instanta pozitiva de invatare  
Genereaza aleator un sir de biti  $s$  care defineste o formula  
 $\varphi \in H(\Lambda)$   
Selecteaza aleator o legare  $\mathbf{b}$  pentru  $\xi$  care e compatibila cu  $\Lambda_n$   
Pozitioneaza pe  $1$  cel mai mic numar de biti din  $s$  pentru a  
satisface  $\varphi$  pe  $\xi$   
**intoarce** ( $\varphi$ )

**FIGURA 3** – Descrierea abstracta a algoritmului de inmultire.

Ca un exemplu, sa consideram din nou sablonul de limbaj din Figura 1 si sa presupunem ca exemplul  $\xi$  de acoperit consta din urmatoarea secventa de trei obiecte:

$\alpha_1$ :<culoare = albastra, pozitie = 1, greutate = 4, forma = patrata>  
 $\alpha_2$ :<culoare = verde, pozitie = 2, greutate = 1, forma = patrata>  
 $\alpha_3$ :<culoare = albastra, pozitie = 5, greutate = 10, forma = circulara>

Sa mai presupunem ca legarea  $\mathbf{b}$ ,  $\langle x = \alpha_1; y = \alpha_3 \rangle$ , satisface constrangerea greutate( $x$ , [3, 4, 5]). Fie sirul  $s = "10000110101001"$ , corespunzator formulei selectate  $\varphi = \text{culoare}(x, [\text{rosie}]) \wedge \text{forma}(x, [\text{triunghiulara}, \text{circulara}]) \wedge \text{distanta}(x, y, [1, 2, *])$ . Formula  $\varphi$  nu este satisfacuta de  $\xi$ . De aceea, sirul  $s$  va fi modificat in  $s' = "1101110101001"$ , corespunzator formulei  $\varphi' = \text{culoare}(x, [\text{rosie}, \text{albastra}]) \wedge \text{forma}(x, [\text{patrata}, \text{triunghiulara}, \text{circulara}]) \wedge \text{distanta}(x, y, [1, 2, *])$  care este acum satisfacuta de  $\xi$ .

Dovedindu-se ca este asigurata compatibilitatea cu constrangerile impuse de sablonul  $\Lambda_n$ , algoritmul de inmultire incearca sa introduca un factor aleator cat mai mare posibil pentru a mari diversitatea genetica a populatiei. Acesta este folosit la inceputul cautarii pentru a initializa populatia si apoi, este apelat fie de operatorul de selectie, fie ca o alternativa la operatorul clasic de mutatie.

### 6.3. Operatorul de selectie "sufragiu universal"

Operatorul de selectie este special croit pentru problema invatarii conceptelor. Ideea de baza poate fi explicata printr-o metafora. Formulele conjunctive se constituie in "candidati" spre a fi alesi intr-un parlament (populatia), unde exemplele pozitive de invatare sunt electori; un exemplu poate vota pentru una din formulele pe care le acopera. Marea diferenta dintre acest operator si altii propusi pana in momentul de fata este faptul ca indivizii ce vor fi imperecheati nu sunt alesi direct din populatia curenta, ci indirect prin selectarea unui numar egal de exemple pozitive, dupa cum se va descrie mai jos.

Fie populatia  $A(t) = \{\varphi_1^{x_1(t)}, \dots, \varphi_m^{x_m(t)}\}$  la momentul  $t$  o multime de cardinalitate  $M$ .  $A(t)$  contine  $m$  indivizi diferiti (adica formule conjunctive ale limbajului de descriere  $L$ ), fiecare dintre ele avand multiplicitatea  $x_j(t)$  ( $1 \leq j \leq m$ ). Fie  $COV(\varphi_j)$  submultimea lui  $\mathbf{E}$  acoperita de  $\varphi_j$ . Procedura de selectie functioneaza dupa cum urmeaza: La fiecare generatie  $t$ , un numar  $g * M \leq M$  de exemple este ales aleator din setul de exemple pozitive  $\mathbf{E}$ . Parametrul  $g$  reprezinta proportia de indivizi selectati pentru imperechere. Fiecarui exemplu selectat  $\xi_k$  i se asociaza o multime  $\mathbf{R}(\xi_k)$  continand formulele din  $A(t)$  care acopera exemplul  $\xi_k$  asociat. Multimea  $\mathbf{R}(\xi_k)$  corespunde unei "roti de ruleta"  $r_k$ , impartita in sectoare, fiecare sector fiind asociat unei formule  $\varphi_j \in \mathbf{R}(\xi_k)$ . Marimea fiecarui sector asociat unei formule  $\varphi_j$  este proportional cu raportul dintre valoarea totala a fitness-ului formulei (adica fitness-ul lui  $\varphi_j$  inmultit cu multiplicitatea sa,  $x_j(t)$ , in  $A(t)$ ) si suma valorilor totale ale fitness-ului ale tuturor formulelor ce apar in  $\mathbf{R}(\xi_k)$ . Pentru fiecare rotire a ruletei  $r_k$  se alege o formula invingatoare. Iata un algoritim mai formal al celor prezentate:

**Selectia prin sufragiu universal**

Fie  $\mathbf{B}(t) = \emptyset$

Selecteaza aleator  $g * M$  exemple din  $\mathbf{E}$

**pentru** fiecare exemplu  $\xi_k$  selectat **executa**

**daca**  $\mathbf{R}(\xi_k) \neq \emptyset$  **atunci**

Roteste  $r_k$  si adauga formula castigatoare la  $\mathbf{B}(t)$

**altfel**

Creeaza o noua formula  $\psi$  ce acopera  $\xi_k$  prin aplicarea operatorului de inmultire si adauga  $\psi$  la  $\mathbf{B}(t)$ .

**sfarsit**

**FIGURA 4** – Descrierea abstracta a algoritmului de selectie folosit.  $M$  este cardinalitatea globala a populatiilor,  $g \in [0, 1]$  este un parametru numit *bresa dintre generatii*.

Figura 5 exprima o reprezentare grafica a ciclului de baza din REGAL. In conformitate cu metafora "parlamentului", la fiecare generatie  $g * M$  exemple pozitive trebuie sa-si exprime preferinta, prin "votarea" unuia dintre "reprezentativii" lor invarbind propria roata a ruletei. E important de observat faptul ca doar formulele care acopera aceleasi exemple concureaza intre ele si ca exemplele "voteaza" (stochastic) pentru cea mai buna dintre ele. Procesul de selectie favorizeaza formule cu acoperire mai mare (formule ce apar in mai multe roti de ruleta) si formule cu o valoare mai mare a fitness-ului total (formule cu o probabilitate mai mare de a castiga in ruleta in care apar).

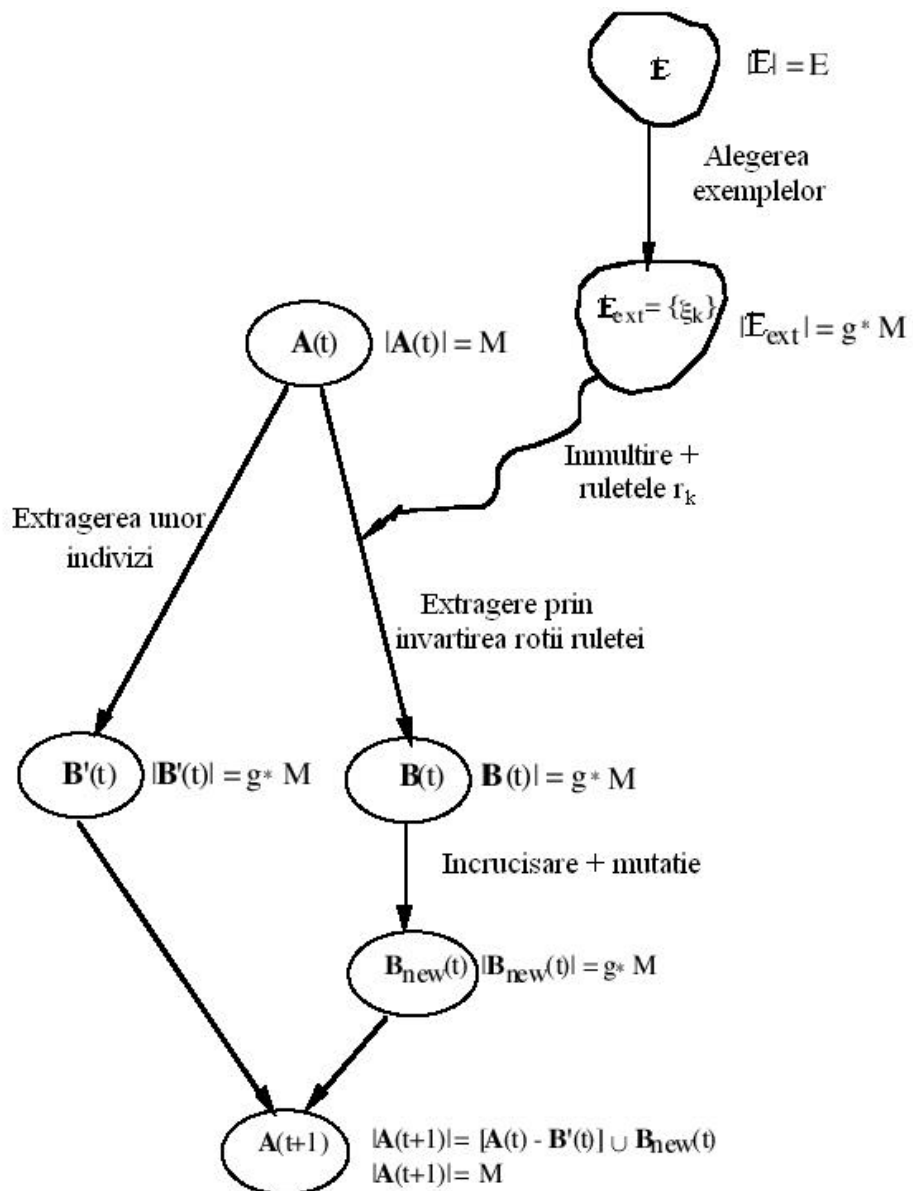


FIGURA 5 – Reprezentarea grafica a ciclului de baza al algoritmului

Numele de sufragiu universal deriva din faptul ca fiecare exemplu are aceeasi probabilitate de a fi extras. Mai precis, procesul extragerii exemplurilor  $\xi_k$  din  $\mathbf{E}$  la fiecare generatie, urmeaza o distributie binomiala cu probabilitatea de succes de  $1/E$  si cu numarul de incercari egal cu  $g * M$ . Deoarece procesul de selectie este eacelasi la fiecare generatie, probabilitatea de distributie a numarului de selectii ale exemplului  $\xi_k$  in  $t$  generatii este tot una binomiala, cu  $g * M * t$  incercari si cu aceeasi probabilitate de succes de  $1/E$ .

Considerand problema inversa a esantionarii lui Bernoulli, este usor de demonstrat ca pentru un exemplu generic  $\xi_k$  exista o probabilitate mai mare decat  $\eta$  de a fi extras in primele  $t$  generatii.

$$M \geq \frac{1}{g^t} \frac{\lg\left(\frac{1}{\eta}\right)}{\lg\left(\frac{E}{E-1}\right)} \quad (5.4)$$

Din relatia (5.4) deducem ca cel mai favorabil caz se obtine pentru  $g = 1$ . De fapt, cand intreaga populatie este innoita la fiecare generatie, un numar  $M$  mai mic este suficient pentru ca, dandu-se  $t$ , sa apara  $\xi_k$ .

Numarul mediu de generatii  $\overline{t_{tot}}$  necesar pentru a extrage toate exemplele este cel mult:

$$\overline{t_{tot}} = \frac{E^2}{gM} \quad (5.5)$$

Din nou, cea mai convenabila valoare este  $g = 1$ . Este clar faptul ca, in general, nu este necesar sa se execute toate cele  $E^2$  incercari, deoarece orice formula, care nu concide cu descrierea unui singur exemplu, va acoperi mai multe concepte in acelasi timp.

