1. **Bazele teoretice ale calculului evolutiv. Evoluţia căutare directă-căutare stochastică. Componentele si clasificarea algoritmilor evoluţionişti**

**Introducere**

Ideea de a aplica principiile darwiniste ale evoluţiei în rezolvarea automată a problemelor (*Problem Solving* - PS) datează din anii 1940, înaintea apariţiei calculatoarelor electronice. În 1948 Turing propunea o tehnică de rezolvare a problemelor numită *căutare genetică* sau *evolutivă*. În anii 1960 Fogel, Qwens şi Walsh introduceau conceptul de *programare evolutivă* (sau evoluţionistă), în timp ce Holland dezvolta *algoritmii genetici*. În aceeaşi perioadă, Rechenberg şi Schwefel introduceau *strategiile evolutive* (evoluţioniste) ca modalităţi alternative de rezolvare automată a problemelor. În anii 1990, Koza dezvoltă o nouă tehnică de căutare în spaţiul soluţiilor, *programarea genetică*. În terminologia actuală, întregul spectru de metode de rezolvare automată de inspiraţie darwinistă este desemnat prin termenul de *calcul evolutiv* (evoluţionist) şi include subdomeniile: programare evolutivă, strategii evolutive, algoritmi genetici şi programare genetică.

Calculul evolutiv este un domeniu al informaticii inspirat din procesul evoluţiei naturale; ideea care stă la baza calculului evolutiv este conexiunea evoluţie naturală – tehnica de rezolvare a problemelor de tip experiment-eroare (sau generare-testare). Cu alte cuvinte, într-un mediu dat, indivizii constituiţi într-o populaţie intră în competiţie pentru a supravieţui şi a se reproduce. Abilitatea indivizilor de a-şi atinge aceste scopuri în mediul în care trăiesc este strict corelată cu şansele lor de supravieţuire şi multiplicare şi determină evoluţia în timp a populaţiei. În contextul modalităţii de rezolvare a problemelor de tip generare-testare stochastice, populaţia este modelată ca o colecţie de elemente candidat la soluţie. Calitatea candidaţilor la soluţie, definită în termenii gradului în care fiecare element rezolvă problema, determină şansa lor de a fi menţinuţi şi utilizaţi pentru construirea unor noi candidaţi.

**Suportul de natură biologică al calculului evolutiv**

Teoria evoluţionistă a lui Darwin oferă o explicaţie a diversităţii biologice şi a mecanismului care stă la baza acesteia. În centrul interpretării macroscopice a evoluţiei este plasată selecţia naturală. Considerând un mediu care poate susţine un număr limitat de indivizi şi instinctul primar al fiecărui individ de a se reproduce, procesul de selecţie este esenţial şi inevitabil în controlul dimensiunii populaţiei. Selecţia naturală favorizează indivizii cei mai competitivi în însuşirea resurselor, adică acei indivizi care sunt cel mai bine adaptaţi condiţiilor de mediu. Fenomenul este cunoscut drept supravieţuirea celor mai bine adaptaţi/ potriviţi (survival of the fittest).

Teoria evoluţiei are ca principii fundamentale selecţia bazată pe competiţie şi variaţiile de *fenotip* în rândul membrilor populaţiei. Fenotipul este ansamblul de însuşiri şi caractere care se manifestă în mod vizibil la un individ şi care este determinat pe bază ereditară şi de condiţiile de mediu (DEX). Caracteristicile fenotipului unui individ determină gradul lui de adaptabilitate la condiţiile de mediu (fitness). Fiecare individ reprezintă o combinaţie unică de caracteristici ale fenotipului şi este evaluat de condiţiile de mediu. Dacă evaluarea este favorabilă, atunci fenotipul individului este propagat spre urmaşi (progenituri), altfel caracteristicile fenotipului dispar şi individul moare fără a se putea reproduce. Viziunea lui Darwin despre evoluţie este aceea că, în procesul trecerii de la o generaţie la alta prin reproducere, apar mutaţii (variaţii) mici, aleatoare, în caracteristicile fenotipului. Ca rezultat al acestor variaţii, apar şi sunt evaluate noi combinaţii de caracteristici ale fenotipului. Cele mai bune dintre ele supravieţuiesc şi se reproduc şi în acest mod evoluţia conduce la progres.