Procedul de selectie bazat pe operatorul de sufragiu universal poate fi descris formal print-o matrice stochastica ( $E \times M$ )  $\mathbf{P}(t)$ , ale carei linii corespund exemplurilor  $\xi_k$  din  $\mathbf{E}$ , si ale carei coloane corespund formulelor  $\varphi_j$  din  $\mathbf{A}(t)$ . Fiecare element  $p_{kj}(t)$  al matricii este egal cu probabilitatea ca formula  $\varphi_j$  sa castige in ruleta corespunzatoare  $r_k$ , dandu-se un exemplu extras  $\xi_k$ :

$$\{p_{kj}(t)\} = p_{kj}(t) = \frac{v_{kj} f_j x_j(t)}{\sum_{i=1}^m v_{ki} f_i x_i(t)} \quad (1 \leq k \leq E, 1 \leq j \leq m) \quad (5.6)$$

In relatia (4.6)  $f_j$  este valoarea fitness-ului formulei  $\varphi_j$ , iar  $v_{kj}$  este functia caracteristica a multimii  $\text{COV}(\varphi_j)$ :

$$v_{kj} = \begin{cases} 1, & \text{daca } \varphi_k \in \text{COV}(\varphi_j) \\ 0, & \text{altfel} \end{cases} \quad (1 \leq k \leq E, 1 \leq j \leq m) \quad (5.7)$$

Elementele  $p_{kj}$  sunt normalizate relativ la linia  $k$  a matricii  $P$ .

#### 6.4. Functia de fitness

In problema invatarii conceptelor criteriile luate in considerare pentru a evalua solutiile au fost, in mod traditional, completitudinea, consistenta si simplitatea. Daca primele doua concepte, completitudinea si consistenta, isi gasesc o definitie precisa in logica, cel al simplitatii nu este univoc acceptat si gaseste interpretari diferite in contexte diferite. Cel mai frecvent folosit inteles al acestuia este de simplitate sintactica, care se reduce la numararea numarului de literali necesari pentru a reprezenta formula. Dezavantajul il reprezinta faptul ca definitia depinde strict de limbajul de reprezentare. Mai mult, o aceeași formula poate fi scrisa in forme echivalente avand valori diferite ale simplitatii. In cazul algoritmului REGAL s-a adoptat o definitie computationala a simplitatii, corespunzatoare numarului de teste care trebuie efectuate pentru a se verifica daca o formula este satisfacuta de un exemplu. Acest lucru se face prin simpla verificare a faptului ca, pentru fiecare predicat din sablonul de limbaj, caracteristica asociata nu ia o valoare care nu apartine multimii de termeni din disjunctia interna corespunzatoare, mai exact se verifica daca este incalcată vreă conditie negata. Apoi, se obtine imediat o masura a complexitatii  $c(\varphi)$  pentru o formula  $\varphi$  prin numararea termenilor din sablon care nu apar in formula  $\varphi$  sau, echivalent, prin numararea zerourilor care apar sirul de biti  $s(\varphi)$ . Pornind de la aceasta masura a complexitatii, sa definit o masura a simplitatii  $z(\varphi)$  ca fiind raportul:

$$z(\varphi) = \frac{\text{lungimea}(s(\varphi)) - c(\varphi)}{\text{lungimea}(s(\varphi))} \quad (5.8)$$

Pentru consistenta s-a luat in considerare numarul de exemple negative acoperite de o formula  $\varphi$ :

$$w(\varphi) = n(\varphi) \quad (5.9)$$

Deoarece operatorul de selectie trateaza completitudinea prin mecanismul invarierii rotii ruletei, doar consistenta si simplitatea sunt luate in considerare in functia de fitness. Mai precis, se foloseste functia:

$$f(\varphi) = f(z, w) = (1 + Az) e^{-w} \quad (5.10)$$

In relatia (5.10), parametrul  $A$  este o valoare ajustabila de utilizator, valoarea sa curenta fiind  $A = 0.1$ . Se observa ca  $f_{\text{Max}} = 1.1$  si ca  $\lim_{w \rightarrow \infty} f(z, w) = 0$ . Un grafic al functiei  $f$  se afla in Figura 6.

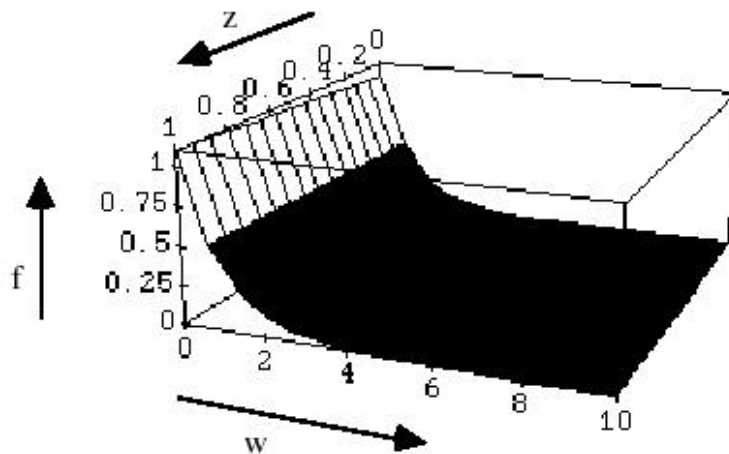


FIGURA 6 – reprezentarea grafica a functiei de fitness  $f(\varphi)$  in raport cu simplicitatea  $z$  si cu consistenta  $w$  a unei formule  $\varphi$ . Valori mari ale lui  $z$  ( $w$ ) corespund unor descresteri ale simplicitatii (consistentei).

Expresia (5.10) deriva din experimente cu diverse functii de fitness. Sa dovedit a oferi evaluari rezonabile, dar nu se poate spune ca se vrea a fi o alerege unica sau cea mai buna.

Cu privire la complexitatea operatorului de selectie, pentru a calcula valorile probabilitatilor din relatia (5.6) trebuie evaluat, pentru fiecare  $\varphi_j$ , un numar de sume cel mult egal cu numarul de rulete disponibile: deoarece exista cel mult o ruleta pentru fiecare exemplu din  $\mathbf{E}$ , atunci numarul de pasi executati este cel mult  $E \cdot m \leq E \cdot M$ . De aceea, complexitatea globala este de ordinul  $O(E \cdot M)$ . Aceasta complexitate este in contrast cu cea care se obtine folosind functii partajate, adica  $O(M^2 \cdot E)$ . Aceasta ultima valoare s-a obtinut pentru folosirea unei distante fenotipice la evaluarea acoperirii exemplurilor, pentru a face o comparatie corecta cu operatorul de selectie "sufragiul universal".

In final, trebuie evidentiat un fapt important in ceea ce priveste evaluarea fitness-ului si probabilitatea compararii: in REGAL nu exista notiunea de valoare medie globala a fitness-ului; de fapt, probabilitatile din (5.6) sunt normalizate

doar relativ la valoarea medie a fitness-ului indivizilor ce apartin aceleiasi rulete. Indivizii care nu apar in aceeasi ruleta nu sunt comparabili. Argumentam ca exact aceasta proprietate de "localitate" a fitness-ului este cea care determina abilitatea algoritmului REGAL de a mentine specii diferite in echilibru.

In [Giordana, Neri si Saitta, 1994] a fost propusa o metoda generala de investigare a proprietatilor macroscopice ale populatiilor in evolutie in conformitate cu o matrice de tranzitie a probabilitatilor. Aceasta metoda, bazata pe definirea unei "*populatii medii virtuale*", permite depasirea unor dificultati tehnice implicate in acst tip de analiza, oferind totusi in acelasi timp estimari precise ale parametrilor ce controleaza evolutia. Metoda este foarte potrivita pentru o varietate de abordari si in [Neri si Saitta, 1995] se pot gasi aplicatiile sale in mai multe metode de cautare.

## 7. Teoria formarii speciilor si a niselor

Formarea speciilor pare in mod special interesanta pentru invatarea de concepte. In primul rind ofera posibilitatea invatarii simultane atat a mai multor disjuncti cat si a mai multor concepte. Mai mult, resursele de calcul sunt exploatate mai eficient prin evitarea redundanțelor si a replicarii nefolositoare si prin exploatarea naturala a facilitatilor calculului distribuit. Metode propuse pentru formarea niselor si a speciilor includ "Crowding" [De Jong, 1975] si "Sharing Functions" [Goldberg & Richardson, 1987].

Crowding este o varianta a unui algoritm genetic simplu care inlocuieste indivizii vechi. Noii indivizi, generati prin incrucisare si mutatii, ii inlocuiesc pe cei vechi care sunt cei mai similari cu ei, corespunzator unei masuri date a similitudinii. In acest fel este posibil ca subpopulatiile sa creasca deoarece presiunea genetica tinde sa se manifeste in primul rand printre indivizii similari.

Metoda bazata pe functii partajate modifica probabilitatea de selectie cu scopul de a inhiba cresterea excesiva a presiunii genetice a unie subpopulatii. Acest lucru este obtinut prin reducerea valorii fitness-ului unui individ dupa numarul de indivizi asemanatori lui. In formularea initiala s-a luat in considerare partajarea genotipica. Valoarea fitness-ului  $f(\varphi)$ , ascoiat unui individ  $\varphi$ , a fost considerata ca o recompensa a mediului pentru a fi impartita cu alti indivizi. Similaritatea dintre doi indivizi,  $\varphi$  si  $\varphi'$ , a fost definita ca fiind distanta Hamming dintre sirurile de biti asociate lor. De asemenea, poate fi folosita si partajarea fenotipica, considerand distanta intre doi indivizi in domeniul lor semantic. Prin aplicarea atat a crowding-ului cat si a functiilor partajate asupra unui set de test

de probleme de optimizare, s-a observat ca crowding-ul a fost mai putin efectiv, deoarece nu poate gasi tot timpul fitness-ul maxim. Metoda s-a dovedit a functiona bine pentru probleme de invatare a mai multor concepte unimodale la un moment dat, dar nu a fost capabila sa permita o formare stabila a subpopulatiilor, reprezentative definitiilor disjuncte ale aceluiasi concept. In toate experimentele efectuate, pe termen lung, un singur disjunct a iesit ca fiind "cel ma bun".

Partajarea fenotipica sau genotipica functioneaza bine in spatii vectoriale, unde notiunea de distanta are o semantica clara. Deb si Goldberg au gasit o diferenta de performanta intre distanta fenotipica si cea genotipica, in favoarea celei fenotipice: folosind ca distanta genotipica definitia lui Hamming, algoritmul genetic a avut o comportare intermediara intre crowding si partajare fenotipica, avand abilitatea de a gasi toate maximele existente.

## 8. Concluzii

In aceasta lucrare s-a prezentat o metoda de abordare baza pe un algoritm co-evolutiv pentru rezolvarea problemelor si in particular pentru problema acoperirii optime a sirurilor [Forrest et al., 1993]. Credem ca am reusit sa aratam in aceasta lucrare faptul ca sistemul propus este capabil sa lucreze in domenii complexe si se poate descurca cu seturi mari de date, putand concura cu sisteme asemanatoare. Oricum, comportamentul sistemului si potentialul acestuia poate ridica mai multe intrebari decat raspunsuri.