Modelul primar poate fi deci rezumat după cum urmează. O populaţie este formată dintr-un număr de indivizi, priviţi ca unităţi de selecţie; succesul fiecărui individ în încercarea de a se reproduce depinde de cât de bine este adaptat condiţiilor de mediu comparativ cu restul indivizilor. Pe parcursul reproducerii celor mai buni (bine adaptaţi la mediu) indivizi, apar mutaţii ocazionale, care generează noi indivizi, ce vor fi ulterior evaluaţi. În consecinţă, pe măsură ce timpul trece, se produc schimbări în structura populaţiei, cu alte cuvinte populaţia reprezintă o unitate a evoluţiei. Întregul proces poate fi asimilat din punct de vedere intuitiv cu modelul unui peisaj adaptiv (dinamic) (respectiv o suprafaţă adaptive) în spaţiul 3D (0-x-y-z). Fiecare punct p(x,y,z) al suprafeţei este asimilat unui individ, unde în planul x-y este figurată combinaţia de caracteristici ale fenotipului individului, iar altitudinea lui p (valoarea lui pe axa 0z) corespunde nivelului de adaptabilitate (fitness) a individului reprezentat de p. În acest context, evoluţia este procesul de “avans” al populaţiei spre zone aflate la o altitudine mai mare, acest avans fiind realizat pe baza mutaţiilor şi selecţiei naturale. Este obţinută astfel legătura cu conceptul de *probleme multimodale*, adică probleme în care există o mulţime de puncte de optim local (superioare tuturor soluţiilor din vecinătatea lor), cel mai bun element al mulţimii fiind optimul global. O problemă în care există un singur optim local este numită *unimodală*.

Legătura dintre procesul evoluţiei şi un proces de optimizare este pe cât de directă, pe atât de înşelătoare, pentru că evoluţia nu presupune întotdeauna creşterea globală a calităţii populaţiei (în termenii funcţiei de fitness). Deoarece, la fiecare epocă, populaţia este finită şi operatorii de selecţie şi mutaţie includ componente aleatoare, poate fi constatată o inactivitate genetică, manifestată fie prin dispariţia din populaţie a unor indivizi foarte adaptaţi condiţiilor de mediu, fie prin variaţia foarte mică sau chiar lipsa de variaţie a unor caracteristici ale fenotipului indivizilor din populaţie. Unul din efectele posibile este acela al concentrării indivizilor într-o zonă de adaptabilitate scăzută. Efectul rezultat prin combinarea procesului de selecţie cu inactivitatea genetică poate conduce în egală măsură la creşterea nivelului de adaptabilitate globală a populaţiei, respectiv la descreşterea acestuia. În plus, nu există nici o garanţie că, dacă evoluţia duce la creşterea calităţii populaţiei, nivelul de optim local nu a fost deja atins înaintea unui fenomen de inactivitate genetică. În scopul evitării “ciclării” evoluţiei populaţiei într-o regiune de optim local, au fost dezvoltate o serie de teorii, una dintre cele mai cunoscute fiind teoria lui Wright conform căreia poate fi determinat optimul global în cazul unei suprafeţe fixe (teorie referită drept *shifting balance*).

**Tipuri de probleme ce pot fi rezolvate pe baza calculului evolutiv**

În literatura de specialitate sunt evidenţiate două clase de metode PS de inspiraţie biologică: calculul neuronal (*neurocomputing*), prin intermediul căruia problemele sunt rezolvate imitând modul de funcţionare (raţionament) a (al) creierului uman şi procesele de tip evolutiv, problemele fiind rezolvate prin imitarea creării creierului uman.

În general, un sistem funcţional de rezolvare automată a problemelor conţine trei componente: datele de intrare, datele de ieşire şi modulul intern care conectează primele două componente. Modalitatea de funcţionare a modelului permite explicarea modului de funcţionare a sistemului, în sensul că, răspunsul sistemului poate fi calculat pentru orice date de intrare specificate. În funcţie de componentele din sistem cunoscute, pot fi diferenţiate următoarele tipuri de probleme:

* Probleme de optimizare: sunt cunoscute modelul şi datele de ieşire dorite (respectiv o descriere a acestora), iar problema este de a determina datele de intrare care corespund rezultatelor dorite. Un exemplu de astfel de problemă este cea a comis voiajorului (în care trebuie determinat cea mai scurtă sau ieftină rută care să lege un număr dat de oraşe): modelul este cunoscut şi corespunde formulei de calcul a lungimii unei rute date, în care lungimea (sau costul) calculată (calculat) este data de ieşire. Proprietatea pe care rezultatul trebuie să o îndeplinească este un criteriu de optim (lungime minimă), iar problema este de a determina acea dată de intrare, corespunzătoare unei rute, care să conducă la rezultatul dorit. O problemă de optimizare rezolvată cu succes prin calcul evolutiv este generarea orarului în cadrul unei universităţi. În cursul unei zile sunt programate în general mii de activităţi, restricţiile pe care trebuie să la îndeplinească o programare corectă a acestora fiind multiple iar soluţiile fezabile ale problemei în număr foarte mic relativ la mulţimea tuturor programărilor posibile.
* Probleme de modelare sau de identificare a sistemului: sunt cunoscute datele de intrare şi rezultatele corespunzătoare lor, iar modelul este necunoscut. Modelul trebuie determinat astfel încât, pentru fiecare intrare dată, să calculeze rezultatul corect. Un exemplu de astfel de problemă: clasificarea supervizată în cazul modelului cu două clase. Problema este de a determina un clasificator care să separe corect elementele celor două clase. Datele de intrare corespund elementelor celor două clase, pentru fiecare dată de intrare rezultatul fiind eticheta clasei de provenienţă. Identificarea modelului revine la determinarea unei funcţii de decizie, care, de exemplu, să calculeze valori pozitive pentru exemplele care provin din prima clasă, respectiv valori negative pentru celelalte. Cu alte cuvinte, în acest caz scopul este de a determina o formulă (în acest caz expresia analitică a unei funcţii de decizie) care să „lege” datele de intrare (cunoscute) de rezultate (cunoscute). Problemele de acest tip apar în medii în care sunt disponibile foarte multe date (observaţii, înregistrări etc.), de exemplu în situaţii în care există un set de dimensiuni considerabile de observaţii/înregistrări asupra/ relative la un fenomen/ eveniment. Identificarea modelului care să explice conexiunile dintre datele de intrare şi rezultate trebuie realizată şi astfel încât acesta să asigure o capacitate de generalizare rezonabilă (pentru noi date de intrare, sistemul trebuie să furnizeze în general răspunsuri corecte). Astfel de probleme sunt cele din domeniile instruirii automate (machine learning) şi data mining.
* Probleme de simulare: sunt cunoscuţi modelul şi o serie de date de intrare şi cerinţa este de a determina datele de ieşire corecte, corespunzătoare intrărilor date. Un exemplu de probleme de simulare care pot fi rezolvate utilizând calculul evolutiv sunt cele în care este căutat răspunsul la întrebări de tipul “ce se întâmplă dacă” (what-if questions), în condiţiile în care problema subiectului investigat evoluează (în termenii operaţiilor de variaţie şi selecţie). Economia evolutivă este un domeniu de cercetare relativ nou, care, în principiu, are la bază ideea că jocul şi jucătorii din arena socio-economică sunt asemenea evoluţiei vieţii (a “jocului” şi “jucătorilor” ei).

1. **Algoritmi evolutivi**

În literatura de specialitate sunt prezentate diverse clase de algoritmi evolutivi (EA – Evolutionary Algorithms), toate având la bază acelaşi principiu: dată fiind o populaţie de indivizi, influenţa mediului determină un proces de selecţie naturală (indusă de adaptabilitatea fiecărui individ la mediu), care are ca efect creşterea globală a calităţii populaţiei, exprimată prin intermediul funcţiei de fitness.