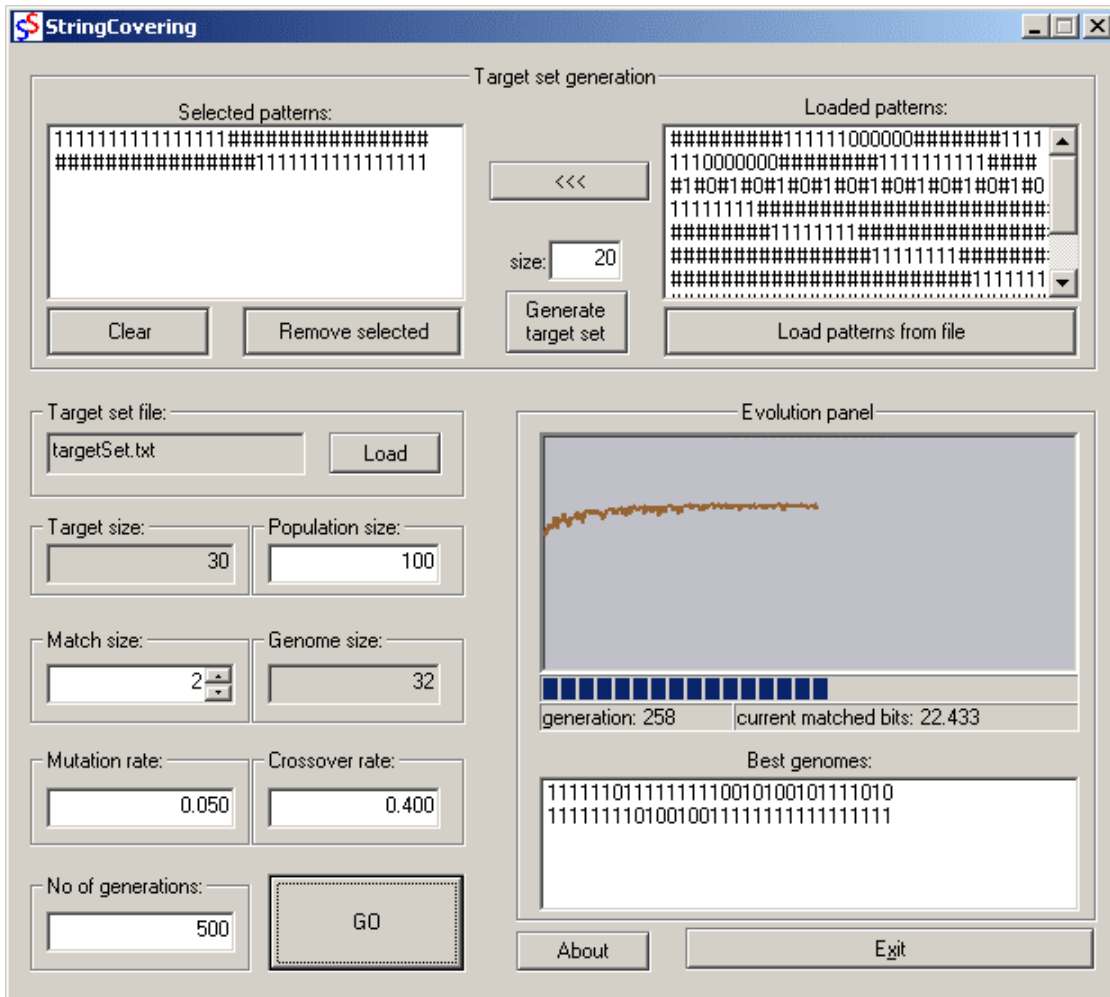
Un aspect important este rolul pe care algoritmiile genetice il pot juca in problemele de invatare automata (machine learning). Suntem de acord cu faptul ca problema invatarii trebuie sa exploateze cat mai mult posibil cunostintele de baza specifice domeniului, incercand sa limiteze astfel cautarea in spatiul ipotezelor prin constrangeri apriorice. De fapt, in acest caz algoritmiile de cautare "simbolica" pot fi mult mai convenabili, atat din cauza ca ei pot exploata chiar acea informatie apriorica, cat si din cauza ca au nevoie de resurse computationale mai reduse atunci cand efectueaza cautari in spatii restranse. Pe de alta parte, atunci cand sunt disponibile putine cunostinte, sau cand acestea nu se conformeaza constrangerilor, spatiul de cautare ramane vast. Totusi, credem ca algoritmiile genetice au potentialul de a deveni abordarea castigatoare a problemei, datorita puterii de explorare pe care o ofera fara a cere reducerea limbajului de reprezentare a ipotezelor.

## Bibliografie

- Hugues Juille, Jordan B. Pollack**(1995)  
*Semantic Niching and Coevolution in Optimization Problems*
- Mitchell A. Potter, Kenneth A. De Jong** (2000)  
*Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents*
- S. A. Kauffman and S. Johnsen** (1991)  
*Co-evolution to the edge of chaos: Coupled fitness landscapes, poised states, and co-evolutionary avalanches*
- M. A. Potter, K. A. De Jong and J. J. Grefenstette** (1995)  
*A coevolutionary approach to learning sequential decision rules*
- Eric V. Siegel** (1999)  
*Competitively Evolving Decision Trees Against Fixed Training Cases for Natural Language Processing*
- S. Forrest, B. Javornik, R.E. Smith and S.A. Perelson** (1993)  
*Using genetic algorithms to explore pattern recognition in the immune system*
- Mitchell A. Potter and K. A. De Jong**(1994)  
*A cooperative coevolutionary approach to function optimization*
- H. P. Schwefel** (1995)  
*Evolution and Optimum Seeking*
- J. P. Rosca and D. H. Ballard** (1994)  
*Hierarchical self-organization in genetic programming*
- R. E. Smith, S. Forrest and S. Perelson** (1993)  
*Searching for diverse, cooperative populations with genetic algorithms*

Anexe

## A. Imagini ale aplicatiei



## **B. Prezentarea aplicatiei (slide-show)**

# Algoritmi genetici pentru rezolvarea problemelor prin evolutie si co-evolutie

Coordonator proiect:

Prof. dr. ing. **Adina Magda Florea**

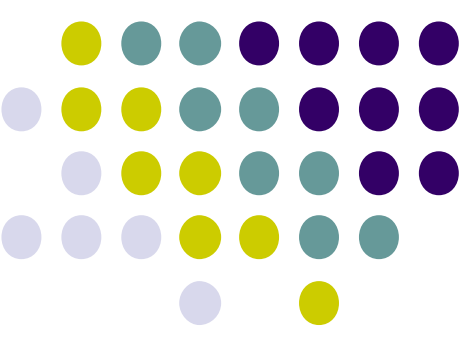
Absolvent:

**Sorin OSTAFIEV**

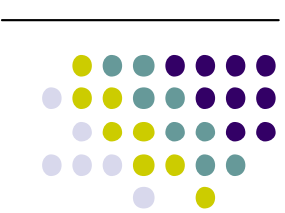
UNIVERSITATEA POLITEHNICA BUCURESTI

FACULTATEA DE AUTOMATICA SI CALCULATORARE

2003

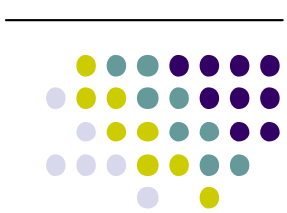


# Scopul si motivatia



- Utilizarea algoritmilor evolutivi pentru rezolvarea problemelor din ce in ce mai complexe necesita oportunitati pentru ca sub-problemele rezolvate individual sa interactioneze
- Este prezentata o arhitectura pentru co-evolutia unor astfel de componente ca o colectie de specii cooperante

## De ce algoritmi evolutivi clasici nu sunt suficient de buni?



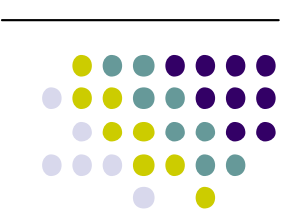
- Populatiile de indivizi au o puternica convergenta
  - se inlatura persistenta pe termen lung a sub-componentelor diverse
- Indivizii reprezinta solutii complete evaluate individual, in izolare
  - nu exista ocazii pentru aparitia co-adaptarii
- Cum ar trebui sa fie sub-componentele? Cate ar trebui sa fie?

# Pentru extinderea modelului clasic trebuie sa avem in vedere urmatoarele aspecte



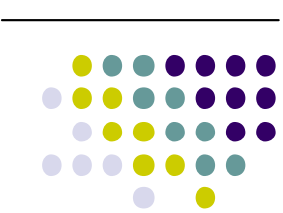
- **Descompunerea problemei**
  - de la inceput sau ca urmare a evolutiei algoritmului
- **Dependentele dintre sub-componente**
  - co-adaptarea poate aparea doar daca interactiunile sunt modelate
- **Asocierea rolurilor**
  - Determinarea contributiei fiecărei sub-componente la întreaga soluție
- **Mentineră diversitatii**
  - Diversitatea trebuie păstrată mai mult decât în cazul algoritmilor evolutivi clasici

# Funcția de fitness



- Fiecare specie evoluează în propria populație și se adaptează mediului prin intermediul algoritmului evolutiv
- Indivizii dintr-o specie sunt evaluați între ei prin colaborarea cu indivizii “reprezentativi” din alte specii

# Exemple



- **Maximizarea unei functii  $f(x)$  cu  $n$  variabile independente ( $n$  specii, cate una pentru fiecare variabila).**
  - indivizii sunt selectati in functie de cat de bine maximizeaza functia
- **Dezvoltarea unui sistem de control bazat pe reguli pentru simularea unui robot autonom**
  - fiecare populatie reprezinta un set de reguli pentru diferite comportamente
  - indivizii sunt selectati in functie de cat de bine completeaza seturile de reguli ale altor specii

# Acoperirea sirurilor



- Gasirea unui set potrivit de  $N$  vectori binari care se potrivesc (cat mai bine cu putinta) pe un set sursa de  $K$  vectori binari ( $K \gg N$ )
- In acest context, problema descompunerii consta in:
  - determinarea dimensiunii setului de potrivit
  - determinare acoperirii fiecarui element din set
- Calcularea fitness-ului unui set potrivit  $M$  se face folosind formula:

$$S(M) = \frac{1}{K} \sum_{i=1}^K \max(S(\vec{m}_1, \vec{t}_i), \dots, S(\vec{m}_N, \vec{t}_i))$$



# Algoritmul de baza

```
gen = 0
for each species s do begin
     $Pop_s(gen)$  = initializeaza populatia cu valori aleatorii
    evalueaza fitness-ul fiecarui individ  $Pop_s(gen)$ 
end
while conditia de terminare = false do begin
    gen = gen + 1
    for each species s do begin
        selecteaza  $Pop_s(gen)$  din  $Pop_s(gen-1)$  pe baza fitness-ului
        aplica operatorii genetici pe  $Pop_s(gen)$ 
        evalueaza fitness-ul fiecarui individ  $Pop_s(gen)$ 
    end
end
```



# Concluzii



- Arhitectura cooperativa co-evolutiva poate fi privita ca o extensie generala a oricarei paradigme evolutive si nu doar a algoritmilor genetici
  - de exemplu: software de control al robotilor in medii multi-agent
- Furnizeaza stimuli pentru evolutia spre un anumit numar interdependent de sub-componente care acopera nise multiple
- Evolueaza spre un nivel de generalitate potrivit
- Deoarece domeniile devin tot mai complexe, evolutia speciilor ar putea fi condusa de mai mult decat fitness-ul sistemului pentru a putea produce o descompunere optima

## C. Listingul codului sursa

```
1 ///////////////////////////////////////////////////////////////////
2 // (c) 2003 Sorin OSTAFIEV (sorin@ostafiev.com) //
3 ///////////////////////////////////////////////////////////////////
4
5 // GA.h: interface for the GA class.
6 //
7 ///////////////////////////////////////////////////////////////////
8
9 #if !defined(AFX_GA_H__7B677626_ODC7_4B47_9E97_734A880F694C__INCLUDED_)
10 #define AFX_GA_H__7B677626_ODC7_4B47_9E97_734A880F694C__INCLUDED_
11
12 #include "globals.h" // Added by ClassView
13 #if _MSC_VER > 1000
14 #pragma once
15 #endif // _MSC_VER > 1000
16
17
18 #include <vector>
19 using namespace std;
20
21 class GA
22 {
23
24
25 public:
26     double getAverageMatchedBits();
27     void step();
28     friend class Population;
29     friend class Genome;
30
31
32     GA(const int npops, const int pop_size, const double crossoverrate, const double mutationrate);
33     virtual ~GA();
34
35     vector <Population> _populations;
36
37 private:
38
39     GA(const GA& ga);
40     GA& operator=(const GA &ga);
41
42     double _cross_over_rate;
43     double _mutation_rate;
44     int _number_of_populations;
45 };
46
47
48 #endif // !defined(AFX_GA_H__7B677626_ODC7_4B47_9E97_734A880F694C__INCLUDED_)
49
```

```
1 ///////////////////////////////////////////////////////////////////
2 // (c) 2003 Sorin OSTAFIEV (sorin@ostafiev.com) //
3 ///////////////////////////////////////////////////////////////////
4
5 // GA.cpp: implementation of the GA class.
6 //
7 ///////////////////////////////////////////////////////////////////
8
9 #include "stdafx.h"
10 #include "StringCovering.h"
11 #include "GA.h"
12
13
14 #include "Population.h"
15
16
17 #include <cassert>
18 using namespace std;
19
20 #ifndef _DEBUG
21 #undef THIS_FILE
22 static char THIS_FILE[]=__FILE__;
23 #define new DEBUG_NEW
24 #endif
25
26 ///////////////////////////////////////////////////////////////////
27 // Construction/Destruction
28 ///////////////////////////////////////////////////////////////////
29
30 GA::GA(const int npops, const int pop_size, const double crossoverrate, const double mutationrate):
31     _cross_over_rate(crossoverrate),
32     _mutation_rate(mutationrate),
33     _number_of_populations(npops)
34 {
35     assert ((0.0f <= _mutation_rate) && (_mutation_rate <= 1.0f));
36     assert ((0.0f <= _cross_over_rate) && (_cross_over_rate <= 1.0f));
37
38     _populations.reserve(_number_of_populations);
39     for (int i = 0; i < _number_of_populations; i++)
40     {
41         _populations.push_back(Population(this, pop_size, Genome::GENOME_RANDOM));
42     };
43 }
44
45
46
47 GA::~GA()
48 {
49 }
50
51
52
53
54 void GA::step()
55 {
56     int i = 0;
57     for (i = 0; i < _populations.size(); i++)
58     {
59
60         _populations.at(i).pre_step();
61
62     };
63
64
65     for (i = 0; i < _populations.size(); i++)
66     {
67
68         _populations.at(i).step();
69
70     };
71
72
73     for (i = 0; i < _populations.size(); i++)
74     {
75
76         _populations.at(i).post_step();
77
78     };
79 }
80
81
82
83 double GA::getAverageMatchedBits()
```

```
84 {
85     const int target_set_size = targetSet->_genomes.size();
86
87     assert (_number_of_populations == _populations.size());
88
89     double overall_fitness = 0;
90
91     for (GenomesVector::const_iterator i = targetSet->_genomes.begin(); i != targetSet->_genomes.end(); i++)
92     {
93         double max = 0;
94
95         for (int k = 0; k < _number_of_populations; k++)
96         {
97             assert(0 < _populations.at(k)._genomes.size());
98             const Genome& best = _populations.at(k)._genomes.at(0);
99
100            const double current_fitness = best.getGenomeFitness(*i);
101
102            if (max < current_fitness)
103            {
104                max = current_fitness;
105            };
106
107        };
108
109        overall_fitness += max;
110
111    };
112
113    return (overall_fitness / target_set_size) * Genome::_genome_size;
114 }
115
```

```
1 ////////////////////////////////////////////////////////////////////
2 // (c) 2003 Sorin OSTAFIEV (sorin@ostafiev.com) //
3 ////////////////////////////////////////////////////////////////////
4
5 // Population.h: interface for the Population class.
6 //
7 ////////////////////////////////////////////////////////////////////
8
9 #if !defined(AFX_POPULATION_H_F09111B1_0480_45B1_A376_A230915BEF67__INCLUDED_)
10 #define AFX_POPULATION_H_F09111B1_0480_45B1_A376_A230915BEF67__INCLUDED_
11
12 #if _MSC_VER > 1000
13 #pragma once
14 #endif // _MSC_VER > 1000
15
16 #include "GA.h"
17 #include "Genome.h"
18 #include <vector>
19 using namespace std;
20
21 #include "globals.h"
22
23 class Population
24 {
25
26 public:
27     friend class Genome;
28     friend class GA;
29
30     const int getPopulationSize() const;
31     const GenomesVector::const_iterator getBestIndividual() const;
32
33     virtual ~Population();
34
35     Population(const GA* const ga, const int pop_size, Genome::genome_type type);
36     Population(const Population& population);
37     const Population& operator=(const Population &population);
38     GenomesVector _genomes;
39
40 private:
41     Population* _next_generation;
42     void pre_step();
43     void step();
44     void post_step();
45
46     int _population_size;
47
48     const GA* const _ga;
49     mutable GenomesVector::const_iterator _best_genome;
50
51 };
52
53
54 #endif // !defined(AFX_POPULATION_H_F09111B1_0480_45B1_A376_A230915BEF67__INCLUDED_)
55
```

```
1 ///////////////////////////////////////////////////////////////////
2 // (c) 2003 Sorin OSTAFIEV (sorin@ostafiev.com) //
3 ///////////////////////////////////////////////////////////////////
4
5 // Population.cpp: implementation of the Population class.
6 //
7 ///////////////////////////////////////////////////////////////////
8
9 #include "stdafx.h"
10 #include "StringCovering.h"
11 #include "Population.h"
12
13
14 #include <cassert>
15 #include <algorithm>
16 #include <functional>
17 using namespace std;
18
19
20 #ifdef _DEBUG
21 #undef THIS_FILE
22 static char THIS_FILE[] = __FILE__;
23 #define new DEBUG_NEW
24 #endif
25
26 ///////////////////////////////////////////////////////////////////
27 // Construction/Destruction
28 ///////////////////////////////////////////////////////////////////
29
30
31 Population::Population(const GA* const ga, const int pop_size, Genome::genome_type type):
32     _best_genome(NULL),
33     _ga(ga),
34     _population_size(pop_size),
35     _next_generation(NULL)
36 {
37
38
39     _genomes.reserve(_population_size);
40     for (int i = 0; i < _population_size; i++)
41     {
42         _genomes.push_back(Genome(this, ga, type));
43     };
44 }
45
46
47 Population::~Population()
48 {
49
50 }
51
52 const GenomesVector::const_iterator Population::getBestIndividual() const
53 {
54     if (NULL == _best_genome)
55     {
56         double max_fitness = -1;
57         for (GenomesVector::const_iterator i = _genomes.begin(); i != _genomes.end(); i++)
58         {
59
60             const double current_fitness = i->getPopulationFitness();
61             assert ((0.0f <= current_fitness) && (current_fitness <= 1.0f));
62
63             if (max_fitness < i->getPopulationFitness())
64             {
65                 max_fitness = i->getPopulationFitness();
66                 _best_genome = (const GenomesVector::const_iterator) i;
67             };
68
69         };
70
71         assert ((0.0f <= max_fitness) && (max_fitness <= 1.0f));
72
73     };
74     return _best_genome;
75 }
76
77 const int Population::getPopulationSize() const
78 {
79     assert (0 < _genomes.size());
80     return _genomes.size();
81 }
82
83
```

```
84 Population::Population(const Population& population):
85     _ga(population._ga)
86 {
87     operator=(population);
88 };
89
90
91 const Population& Population::operator=(const Population& population)
92 {
93     if (this != &population)
94     {
95         _best_genome = population._best_genome;
96         const_cast <const GA*> (_ga) = population._ga;
97         _genomes = population._genomes;
98         _population_size = population._population_size;
99         _next_generation = population._next_generation;
100        assert (_ga == population._ga);
101
102        for (int i = 0; i < population._genomes.size(); i++)
103        {
104            const_cast <const Population*> (_genomes.at(i)._population) = this;
105        }
106    };
107 };
108
109
110 return (*this);
111 };
112 };
113
114
115 void Population::pre_step()
116 {
117     assert (NULL == _next_generation);
118     _next_generation = new Population(NULL, _population_size, Genome::GENOME_ZERO);
119     assert (NULL != _next_generation);
120
121     assert(_genomes.size() == _population_size);
122
123     typedef pair <double, int> DOUBLE2INT;
124
125     vector <DOUBLE2INT> d2i;
126
127     d2i.reserve(_genomes.size());
128     for (int i = 0; i < _genomes.size(); i++)
129     {
130         d2i.push_back(make_pair(_genomes.at(i).getGAFitness(), i));
131     };
132
133     vector<DOUBLE2INT>::iterator i1 = d2i.begin();
134     vector<DOUBLE2INT>::iterator i2 = d2i.end();
135
136     sort (i1, i2, greater<DOUBLE2INT>());
137
138     double max_sum = 0;
139     for (i = 0; i < _genomes.size(); i++)
140     {
141         max_sum += d2i.at(i).first;
142     };
143
144     int k = 0;
145     double sum = 0;
146     for (i = 0; (i < _population_size) && (k < _population_size); i++)
147     {
148         sum += d2i.at(i).first;
149
150         const int delta = int(1 + double(sum * _population_size) / max_sum) - k;
151         assert (0 <= delta);
152
153         for (int j = k; j < k + delta; j++)
154         {
155             if (j < _population_size)
156             {
157                 _next_generation->_genomes.at(j) = _genomes.at(d2i.at(i).second);
158             };
159         };
160     };
161 };
162
163
164
165
166
```

```
167         k += delta;
168     };
169 };
170     assert (_population_size <= k);
171 }
172 }
173 }
174
175 void Population::step()
176 {
177     vector <bool> b(_population_size, false);
178
179     const double cross_over_rate = _ga->_cross_over_rate;
180     const int mutation_factor = int(100 * _ga->_mutation_rate);
181     assert (0 < mutation_factor);
182
183     const int cross_overs = int((_population_size * cross_over_rate) / 2);
184
185     for(int i = 0; i < cross_overs; i++)
186     {
187         const int i1 = rand() % _population_size;
188         const int i2 = rand() % _population_size;
189
190         _next_generation->_genomes.at(i1).Crossover(_next_generation->_genomes.at(i2));
191     };
192
193
194     for (i = 0; i < _population_size; i++)
195     {
196         if ((rand() % 100) < mutation_factor)
197         {
198             _next_generation->_genomes.at(i).Mutate();
199         };
200     };
201 };
202
203 };
204 };
205
206
207 void Population::post_step()
208 {
209
210     for (int i = 0; i < _genomes.size(); i++)
211     {
212         _genomes.at(i) = _next_generation->_genomes.at(i);
213         _genomes.at(i)._gaFitness = -1;
214         _genomes.at(i)._populationFitness = -1;
215     };
216 };
217
218 _best_genome = NULL;
219
220 // clean up
221 assert (NULL != _next_generation);
222 delete _next_generation;
223 _next_generation = NULL;
224 }
225
226
```

```
1 ////////////////////////////////////////////////////////////////////
2 // (c) 2003 Sorin OSTAFIEV (sorin@ostafiev.com) //
3 ////////////////////////////////////////////////////////////////////
4
5 // Genome.h: interface for the Genome class.
6 //
7 ////////////////////////////////////////////////////////////////////
8
9 #if !defined(AFX_GENOME_H_1B30186E_485F_434A_B615_BE44BC598C7A__INCLUDED_)
10 #define AFX_GENOME_H_1B30186E_485F_434A_B615_BE44BC598C7A__INCLUDED_
11
12 #if _MSC_VER > 1000
13 #pragma once
14 #endif // _MSC_VER > 1000
15
16 class Population;
17 class GA;
18 #include "globals.h"
19 #include <bitset>
20
21 using namespace std;
22
23 class Genome
24 {
25     static int _genome_size;
26
27 public:
28     void setFromPattern(const string& pattern);
29
30     friend class Population;
31     friend class GA;
32
33
34     enum genome_type
35     {
36         GENOME_RANDOM,
37         GENOME_ZERO
38     };
39
40     const double getGAFitness() const;
41     const double getPopulationFitness() const;
42     void Crossover(Genome& genome);
43     void Mutate();
44     Genome(const Population* const population, const GA* const ga, const genome_type type = GENOME_ZERO);
45     virtual ~Genome();
46
47
48     Genome(const Genome& genome);
49     Genome& operator=(const Genome &genome);
50
51     bitset <MAX_GENOME_LENIGHT> _body;
52
53 private:
54
55     const double getGenomeFitness(const Genome& genome) const; // genome over genome
56     const Population* const _population; // which population we belong
57     const GA* const _ga; // ...and which ga
58
59     mutable double _populationFitness; // fitness within population
60     mutable double _gaFitness; // fitness within ga
61 };
62
63
64
65 #endif // !defined(AFX_GENOME_H_1B30186E_485F_434A_B615_BE44BC598C7A__INCLUDED_)
66
```

```
1 ///////////////////////////////////////////////////////////////////
2 // (c) 2003 Sorin OSTAFIEV (sorin@ostafiev.com) //
3 ///////////////////////////////////////////////////////////////////
4
5 // Genome.cpp: implementation of the Genome class.
6 //
7 ///////////////////////////////////////////////////////////////////
8
9
10 #include "stdafx.h"
11 #include "StringCovering.h"
12 #include "Genome.h"
13
14 #include "globals.h"
15 #include <cassert>
16 #include "Population.h"
17
18 #ifndef _DEBUG
19 #undef THIS_FILE
20 static char THIS_FILE[] = __FILE__;
21 #define new DEBUG_NEW
22 #endif
23
24
25
26 ///////////////////////////////////////////////////////////////////
27 // Construction/Destruction
28 ///////////////////////////////////////////////////////////////////
29
30 Genome::Genome(const Population* const population, const GA* const ga, const genome_type type /* = GENOME_ZERO */)
31     _population(population),
32     _ga(ga),
33     _populationFitness(-1),
34     _gaFitness(-1),
35     _body(_genome_size)
36 {
37     switch (type)
38     {
39
40         case GENOME_RANDOM:
41             {
42
43                 for (int i = 0; i < _genome_size; i++)
44                 {
45                     if (0 == (rand() & 1))
46                     {
47                         _body.reset(i);
48                     }
49                     else
50                     {
51                         _body.set(i);
52                     }
53                 };
54             };
55             break;
56
57         case GENOME_ZERO:
58             {
59                 _body.reset();
60             };
61             break;
62
63         default:
64             assert(0);
65             break;
66     };
67 }
68
69 Genome::~Genome()
70 {
71 }
72
73
74 void Genome::Mutate()
75 {
76     assert (_genome_size == _body.size());
77
78     const int mutation_bit = rand() % _genome_size;
79     _body.flip(mutation_bit);
80
81 }
82
83 void Genome::Crossover(Genome &genome)
```

```
84 {
85     if (this != &genome) //cross with someone, but not with me...
86     {
87         assert (_genome_size == _body.size());
88         const int half_genome_size = _genome_size / 2;
89         for (int i = 0; i < half_genome_size; i++)
90         {
91             const bool b = _body.at(i);
92             _body.at(i) = genome._body.at(i);
93             genome._body.at(i) = b;
94         };
95     };
96 };
97 };
98 };
99 }
100
101
102 // genome over population
103 const double Genome::getPopulationFitness() const
104 {
105     if (_populationFitness < 0)
106     {
107         double population_fitness = 0;
108
109         for (GenomesVector::const_iterator i = targetSet->_genomes.begin(); i != targetSet->_genomes.end(); i++)
110         {
111             population_fitness += getGenomeFitness(*i);
112         };
113         population_fitness /= targetSet->getPopulationSize();
114
115         _populationFitness = population_fitness;
116         assert ((0.0f <= _populationFitness) && (_populationFitness <= 1.0f));
117     };
118     return _populationFitness;
119 }
120
121 // genome over genome;
122 const double Genome::getGenomeFitness(const Genome &genome) const
123 {
124     double genome_fitness = 0;
125     for (int i = 0; i < _genome_size; i++)
126     {
127         if (genome._body.at(i) == _body.at(i))
128         {
129             genome_fitness++;
130         };
131     };
132     return genome_fitness / _genome_size;
133 }
134
135 const double Genome::getGAFitness() const
136 {
137     if (NULL == _ga)
138     {
139         return -1;
140     };
141     if (_gaFitness < 0)
142     {
143         double ga_fitness = 0;
144
145         for (GenomesVector::const_iterator i = targetSet->_genomes.begin(); i != targetSet->_genomes.end(); i++)
146         {
147             double max_fitness = -1;
148             for (PopulationsVector::const_iterator k = _ga->_populations.begin(); k != _ga->_populations.end(); k++)
149             {
150                 double current_fitness = -1;
151
152                 if (_population != k)
153                 {
154                     current_fitness = i->getGenomeFitness(*k->getBestIndividual());
155                 }
156                 else
157                 {