* 1. **Schema generală a algoritmilor evolutivi**

Dacă este cunoscută o funcţie de tip calitate care trebuie maximizată, setul iniţial de candidaţi la soluţie (elemente din mulţimea ) poate fi generat aleator, fiind obţinută astfel populaţia iniţială, (). Pentru fiecare , reprezintă calitatea candidatului *x* (ca măsură abstractă de evaluare – funcţie de tip fitness). Populaţia următoare este determinată pe baza funcţiei de evaluare prin aplicarea următorului mecanism. Este selectată , o submulţime a lui formată din cei mai “buni” membri ai populaţiei curente, adică din acele elemente cu cele mai bune scoruri obţinute în urma evaluării prin . Membrii mulţimii generează populaţia următoare prin aplicarea operatorilor de recombinare şi/sau mutaţie. Operaţia de recombinare (*recombination*) este definită pentru doi sau mai mulţi indivizi din mulţimea (numiţi părinţi) şi are ca rezultat unul sau mai mulţi noi candidaţi la soluţie (numiţi copii). Operatorul mutaţie este definit pe un element al mulţimii şi are drept rezultat un nou candidat la soluţie. Prin aplicarea operatorilor de recombinare şi mutaţie sunt generaţi noi indivizi (numiţi progenituri ai mulţimii ) care intră în competiţie, pe baza măsurii de fitness (posibil şi a vârstei), cu elementele populaţiei pentru obţinerea „unui loc” în populaţia următoare, . Procesul poate fi iterat fie până la obţinerea unui candidat suficient de bun (soluţia problemei, corespunzând unui punct de maxim al funcţiei ), fie până la atingerea unei limite de calcul date.

În cadrul acestui proces intervin două elemente fundamentale care constituie baza sistemelor evoluţioniste:

* *operatorii de variaţie* (recombinare şi mutaţie), care asigură diversitatea necesară creării de indivizi cu caracteristici noi şi
* *selecţia*, care “forţează” creşterea calităţii indivizilor unei populaţii.

Combinarea aplicării operatorilor de variaţie şi selecţie determină în general creşterea calităţii globale de la o populaţie la populaţia următoare. Un astfel de algoritm poate fi privit ca o evoluţie a procesului de optimizare prin “apropierea” succesivă de valoarea optimă. Alternativ, procesul evolutiv poate fi considerat un proces de adaptare. Din această perspectivă, măsura de adaptabilitate (de fitness) este o expresie a cerinţelor mediului în care evoluează populaţia şi nu o funcţie obiectiv care trebuie optimizată. Măsura în care sunt atinse cerinţele mediului este direct proporţională cu măsura de viabilitate şi este reflectată în numărul de progenituri. Procesul evolutiv determină obţinerea de populaţii succesive din ce în ce mai bine adaptate mediului în care trăiesc.

O serie de componente ale unui proces evolutiv sunt stochastice. De exemplu, deşi indivizii mai bine adaptaţi mediului au o şansă mai mare să fie selectaţi pentru generarea de noi candidaţi soluţie, în cele mai multe tipuri de implementări evolutive şi indivizii mai “slabi” au o şansă de a deveni părinte sau de a supravieţui (în sensul selectării lor în populaţia sau generaţia următoare). De asemenea, în cadrul operaţiei de recombinare, alegerea perechilor sau *n*-tuplurilor (secvenţe de *n* elemente) de indivizi care interschimbă material genetic, dar şi părţile (fracţiunile) de material genetic interschimbat sunt aleatoare. În mod similar, la efectuarea unei mutaţii, atât porţiunile din individ care vor suferi mutaţia cât şi noile părţii care le înlocuiesc sunt alese aleator.

În continuare este prezentată schema generală a unui algoritm evolutiv.

Pas1. *Iniţilizarea populaţiei*: este obţinută prin generarea aleatoare a candidaţilor la soluţie

Pas2. *Evaluarea candidaţilor*: pentru fiecare , determină

Pas3. *Repetă*

3.1. *Selectează* mulţimea de părinţi

3.2. *Recombină* perechi (sau *n*-tupluri) de părinţi

3.3. Efectuează *mutaţii* asupra progeniturilor rezultate

3.4. *Evaluează* noii candidaţi la soluţie

3.5. Selectează indivizii pentru constituirea generaţiei următoare, ;

3.6.