```

```
167         current_fitness = i->getGenomeFitness(*this);
168     };
169     assert ((0.0f <= current_fitness) && (current_fitness <= 1.0f));
170
171
172
173     if (max_fitness < current_fitness)
174     {
175         max_fitness = current_fitness;
176     };
177
178
179     };
180
181     assert ((0.0f <= max_fitness) && (max_fitness <= 1.0f));
182
183     ga_fitness += max_fitness;
184
185 };
186
187 ga_fitness /= targetSet->_genomes.size();
188
189 _gaFitness = ga_fitness;
190 assert ((0.0f <= _gaFitness) && (_gaFitness <= 1.0f));
191 };
192
193 return _gaFitness;
194 }
195
196
197 Genome::Genome(const Genome& genome):
198     _ga(genome._ga),
199     _population(genome._population)
200 {
201     operator=(genome);
202 };
203
204
205 Genome& Genome::operator=(const Genome &genome)
206 {
207     if (this != &genome)
208     {
209         _body = genome._body;
210         const_cast <const GA*> (_ga) = genome._ga;
211         _gaFitness = genome._gaFitness;
212         const_cast <const Population*> (_population) = genome._population;
213         _populationFitness = genome._populationFitness;
214
215
216
217         assert (_ga == genome._ga);
218         assert (_populationFitness == genome._populationFitness);
219     };
220
221     return (*this);
222 };
223
224
225 void Genome::setFromPattern(const string &pattern)
226 {
227     assert (_genome_size == _body.size());
228     assert (_genome_size == pattern.size());
229
230     for (int i = 0; i < _genome_size; i++)
231     {
232         switch (pattern.at(_genome_size - 1 - i))
233         {
234             case '0':
235                 _body.at(i) = false;
236                 break;
237
238             case '1':
239                 _body.at(i) = true;
240                 break;
241
242             default:
243                 if (0 == rand() % 2)
244                 {
245                     _body.at(i) = false;
246                 }
247                 else
248                 {
249                     _body.at(i) = true;
```

```
250         };  
251         break;  
252  
253     };  
254  
255 };  
256  
257 }  
258
```

```
1 ///////////////////////////////////////////////////////////////////
2 // (c) 2003 Sorin OSTAFIEV (sorin@ostafiev.com) //
3 ///////////////////////////////////////////////////////////////////
4
5 #ifndef __global_h__
6 #define __global_h__
7
8 #pragma warning(disable:4786) //stl issue - decoration too long
9
10 #include <vector>
11
12 using namespace std;
13
14 class Genome;
15 class Population;
16
17 typedef vector<Genome> GenomesVector;
18 typedef vector<Population> PopulationsVector;
19
20 const MAX_GENOME_LENGTH = 32;
21
22
23 extern Population* targetSet;
24
25 #endif
```

```
1 ////////////////////////////////////////////////////////////////////
2 // (c) 2003 Sorin OSTAFIEV (sorin@ostafiev.com) //
3 ////////////////////////////////////////////////////////////////////
4
5 // StringCovering.h : main header file for the STRINGCOVERING application
6 //
7
8 #if !defined(AFX_STRINGCOVERING_H_F93D04E3_5CC0_408C_A598_27A29EF3CA33__INCLUDED_)
9 #define AFX_STRINGCOVERING_H_F93D04E3_5CC0_408C_A598_27A29EF3CA33__INCLUDED_
10
11 #if _MSC_VER > 1000
12 #pragma once
13 #endif // _MSC_VER > 1000
14
15 #ifndef __AFXWIN_H__
16 #error include 'stdafx.h' before including this file for PCH
17 #endif
18
19 #include "resource.h" // main symbols
20
21 ////////////////////////////////////////////////////////////////////
22 // CStringCoveringApp:
23 // See StringCovering.cpp for the implementation of this class
24 //
25
26 class CStringCoveringApp : public CWinApp
27 {
28 public:
29     CStringCoveringApp();
30
31 // Overrides
32 // ClassWizard generated virtual function overrides
33 //{{AFX_VIRTUAL(CStringCoveringApp)
34 public:
35     virtual BOOL InitInstance();
36 //}}AFX_VIRTUAL
37
38 // Implementation
39
40 //{{AFX_MSG(CStringCoveringApp)
41 // NOTE - the ClassWizard will add and remove member functions here.
42 // DO NOT EDIT what you see in these blocks of generated code !
43 //}}AFX_MSG
44 DECLARE_MESSAGE_MAP()
45 };
46
47
48 ////////////////////////////////////////////////////////////////////
49
50 //{{AFX_INSERT_LOCATION}}
51 // Microsoft Visual C++ will insert additional declarations immediately before the previous line.
52
53 #endif // !defined(AFX_STRINGCOVERING_H_F93D04E3_5CC0_408C_A598_27A29EF3CA33__INCLUDED_)
54
```

```
1 ////////////////////////////////////////////////////////////////////
2 // (c) 2003 Sorin OSTAFIEV (sorin@ostafiev.com) //
3 ////////////////////////////////////////////////////////////////////
4
5 // StringCovering.cpp : Defines the class behaviors for the application.
6 //
7
8 #include "stdafx.h"
9 #include "StringCovering.h"
10 #include "StringCoveringDlg.h"
11
12 #ifdef _DEBUG
13 #define new DEBUG_NEW
14 #undef THIS_FILE
15 static char THIS_FILE[] = __FILE__;
16 #endif
17
18 ////////////////////////////////////////////////////////////////////
19 // CStringCoveringApp
20
21 BEGIN_MESSAGE_MAP(CStringCoveringApp, CWinApp)
22     //{AFX_MSG_MAP(CStringCoveringApp)
23     // NOTE - the ClassWizard will add and remove mapping macros here.
24     // DO NOT EDIT what you see in these blocks of generated code!
25     //}AFX_MSG
26     ON_COMMAND(ID_HELP, CWinApp::OnHelp)
27 END_MESSAGE_MAP()
28
29 ////////////////////////////////////////////////////////////////////
30 // CStringCoveringApp construction
31
32 CStringCoveringApp::CStringCoveringApp()
33 {
34     // TODO: add construction code here,
35     // Place all significant initialization in InitInstance
36 }
37
38 ////////////////////////////////////////////////////////////////////
39 // The one and only CStringCoveringApp object
40
41 CStringCoveringApp theApp;
42
43 ////////////////////////////////////////////////////////////////////
44 // CStringCoveringApp initialization
45
46 BOOL CStringCoveringApp::InitInstance()
47 {
48     AfxEnableControlContainer();
49
50     // Standard initialization
51     // If you are not using these features and wish to reduce the size
52     // of your final executable, you should remove from the following
53     // the specific initialization routines you do not need.
54
55 #ifdef _AFXDLL
56     Enable3dControls();           // Call this when using MFC in a shared DLL
57 #else
58     Enable3dControlsStatic();     // Call this when linking to MFC statically
59 #endif
60
61     CStringCoveringDlg dlg;
62     m_pMainWnd = &dlg;
63     int nResponse = dlg.DoModal();
64     if (nResponse == IDOK)
65     {
66         // TODO: Place code here to handle when the dialog is
67         // dismissed with OK
68     }
69     else if (nResponse == IDCANCEL)
70     {
71         // TODO: Place code here to handle when the dialog is
72         // dismissed with Cancel
73     }
74
75     // Since the dialog has been closed, return FALSE so that we exit the
76     // application, rather than start the application's message pump.
77     return FALSE;
78 }
79
```

```
1 ///////////////////////////////////////////////////////////////////
2 // (c) 2003 Sorin OSTAFIEV (sorin@ostafiev.com) //
3 ///////////////////////////////////////////////////////////////////
4
5 // StringCoveringDlg.h : header file
6 //
7
8 #if !defined(AFX_STRINGCOVERINGDLG_H_C7A2133D_47D6_42CB_B33B_711F041DF8DF__INCLUDED_)
9 #define AFX_STRINGCOVERINGDLG_H_C7A2133D_47D6_42CB_B33B_711F041DF8DF__INCLUDED_
10
11 #if _MSC_VER > 1000
12 #pragma once
13 #endif // _MSC_VER > 1000
14
15 ///////////////////////////////////////////////////////////////////
16 // CStringCoveringDlg dialog
17
18 #include "MyStatic.h"
19
20 class CStringCoveringDlg : public CDialog
21 {
22 // Construction
23     friend unsigned int DoEvolution(void *data);
24
25 public:
26     void EnableInterface();
27     void DisableInterface();
28     void UpdateEvolutionPanel();
29     void UpdateDialogFromValues();
30     bool UpdateValuesFromDialog();
31     bool UpdateValuesFromDialog1();
32     CStringCoveringDlg(CWnd* pParent = NULL); // standard constructor
33
34 // Dialog Data
35     //{AFX_DATA(CStringCoveringDlg)
36     enum { IDD = IDD_STRINGCOVERING_DIALOG };
37     CButton m_ok;
38     CButton m_load_target;
39     CButton m_load_patterns;
40     CButton m_select_pattern;
41     CButton m_generate;
42     CButton m_remove_selected;
43     CButton m_clear_patterns;
44     CButton m_go;
45     CMyStatic m_evolution;
46     CListBox m_best;
47     CProgressCtrl m_progress;
48     CStatic m_gen_no;
49     CStatic m_matched_bits;
50     CEdit m_target_set_size;
51     CListBox m_selected_patterns;
52     CListBox m_patterns;
53     CEdit m_target_set_file;
54     CEdit m_target_size;
55     CEdit m_genome_size;
56     CEdit m_match_size;
57     CEdit m_population_size;
58     CEdit m_number_of_generations;
59     CEdit m_mutation_rate;
60     CEdit m_crossover_rate;
61     //}AFX_DATA
62
63     // ClassWizard generated virtual function overrides
64     //{AFX_VIRTUAL(CStringCoveringDlg)
65     protected:
66     virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
67     //}AFX_VIRTUAL
68
69 // Implementation
70
71 private:
72     void InitializeValues();
73     int target_size;
74     int population_size;
75     int match_size;
76     int number_of_generations;
77     double crossover_rate;
78     double mutation_rate;
79     int genome_size;
80
81     CSpinButtonCtrl* spin;
82
83     CDC* pDC;
```

```
84     int target_set_size;
85
86     int gen_no;
87     double matched_bits;
88
89     int w, h;
90     int last_x, last_y;
91
92     bool running;
93 protected:
94     HICON m_hIcon;
95
96     // Generated message map functions
97     //{AFX_MSG(CStringCoveringDlg)
98     virtual BOOL OnInitDialog();
99     afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
100    afx_msg void OnPaint();
101    afx_msg HCURSOR OnQueryDragIcon();
102    afx_msg void OnGo();
103    afx_msg void OnLoadTargetFile();
104    afx_msg void OnLoadPatterns();
105    afx_msg void OnSelectPatterns();