*până când* (*Condiţie\_terminare* este satisfăcută)

**Observaţii**

1. Schema prezentată mai sus corespunde familiei metodelor PS de tip generare-testare, pe baza următoarelor consideraţii. În cadrul algoritmilor evolutivi sunt procesaţi simultan membrii unei întregi populaţii, combinarea informaţiilor oferite de doi sau mai mulţi candidaţi fiind realizată în principal prin operaţia de recombinare. De asemenea, algoritmii evolutivi sunt de tip stochastic.
2. Funcţia de evaluare (de fitness) este o estimare de tip euristic a calităţii fiecărui membru al populaţiei curente, iar procesul de căutare este dirijat de operatorii de variaţie şi selecţie.

Diferitele tipuri de algoritmi evolutivi menţionaţi în partea 1 respectă schema generală prezentată mai sus şi diferă între ele printr-o serie de detalii tehnice, cum este de exemplu modalitatea de reprezentare a unui candidat la soluţie (tipul sau structura de date utilizat/utilizată pentru reprezentarea membrilor populaţiei). În cazul clasei algoritmilor genetici (GA – Genetic Algorithms), un candidat la soluţie este reprezentat prin intermediul unui şir definit pe un alfabet finit. Strategiile evolutive (ES- Evolution Strategy) utilizează vectori cu numere reale pentru a reprezenta membrii populaţiei, în timp ce în programarea evolutivă (EP – Evolutionary Programming) sunt utilizate reprezentările prin maşini cu stări finite. Clasa algoritmilor GP (programare genetică) este dezvoltată pe baza reprezentărilor candidaţilor la soluţie prin intermediul structurii de arbore. Selectarea unei reprezentări a membrilor populaţiei în detrimentul altor variante posibile este realizată astfel încât să fie cel mai bine potrivită problemei particulare de rezolvat, în sensul că uşurează implementarea algoritmului evolutiv sau este cea mai naturală relativ la problema dată. Evident, selectarea operatorilor de recombinare şi mutaţie ţine cont de varianta aleasă pentru reprezentarea candidaţilor la soluţie.

* 1. **Componentele algoritmilor evolutivi**

Componentele de bază ale algoritmilor evolutivi sunt: *reprezentarea* (definirea membrilor populaţiei), *funcţia de evaluare* (de tip fitness), *populaţia*, *mecanismul de selectare a părinţilor* (indivizii care interschimbă material genetic), *operatorii de variaţie* (recombinarea şi mutaţia), *mecanismul de selectare a membrilor generaţiei următoare* (actualizarea populaţiei), *definirea modulului de iniţializare* (modalitatea de determinare a populaţiei iniţiale) şi *definirea condiţiei terminale*.

**Reprezentarea**

Prima etapă în dezvoltarea unui algoritm evolutiv este stabilirea unei conexiuni între contextul problemei particulare de rezolvat şi spaţiul în care evoluează tehnica PS considerată.   
Obiectele care formează soluţiile posibile în contextul problemei de rezolvat sunt numite *fenotipuri*, reprezentarea lor în contextul spaţiului EA fiind referită prin *genotip*. Scopul este de a stabili o corespondenţă între mulţimea fenotipurilor şi cea a genotipurilor, numită reprezentare. De exemplu, dacă problema particulară este de optimizare în mulţimea numerelor întregi, un set prestabilit de numere întregi poate constitui mulţimea fenotipurilor, în timp ce setul genotipurilor este constituit din reprezentarea binară a fiecărui fenotip.

**Observaţie.** Spaţiul fenotipurilor poate fi foarte diferit de cel al genotipurilor, cel în care evoluează un EA. O soluţie este corespunzătoare unui fenotip “bun” şi este obţinută prin decodificarea celui mai bun (din punctul de vedere al funcţiei de evaluare) genotip rezultat în urma aplicării EA.

Ca terminologie, obiectele aparţinând spaţiului problemei particulare de rezolvat sunt referite prin candidaţi la soluţie, fenotipuri sau indivizi. Spaţiul pe care este definită problema este numit spaţiul fenotipurilor. Obiectele ce aparţin spaţiului în care este dezvoltat un EA sunt referite prin termenii de genotipuri, cromozomi sau indivizi. Spaţiul în care evoluează EA este numit spaţiul genotipurilor. În general, un genotip este constituit din mai multe elemente, numite valori sau *alele*, fiecare fiind plasat într-o anumită poziţie, referită prin termenul de variabilă sau *genă*.

**Observaţie.** Termenul reprezentare este utilizat în două moduri diferite. În unele situaţii desemnează transformarea aplicată spaţiului fenotipurilor astfel încât să fie obţinut spaţiul genotipurilor, caz în care termenul utilizat este şi cel de *codificare* (în exemplul considerat mai sus, fiecare genotip este codificarea binară a unui fenotip). Transformarea inversă, aplicată spaţiului genotipurilor pentru a obţine spaţiul fenotipurilor este numită *decodificare*. Evident, în acest caz reprezentarea trebuie să fie inversabilă: fiecărui genotip trebuie să îi corespundă cel puţin un fenotip. În alte situaţii, în definirea unei reprezentări accentul este pus cu precădere pe structura de date utilizată pentru definirea spaţiul genotipurilor şi nu pe transformarea propriu-zisă. Această interpretare este legată spre exemplu de definirea operatorului mutaţie pe spaţiul genotipurilor constituite din reprezentările binare ale fenotipurilor.

**Funcţia de evaluare**

Rolul funcţiei de evaluare (de fitness) este de a măsura gradul de adaptabilitate a fiecărui individ la mediul în care trăieşte, mai exact este de definire a noţiunii de calitate. Funcţia de evaluare stă la baza procesului de selecţie şi, din perspectiva tehnicilor PS, reprezintă modulul de rezolvare a problemei date în contextul evolutiv. Din punct de vedere tehnic, este o funcţie care asociază fiecărui genotip o măsură a calităţii şi, în general, este derivată pe baza unei funcţii de tip calitate definită pe spaţiul fenotipurilor. De exemplu, dacă este funcţia de calitate definită pe spaţiul fenotipurilor, format din numere din mulţimea şi fiecare genotip este reprezentarea binară a unui fenotip, atunci funcţia de evaluare în spaţiul genotipurilor este definită prin,

De exemplu, pentru , atunci .

În cele mai multe situaţii, problema de rezolvat utilizând EA revine la o problemă de optimizare. Dacă funcţia obiectiv trebuie minimizată, atunci este realizată o transformare a ei astfel încât problema de optim să fie una de maxim (din punct de vedere matematic, de exemplu, a minimiza o funcţie *f* este echivalent cu a maximiza funcţia –*f* sau, în situaţia în care *f* nu se anulează pe spaţiul fenotipurilor, cu a maximiza funcţia ). În acest caz, funcţia de evaluare este definită pe baza funcţiei obiectiv şi ţinând cont de reprezentarea fenotipurilor în spaţiul EA.

**Observaţie**. În continuare, prin cel mai bun individ al unei populaţii vom înţelege acel individ care realizează maximul funcţiei de evaluare pe acea populaţie.

**Populaţia**

Rolul populaţiei în dezvoltarea EA este de a menţine o mulţime de genotipuri corespunzătoare unor soluţii posibile. O populaţie este un multiset (o mulţime de elemente nu neapărat distincte) de genotipuri. Indivizii unei populaţii sunt obiecte statice, în sensul că nu pot fi modificaţi şi nu se pot adapta mediului în care trăiesc. Aceste proprietăţi le are, în schimb populaţia. Dacă este stabilit modul de reprezentare (spaţiul genotipurilor), populaţia poate fi definită prin specificarea numărului de indivizi care o compun. În situaţia unor EA complecşi, populaţiei îi este asociată şi o structură spaţială adiţională, definită prin intermediul unei funcţii de tip distanţă sau prin relaţii de tip vecinătate. În astfel de cazuri, definirea populaţiei trebuie însoţită de specificarea structurii spaţiale asociate. Operatorii genetici de selecţie (selecţia indivizilor care interschimbă material genetic – părinţi- şi selecţia populaţiei la momentul de timp următor) sunt definiţi la nivelul unei populaţii şi, în general, construcţia lor presupune consultarea întregii populaţii curente. De exemplu, cel mai bun individ al unei populaţii date poate fi selectat pentru a genera populaţia următoare sau cel mai slab individ al unei populaţii date este înlocuit cu unul nou. În cele mai multe situaţii, EA folosesc populaţii de dimensiune constantă pe tot parcursul evoluţiei.