106    afx_msg void OnRemoveSelected();
107    afx_msg void OnClearPatterns();
108    afx_msg void OnGenerateTargetSet();
109    afx_msg void OnDblclkPatterns();
110    afx_msg void OnDblclkSelectedPatterns();
111    afx_msg void OnKillfocusTsize();
112    afx_msg void OnKillfocusPopulationSize();
113    afx_msg void OnKillfocusGenomesNo();
114    afx_msg void OnKillfocusMutationRate();
115    afx_msg void OnKillfocusCrossoverRate();
116    afx_msg void OnKillfocusNoOfGenerations();
117    virtual void OnOK();
118    afx_msg void OnAbout();
119    afx_msg void OnClose();
120    //}AFX_MSG
121    DECLARE_MESSAGE_MAP()
122 };
123
124 //){AFX_INSERT_LOCATION}}
125 // Microsoft Visual C++ will insert additional declarations immediately before the previous line.
126
127 #endif // !defined(AFX_STRINGCOVERINGDLG_H__C7A2133D_47D6_42CB_B33E_711F041DF8DF__INCLUDED_)
128
```

```
1 ///////////////////////////////////////////////////////////////////
2 // (c) 2003 Sorin OSTAFIEV (sorin@ostafiev.com) //
3 ///////////////////////////////////////////////////////////////////
4
5 // StringCoveringDlg.cpp : implementation file
6 //
7
8 #include "stdafx.h"
9 #include "StringCovering.h"
10 #include "StringCoveringDlg.h"
11
12
13 #include "GA.h"
14 #include "Population.h"
15 #include "Genome.h"
16 #include "MyStatic.h"
17
18
19 #include <string>
20 #include <fstream>
21 #include <cassert>
22 using namespace std;
23
24
25
26 #ifdef _DEBUG
27 #define new DEBUG_NEW
28 #undef THIS_FILE
29 static char THIS_FILE[] = __FILE__;
30 #endif
31
32
33
34 ///////////////////////////////////////////////////////////////////
35 // CAboutDlg dialog used for App About
36
37 class CAboutDlg : public CDialog
38 {
39 public:
40     CAboutDlg();
41
42     // Dialog Data
43     //{AFX_DATA(CAboutDlg)
44     enum { IDD = IDD_ABOUTBOX };
45     //{AFX_DATA
46
47     // ClassWizard generated virtual function overrides
48     //{AFX_VIRTUAL(CAboutDlg)
49     protected:
50     virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
51     //{AFX_VIRTUAL
52
53     // Implementation
54     protected:
55     //{AFX_MSG(CAboutDlg)
56     //{AFX_MSG
57     DECLARE_MESSAGE_MAP()
58 };
59
60 CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
61 {
62     //{AFX_DATA_INIT(CAboutDlg)
63     //{AFX_DATA_INIT
64 }
65
66 void CAboutDlg::DoDataExchange(CDataExchange* pDX)
67 {
68     CDialog::DoDataExchange(pDX);
69     //{AFX_DATA_MAP(CAboutDlg)
70     //{AFX_DATA_MAP
71 }
72
73 BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
74     //{AFX_MSG_MAP(CAboutDlg)
75     // No message handlers
76     //{AFX_MSG_MAP
77 END_MESSAGE_MAP()
78
79 ///////////////////////////////////////////////////////////////////
80 // CStringCoveringDlg dialog
81 const COLORREF background_color = RGB(192, 192, 200);
82 const COLORREF line_color = RGB(150, 100, 50);
83
```

```
84
85
86 CStringCoveringDlg::CStringCoveringDlg(CWnd* pParent /*=NULL*/)
87     : CDialog(CStringCoveringDlg::IDD, pParent)
88 {
89     //{AFX_DATA_INIT(CStringCoveringDlg)
90     // NOTE: the ClassWizard will add member initialization here
91     //}AFX_DATA_INIT
92     // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
93     InitializeValues();
94     pDC = NULL;
95     spin = NULL;
96     m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
97
98     m_evolution.background_color = background_color;
99     m_evolution.line_color = line_color;
100 }
101 }
102
103 void CStringCoveringDlg::DoDataExchange(CDataExchange* pDX)
104 {
105     CDialog::DoDataExchange(pDX);
106     //{AFX_DATA_MAP(CStringCoveringDlg)
107     DDX_Control(pDX, IDOK, m_ok);
108     DDX_Control(pDX, IDC_LOAD_TARGET_FILE, m_load_target);
109     DDX_Control(pDX, IDC_LOAD_PATTERNS, m_load_patterns);
110     DDX_Control(pDX, IDC_SELECT_PATTERNS, m_select_pattern);
111     DDX_Control(pDX, IDC_GENERATE_TARGET_SET, m_generate);
112     DDX_Control(pDX, IDC_REMOVE_SELECTED, m_remove_selected);
113     DDX_Control(pDX, IDC_CLEAR_PATTERNS, m_clear_patterns);
114     DDX_Control(pDX, IDC_GO, m_go);
115     DDX_Control(pDX, IDC_EVOLUTION, m_evolution);
116     DDX_Control(pDX, IDC_BESTS, m_bests);
117     DDX_Control(pDX, IDC_PROGRESS, m_progress);
118     DDX_Control(pDX, IDC_GENNO, m_gen_no);
119     DDX_Control(pDX, IDC_MATCHED_BITS, m_matched_bits);
120     DDX_Control(pDX, IDC_TSIZE, m_target_set_size);
121     DDX_Control(pDX, IDC_SELECTED_PATTERNS, m_selected_patterns);
122     DDX_Control(pDX, IDC_PATTERNS, m_patterns);
123     DDX_Control(pDX, IDC_TARGET_SET_FILE, m_target_set_file);
124     DDX_Control(pDX, IDC_TARGET_SIZE, m_target_size);
125     DDX_Control(pDX, IDC_GENOME_SIZE, m_genome_size);
126     DDX_Control(pDX, IDC_GENOMES_NO, m_match_size);
127     DDX_Control(pDX, IDC_POPULATION_SIZE, m_population_size);
128     DDX_Control(pDX, IDC_NO_OF_GENERATIONS, m_number_of_generations);
129     DDX_Control(pDX, IDC_MUTATION_RATE, m_mutation_rate);
130     DDX_Control(pDX, IDC_CROSSOVER_RATE, m_crossover_rate);
131     //}AFX_DATA_MAP
132 }
133
134 BEGIN_MESSAGE_MAP(CStringCoveringDlg, CDialog)
135     //{AFX_MSG_MAP(CStringCoveringDlg)
136     ON_WM_SYSCOMMAND()
137     ON_WM_PAINT()
138     ON_WM_QUERYDRAGICON()
139     ON_BN_CLICKED(IDC_GO, OnGo)
140     ON_BN_CLICKED(IDC_LOAD_TARGET_FILE, OnLoadTargetFile)
141     ON_BN_CLICKED(IDC_LOAD_PATTERNS, OnLoadPatterns)
142     ON_BN_CLICKED(IDC_SELECT_PATTERNS, OnSelectPatterns)
143     ON_BN_CLICKED(IDC_REMOVE_SELECTED, OnRemoveSelected)
144     ON_BN_CLICKED(IDC_CLEAR_PATTERNS, OnClearPatterns)
145     ON_BN_CLICKED(IDC_GENERATE_TARGET_SET, OnGenerateTargetSet)
146     ON_LBN_DBLCLK(IDC_PATTERNS, OnDbldclkPatterns)
147     ON_LBN_DBLCLK(IDC_SELECTED_PATTERNS, OnDbldclkSelectedPatterns)
148     ON_EN_KILLFOCUS(IDC_TSIZE, OnKillfocusTsize)
149     ON_EN_KILLFOCUS(IDC_POPULATION_SIZE, OnKillfocusPopulationSize)
150     ON_EN_KILLFOCUS(IDC_GENOMES_NO, OnKillfocusGenomesNo)
151     ON_EN_KILLFOCUS(IDC_MUTATION_RATE, OnKillfocusMutationRate)
152     ON_EN_KILLFOCUS(IDC_CROSSOVER_RATE, OnKillfocusCrossoverRate)
153     ON_EN_KILLFOCUS(IDC_NO_OF_GENERATIONS, OnKillfocusNoOfGenerations)
154     ON_BN_CLICKED(IDC_ABOUT, OnAbout)
155     ON_WM_CLOSE()
156     //}AFX_MSG_MAP
157 END_MESSAGE_MAP()
158
159 ////////////////////////////////////////////////////////////////////////////////
160 // CStringCoveringDlg message handlers
161
162
163 int Genome::_genome_size = MAX_GENOME_LENHT;
164
165 Population* targetSet = NULL;
166
```

```
167 const string target_set_file_name = "targetSet.txt";
168 const string patterns_file = "patterns.txt";
169
170 vector<string> patterns;
171 vector<string> selected_patterns;
172 vector<string> bests;
173
174 CStringCoveringDlg* mainDlg = NULL;
175
176
177 string stringFromDouble(const double d)
178 {
179     char ss[100];
180     sprintf(ss, "%f", d);
181
182     string s(ss);
183
184     string::size_type p = s.find_first_of(".");
185     if (p != string::npos)
186     {
187         if (p + 4 < s.size())
188         {
189             s = string(s, 0, p + 4);
190         };
191     };
192 };
193
194
195 return s;
196 };
197
198 string stringFromInt(const int i)
199 {
200     char s[100];
201     itoa(i, s, 10);
202     return string(s);
203 };
204
205 int intFromString(const string& s)
206 {
207     int i = 0;
208     i = atoi(s.c_str());
209     return i;
210 };
211
212 double doubleFromString(const string& s)
213 {
214     double d = 0.0f;
215     d = atof(s.c_str());
216     return d;
217 };
218
219 string stringFromCWnd(const CWnd* const p)
220 {
221     char s[2001];
222     p->GetWindowText(s, 2000);
223     return string(s);
224 };
225
226 BOOL CStringCoveringDlg::OnInitDialog()
227 {
228     CDialog::OnInitDialog();
229
230     // Add "About..." menu item to system menu.
231
232     // IDM_ABOUTBOX must be in the system command range.
233     ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
234     ASSERT(IDM_ABOUTBOX < 0xF000);
235
236     CMenu* pSysMenu = GetSystemMenu(FALSE);
237     if (pSysMenu != NULL)
238     {
239         CString strAboutMenu;
240         strAboutMenu.LoadString(IDS_ABOUTBOX);
241         if (!strAboutMenu.IsEmpty())
242         {
243             pSysMenu->AppendMenu(MF_SEPARATOR);
244             pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
245         }
246     }
247
248     // Set the icon for this dialog. The framework does this automatically
249     // when the application's main window is not a dialog
```

```
250     SetIcon(m_hIcon, TRUE);           // Set big icon
251     SetIcon(m_hIcon, FALSE);        // Set small icon
252
253     // TODO: Add extra initialization here
254
255
256     running = false;
257
258     spin = (CSpinButtonCtrl*)GetDlgItem(IDC_SELECT_NO_GENOMES);
259     assert (NULL != spin);
260     spin->SetBuddy(&m_match_size);
261     spin->SetRange(1, 100);
262
263
264     m_progress.SetRange(0, 100);
265
266     OnLoadTargetFile();
267     OnLoadPatterns();
268     UpdateDialogFromValues();
269
270
271     pDC = m_evolution.GetDC();
272
273     RECT rect;
274     m_evolution.GetClientRect(&rect);
275     w = rect.right;
276     h = rect.bottom;
277
278     m_evolution.w = w;
279     m_evolution.h = h;
280
281     //CBitmap* bmp = new CBitmap();
282     //BOOL boo = bmp->CreateCompatibleBitmap(pDC, 1000, 1000);
283     //pDC->SelectObject(bmp);
284
285
286     return TRUE; // return TRUE unless you set the focus to a control
287 }
288
289 void CStringCoveringDlg::OnSysCommand(UINT nID, LPARAM lParam)
290 {
291     if ((nID & 0xFFFF0) == IDM_ABOUTBOX)
292     {
293         CAboutDlg dlgAbout;
294         dlgAbout.DoModal();
295     }
296     else
297     {
298         CDialog::OnSysCommand(nID, lParam);
299     }
300 }
301
302 // If you add a minimize button to your dialog, you will need the code below
303 // to draw the icon. For MFC applications using the document/view model,
304 // this is automatically done for you by the framework.
305
306 void CStringCoveringDlg::OnPaint()
307 {
308     if (IsIconic())
309     {
310         CPaintDC dc(this); // device context for painting
311
312         SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);
313
314         // Center icon in client rectangle
315         int cxIcon = GetSystemMetrics(SM_CXICON);
316         int cyIcon = GetSystemMetrics(SM_CYICON);
317         CRect rect;
318         GetClientRect(&rect);
319         int x = (rect.Width() - cxIcon + 1) / 2;
320         int y = (rect.Height() - cyIcon + 1) / 2;
321
322         // Draw the icon
323         dc.DrawIcon(x, y, m_hIcon);
324     }
325     else
326     {
327         CDialog::OnPaint();
328     }
329 }
330
331 // The system calls this to obtain the cursor to display while the user drags
332 // the minimized window.
```

```
333 HCURSOR CStringCoveringDlg::OnQueryDragIcon()
334 {
335     return (HCURSOR) m_hIcon;
336 }
337
338
339 unsigned int DoEvolution(void *data);
340
341 void CStringCoveringDlg::OnGo()
342 {
343     // TODO: Add your control notification handler code here
344
345     if (false == running)
346     {
347
348         if (false == UpdateValuesFromDialog())
349         {
350             AfxMessageBox("Invalid parameters!", MB_OK | MB_ICONSTOP);
351
352         }
353         else
354         {
355
356             srand(time(NULL));
357
358             m_evolution.v.clear();
359             assert (NULL != targetSet);
360
361             last_x = -1;
362             last_y = -1;
363
364             mainDlg = this;
365
366             running = true;
367             DisableInterface();
368             m_go.SetWindowText("Stop...");
369             CWinThread* interfaceThread = AfxBeginThread(DoEvolution, NULL);
370
371         };
372     }
373     else
374     { // oops! we are in the middle of something...
375
376         running = false;
377     };
378 }
379
380
381 bool CStringCoveringDlg::UpdateValuesFromDialog()
382 {
383     target_set_size = intFromString(stringFromCWnd(&m_target_set_size));
384     population_size = intFromString(stringFromCWnd(&m_population_size));
385     match_size = intFromString(stringFromCWnd(&m_match_size));
386
387     mutation_rate = doubleFromString(stringFromCWnd(&m_mutation_rate));
388     crossover_rate = doubleFromString(stringFromCWnd(&m_crossover_rate));
389
390     number_of_generations = intFromString(stringFromCWnd(&m_number_of_generations));
391
392     if (target_set_size <= 0)
393     {
394         return false;
395     };
396
397     if ((population_size <= 0) || (1000 <= population_size))
398     {
399         return false;
400     };
401
402     if (match_size <= 0)
403     {
404         return false;
405     };
406
407     if (number_of_generations <= 0)
408     {
409         return false;
410     };
411
412     if ((mutation_rate < 0) || (1 < mutation_rate))
413     {
414         return false;
415     };
416 }
```

```
416
417     if ((crossover_rate < 0) || (1 < crossover_rate))
418     {
419         return false;
420     };
421
422     return true;
423 }
424 }
425
426
427 bool CStringCoveringDlg::UpdateValuesFromDialog1()
428 {
429     int tsize = 0;
430
431     tsize = intFromString(stringFromCWnd(&m_target_set_size));
432
433     const int count_selected = selected_patterns.size();
434
435     if ((0 < count_selected) && (0 < tsize))
436     {
437         target_set_size = tsize;
438         return true;
439     };
440
441     return false;
442 }
443 }
444 }
445
446 void CStringCoveringDlg::UpdateDialogFromValues()
447 {
448     m_match_size.SetWindowText(stringFromInt(match_size).c_str());
449     m_population_size.SetWindowText(stringFromInt(population_size).c_str());
450     m_number_of_generations.SetWindowText(stringFromInt(number_of_generations).c_str());
451     m_mutation_rate.SetWindowText(stringFromDouble(mutation_rate).c_str());
452     m_crossover_rate.SetWindowText(stringFromDouble(crossover_rate).c_str());
453     m_genome_size.SetWindowText(stringFromInt(genome_size).c_str());
454     m_target_size.SetWindowText(stringFromInt(target_size).c_str());
455     m_target_set_file.SetWindowText(target_set_file_name.c_str());
456
457
458     m_target_set_size.SetWindowText(stringFromInt(target_set_size).c_str());
459
460
461     m_patterns.ResetContent();
462     for (int i = 0; i < patterns.size(); i++)
463     {
464         m_patterns.AddString(patterns.at(i).c_str());
465     };
466
467
468     m_selected_patterns.ResetContent();
469     for (i = 0; i < selected_patterns.size(); i++)
470     {
471         m_selected_patterns.AddString(selected_patterns.at(i).c_str());
472     };
473 }
474 }
475 }
476
477 void CStringCoveringDlg::InitializeValues()
478 {
479     population_size = 50;
480     match_size = 3;
481     number_of_generations = 200;
482     crossover_rate = 0.6;
483     mutation_rate = 0.15;
484     genome_size = 32;
485     target_size = 0;
486
487     target_set_size = 100;
488 }
489 }
490
491 void CStringCoveringDlg::OnLoadTargetFile()
492 {
493     // TODO: Add your control notification handler code here
494     ifstream target(target_set_file_name.c_str());
495
496
497     int n = 0;
498     target >> n;
```

```
499     assert (0 < n);
500
501
502     delete targetSet;
503     targetSet = NULL;
504
505     targetSet = new Population(NULL, n, Genome::GENOME_ZERO);
506
507
508
509     string s("");
510     int i = 0;
511     while (false == target.eof())
512     {
513         getline(target, s);
514         if (" " != s)
515         {
516             targetSet->_genomes.at(i).setFromPattern(s);
517
518             i++;
519
520         };
521
522     };
523
524
525     assert (n == i);
526
527     target_size = n;
528
529     assert (NULL != targetSet);
530
531     UpdateDialogFromValues();
532
533 }
534
535 void CStringCoveringDlg::OnLoadPatterns()
536 {
537     // TODO: Add your control notification handler code here
538     patterns.clear();
539     selected_patterns.clear();
540
541     ifstream patterns_stream(patterns_file.c_str());
542
543     string s("");
544     while (false == patterns_stream.eof())
545     {
546         getline(patterns_stream, s);
547         if (" " != s)
548         {
549             patterns.push_back(s);
550         };
551     };
552
553     UpdateDialogFromValues();
554
555 }
556
557 void CStringCoveringDlg::OnSelectPatterns()
558 {
559     // TODO: Add your control notification handler code here
560     vector<string> tmp(selected_patterns);
561
562     char s[2000];
563
564     for (int i = 0; i < m_patterns.GetCount(); i++)
565     {
566         if (0 != m_patterns.GetSel(i))
567         {
568             m_patterns.GetText(i, s);
569             const string what_to_insert(s);
570
571             bool sw = true;
572             for (int k = 0; k < tmp.size(); k++)
573             {
574                 if (what_to_insert == tmp.at(k)) // already in list
575                 {
576                     sw = false;
577                 };
578             };
579
580             if (true == sw)
```

```
582         {
583             tmp.push_back(what_to_insert);
584         };
585     };
586 };
587
588
589     selected_patterns = tmp;
590
591     UpdateDialogFromValues();
592 }
593
594 void CStringCoveringDlg::OnRemoveSelected()
595 {
596     // TODO: Add your control notification handler code here
597     vector <string> tmp;
598
599     char s[2000];
600
601     for (int i = 0; i < m_selected_patterns.GetCount(); i++)
602     {
603         if (0 == m_selected_patterns.GetSel(i)) // if not selected... we keep it
604         {
605             m_patterns.GetText(i, s);
606             const string what_to_insert(s);
607
608             tmp.push_back(what_to_insert);
609         };
610     };
611
612     selected_patterns = tmp;
613
614     UpdateDialogFromValues();
615 }
616
617 void CStringCoveringDlg::OnClearPatterns()
618 {
619     // TODO: Add your control notification handler code here
620     selected_patterns.clear();
621
622     UpdateDialogFromValues();
623 }
624
625 void CStringCoveringDlg::OnGenerateTargetSet()
626 {
627     // TODO: Add your control notification handler code here
628     if (false == UpdateValuesFromDialog1())
629     {
630         AfxMessageBox("Invalid parameters!", MB_OK | MB_ICONSTOP);
631     }
632     else
633     {
634         Population population(NULL, target_set_size, Genome::GENOME_ZERO);
635
636         const int patterns_no = selected_patterns.size();
637         assert (0 < patterns_no);
638
639         for (int i = 0; i < target_set_size; i++)
640         {
641             const int what_pattern = rand() % patterns_no;
642             assert ((0 <= what_pattern) && (what_pattern < patterns_no));
643
644             const string& p = selected_patterns.at(what_pattern);
645
646             population._genomes.at(i).setFromPattern(p);
647         };
648
649         {
650             ofstream of(target_set_file_name.c_str());
651             of << target_set_size << endl;
652             for (i = 0; i < target_set_size; i++)
653             {
654                 of << population._genomes.at(i)._body << endl;
655             };
656         };
657
658         ShellExecute(NULL, NULL, target_set_file_name.c_str(), NULL, NULL, SW_SHOWNORMAL);
659     }
660 }
661
662
663
664
```

```
665
666
667     };
668 }
669
670 void CStringCoveringDlg::OnDblclkPatterns ()
671 {
672     // TODO: Add your control notification handler code here
673     OnSelectPatterns ();
674 }
675
676 void CStringCoveringDlg::OnDblclkSelectedPatterns ()
677 {
678     // TODO: Add your control notification handler code here
679     OnRemoveSelected ();
680 }
681
682 void CStringCoveringDlg::OnKillfocusTsize ()
683 {
684     // TODO: Add your control notification handler code here
685     UpdateValuesFromDialog ();
686     UpdateValuesFromDialog1 ();
687 }
688
689 void CStringCoveringDlg::OnKillfocusPopulationSize ()
690 {
691     // TODO: Add your control notification handler code here
692     UpdateValuesFromDialog ();
693     UpdateValuesFromDialog1 ();
694 }
695
696
697 void CStringCoveringDlg::OnKillfocusGenomesNo ()
698 {
699     // TODO: Add your control notification handler code here
700     UpdateValuesFromDialog ();
701     UpdateValuesFromDialog1 ();
702 }
703
704
705 void CStringCoveringDlg::OnKillfocusMutationRate ()
706 {
707     // TODO: Add your control notification handler code here
708     UpdateValuesFromDialog ();
709     UpdateValuesFromDialog1 ();
710 }
711
712
713 void CStringCoveringDlg::OnKillfocusCrossoverRate ()
714 {
715     // TODO: Add your control notification handler code here
716     UpdateValuesFromDialog ();
717     UpdateValuesFromDialog1 ();
718 }
719
720
721 void CStringCoveringDlg::OnKillfocusNoOfGenerations ()
722 {
723     // TODO: Add your control notification handler code here
724     UpdateValuesFromDialog ();
725     UpdateValuesFromDialog1 ();
726 }
727
728
729 void CStringCoveringDlg::OnOK ()
730 {
731     // TODO: Add extra validation here
732
733     m_evolution.ReleaseDC (pDC);
734     pDC = NULL;
735
736     delete targetSet;
737     targetSet = NULL;
738
739     CDialog::OnOK ();
740 }
741
742 void CStringCoveringDlg::UpdateEvolutionPanel ()
743 {
744     assert ((0 <= gen_no) && (gen_no < number_of_generations));
745
746     m_gen_no.SetWindowText ((string ("generation: ") + stringFromInt (gen_no + 1)).c_str ());
747
```

```
748
749     m_matched_bits.SetWindowText((string("current matched bits: ") + stringFromDouble(matched_bits)).c_str());
750
751     m_progress.SetPos(((gen_no + 1) * 100) / number_of_generations);
752
753     m_bests.ResetContent();
754     for (int i = 0; i < bests.size(); i++)
755     {
756         m_bests.AddString(bests.at(i).c_str());
757     };
758
759
760     int x = (gen_no * w) / (number_of_generations - 1);
761     int y = ((MAX_GENOME LENGHT - matched_bits) * h) / MAX_GENOME LENGHT;
762
763     m_evolution.v.push_back(make_pair<int, int>(x, y));
764     m_evolution.Invalidate(FALSE);
765
766     last_x = x;
767     last_y = y;
768
769 }
770
771
772
773 unsigned int DoEvolution(void *data)
774 {
775     assert(NULL != mainDlg);
776
777     mainDlg->pDC->FillSolidRect(0, 0, mainDlg->w, mainDlg->h, background_color);
778
779     bool sw_stopped = false;
780
781     // GA(match_size, population_size, cross_over_rate, mutation_rate)
782     GA ga(mainDlg->match_size, mainDlg->population_size, mainDlg->crossover_rate, mainDlg->mutation_rate);
783
784     {
785
786         for (int i = 0; i < mainDlg->number_of_generations; i++)
787         {
788
789             ga.step();
790
791             mainDlg->gen_no = i;
792             mainDlg->matched_bits = ga.getAverageMatchedBits();
793
794             {
795                 bests.clear();
796                 bests.reserve(ga._populations.size());
797
798                 for (int k = 0; k < ga._populations.size(); k++)
799                 {
800                     assert(0 < ga._populations.at(k)._genomes.size());
801
802                     bests.push_back(ga._populations.at(k)._genomes.at(0)._body.to_string());
803
804                 };
805             };
806         };
807
808         mainDlg->UpdateEvolutionPanel();
809
810         if (false == mainDlg->running) // hmmm... someone stopped us
811         {
812             AfxMessageBox("Stopped...");
813
814             sw_stopped = true;
815             i = mainDlg->number_of_generations;
816             mainDlg->m_go.SetWindowText("GO");
817             mainDlg->EnableInterface();
818
819         };
820
821         if (MAX_GENOME LENGHT == mainDlg->matched_bits)
822         {
823             AfxMessageBox("You've reached GOD!\nEvolution cannot continue anymore.", MB_OK | MB_ICONINFORMATION);
824
825             i = mainDlg->number_of_generations;
826             mainDlg->m_go.SetWindowText("GO");
827             mainDlg->EnableInterface();
828
829         };
830     }
```

```
831         sw_stopped = true;
832         mainDlg->running = false;
833
834     };
835
836 };
837 };
838 };
839
840 if (false == sw_stopped)
841 {
842     AfxMessageBox("Done");
843
844     mainDlg->running = false;
845     mainDlg->m_go.SetWindowText("GO");
846     mainDlg->EnableInterface();
847
848     mainDlg->m_evolution.Invalidate();
849 };
850
851 assert (false == mainDlg->running);
852 return 0;
853 };
854
855 void CStringCoveringDlg::OnAbout()
856 {
857     // TODO: Add your control notification handler code here
858     CAboutDlg about;
859     about.DoModal();
860 }
861
862 void CStringCoveringDlg::DisableInterface()
863 {
864     m_target_set_size.EnableWindow(FALSE);
865     m_patterns.EnableWindow(FALSE);
866     m_selected_patterns.EnableWindow(FALSE);
867     m_clear_patterns.EnableWindow(FALSE);
868     m_remove_selected.EnableWindow(FALSE);
869     m_generate.EnableWindow(FALSE);
870     m_select_pattern.EnableWindow(FALSE);
871     m_load_patterns.EnableWindow(FALSE);
872     m_target_set_file.EnableWindow(FALSE);
873     m_load_target.EnableWindow(FALSE);
874     m_match_size.EnableWindow(FALSE);
875     spin->EnableWindow(FALSE);
876     m_mutation_rate.EnableWindow(FALSE);
877     m_crossover_rate.EnableWindow(FALSE);
878     m_number_of_generations.EnableWindow(FALSE);
879     m_ok.EnableWindow(FALSE);
880     m_population_size.EnableWindow(FALSE);
881 }
882
883 void CStringCoveringDlg::EnableInterface()
884 {
885     m_target_set_size.EnableWindow(TRUE);
886     m_patterns.EnableWindow(TRUE);
887     m_selected_patterns.EnableWindow(TRUE);
888     m_clear_patterns.EnableWindow(TRUE);
889     m_remove_selected.EnableWindow(TRUE);
890     m_generate.EnableWindow(TRUE);
891     m_select_pattern.EnableWindow(TRUE);
892     m_load_patterns.EnableWindow(TRUE);
893     m_target_set_file.EnableWindow(TRUE);
894     m_load_target.EnableWindow(TRUE);
895     m_match_size.EnableWindow(TRUE);
896     spin->EnableWindow(TRUE);
897     m_mutation_rate.EnableWindow(TRUE);
898     m_crossover_rate.EnableWindow(TRUE);
899     m_number_of_generations.EnableWindow(TRUE);
900     m_ok.EnableWindow(TRUE);
901     m_population_size.EnableWindow(TRUE);
902 }
903
904 void CStringCoveringDlg::OnClose()
905 {
906     // TODO: Add your message handler code here and/or call default
907     if (true == running)
908     {
909         AfxMessageBox("Evolution in progress.\nYou cannot stop it!", MB_OK | MB_ICONINFORMATION);
910         return;
911     }
912 }
913
```

```
914     };  
915  
916     CDialog::OnClose();  
917 }  
918
```

```
1 ////////////////////////////////////////////////////////////////////
2 // (c) 2003 Sorin OSTAFIEV (sorin@ostafiev.com) //
3 ////////////////////////////////////////////////////////////////////
4
5 #if !defined(AFX_MYSTATIC_H__8CF9223E_205D_484B_9D3B_FF7CC1DF1DC1__INCLUDED_)
6 #define AFX_MYSTATIC_H__8CF9223E_205D_484B_9D3B_FF7CC1DF1DC1__INCLUDED_
7
8 #if _MSC_VER > 1000
9 #pragma once
10 #endif // _MSC_VER > 1000
11 // MyStatic.h : header file
12 //
13
14 ////////////////////////////////////////////////////////////////////
15 // CMyStatic window
16
17 #include <vector>
18 using namespace std;
19
20 class CMyStatic : public CStatic
21 {
22 // Construction
23 public:
24     CMyStatic();
25
26 // Attributes
27 public:
28
29 // Operations
30 public:
31
32 // Overrides
33     // ClassWizard generated virtual function overrides
34     //{{AFX_VIRTUAL(CMyStatic)
35     //}}AFX_VIRTUAL
36
37 // Implementation
38 public:
39     COLORREF background_color;
40     COLORREF line_color;
41     int w, h;
42     virtual ~CMyStatic();
43
44     vector < pair<int, int> > v;
45     // Generated message map functions
46 protected:
47     //{{AFX_MSG(CMyStatic)
48     afx_msg BOOL OnEraseBkgnd(CDC* pDC);
49     afx_msg void OnPaint();
50     //}}AFX_MSG
51
52     DECLARE_MESSAGE_MAP()
53 };
54
55 ////////////////////////////////////////////////////////////////////
56
57 //{{AFX_INSERT_LOCATION}}
58 // Microsoft Visual C++ will insert additional declarations immediately before the previous line.
59
60 #endif // !defined(AFX_MYSTATIC_H__8CF9223E_205D_484B_9D3B_FF7CC1DF1DC1__INCLUDED_)
61
```

```
1 ///////////////////////////////////////////////////////////////////
2 // (c) 2003 Sorin OSTAFIEV (sorin@ostafiev.com) //
3 ///////////////////////////////////////////////////////////////////
4
5 // MyStatic.cpp : implementation file
6 //
7
8 #include "stdafx.h"
9 #include "stringcovering.h"
10 #include "MyStatic.h"
11
12 #ifdef _DEBUG
13 #define new DEBUG_NEW
14 #undef THIS_FILE
15 static char THIS_FILE[] = __FILE__;
16 #endif
17
18 ///////////////////////////////////////////////////////////////////
19 // CMyStatic
20
21 CMyStatic::CMyStatic():
22     w(0),
23     h(0)
24 {
25 }
26
27 CMyStatic::~CMyStatic()
28 {
29 }
30
31
32 BEGIN_MESSAGE_MAP(CMyStatic, CStatic)
33     //{AFX_MSG_MAP(CMyStatic)
34     ON_WM_ERASEBKGDND()
35     ON_WM_PAINT()
36     //}AFX_MSG_MAP
37 END_MESSAGE_MAP()
38
39 ///////////////////////////////////////////////////////////////////
40 // CMyStatic message handlers
41
42
43 BOOL CMyStatic::OnEraseBkgnd(CDC* pDC)
44 {
45     // TODO: Add your message handler code here and/or call default
46     pDC->FillSolidRect(0,0, w, h, background_color);
47     return true;
48
49     return CStatic::OnEraseBkgnd(pDC);
50 }
51
52 void CMyStatic::OnPaint()
53 {
54     CPaintDC dc(this); // device context for painting
55
56     // TODO: Add your message handler code here
57     CPen pen(PS_SOLID, 2, line_color);
58     CPen* old_pen = dc.SelectObject(&pen);
59
60     for (int i = 0; i < v.size(); i++)
61     {
62         const int x = v.at(i).first;
63         const int y = v.at(i).second;
64         if (0 == i)
65         {
66             dc.MoveTo(x, y);
67             dc.SetPixel(x, y, line_color);
68         }
69         else
70         {
71             dc.LineTo(x, y);
72         }
73     };
74
75 };
76
77 dc.SelectObject(old_pen);
78
79 // Do not call CStatic::OnPaint() for painting messages
80 }
81
```