Diversitatea unei populaţii este măsurată în termenii numărului de indivizi distincţi ai populaţiei. Există mai multe variante de definire a măsurii de diversitate: numărul valorilor distincte ale funcţiei de evaluare (deşi, dacă , nu rezultă ), numărul fenotipurilor diferite reprezentate în cadrul populaţiei (deşi prezenţa unui fenotip în spaţiul iniţial nu garantează prezenţa unui singur genotip în spaţiul EA: în cadrul populaţiei, repetarea unui genotip este echivalentă fie cu selectarea pentru includere a unui fenotip de mai multe ori, la generarea populaţiei iniţiale, respectiv cu selectarea repetată a unui genotip în construirea unei noi populaţii, de exemplu datorită valorii mari a funcţiei de evaluare corespunzătoare lui), numărul genotipurilor diferite din populaţie (un genotip corespunde unui singur fenotip şi valoarea funcţiei de evaluare corespunzătoare lui este unică), măsuri bazate pe entropia populaţiei ş.a.m.d.

**Mecanismul de selectare a părinţilor**

Rolul operatorului de selectare a părinţilor este de a distinge între indivizii populaţiei pe baza calităţii acestora, în particular de a permite celor mai buni indivizi să se reproducă, deci să participe la generarea populaţiei următoare. Alături de operatorul de selecţie a „supravieţuitorilor” (indivizii care vor compune generaţia următoare), mecanismul de selecţie a părinţilor forţează îmbunătăţirea calităţii globale a populaţiei de la o generaţie la alta. În calculul evolutiv, selectarea părinţilor este de tip probabilist: alegerea unui individ pentru a se reproduce depinde direct proporţional de calitatea lui, deci un individ are şanse mai mari de a se reproduce comparativ cu cei inferior lui din punct de vedere calitativ.

**Observaţie**. Indivizii “slabi” (cu valori mici ale funcţiei de evaluare) nu sunt eliminaţi din procesul de selectare pentru reproducere, ci doar au asociate probabilităţi mici, dar nenule, de selecţie. În acest fel algoritmul de căutare nu este de tip greedy şi riscul de identifica o valoare de optim local fiind diminuat în acest fel.

**Operatorii de variaţie: mutaţia şi recombinarea**

Scopul aplicării operatorilor de variaţie este de a crea noi indivizi, derivaţi din cei ai populaţiei curente. Din punctul de vedere al rezolvării problemelor prin metode de căutare de tip generare-testare, prin aplicarea operatorilor de variaţie este realizată faza de generare. Definirea operatorilor de variaţie depinde esenţial de modalitatea de reprezentare a spaţiului iniţial (definirea spaţiului genotipurilor).

**Operatorul mutaţie**

Mutaţia este operator unar (cu aritate 1), în urma aplicării acestuia asupra unui genotip rezultă o variantă “mutantă”, numită progenitură sau copil. Operatorul mutaţie este întotdeauna stochastic, rezultatul depinzând de o serie de alegeri aleatoare. În general aceste alegeri constau în utilizarea unui generator de numere aleatoare din diferite distribuţii de probabilitate şi sunt numite extrageri aleatoare. Rolul mutaţiei în calculul evolutiv depinde de tipul de algoritm implementat. De exemplu, în cazul algoritmilor genetici, mutaţia are rolul de a „împrospăta” structura genetică a unei populaţii, în cazul programării evolutive este unicul operator de variaţie care dirijează procedura de căutare, în timp ce în cazul programării genetice în general nu este folosit.

Din studiul teoretic al convergenţei algoritmilor evolutivi rezultă că optimul global al funcţiei obiectiv poate fi obţinut în situaţia în care operatorii de variaţie utilizaţi asigură obţinerea oricărui genotip soluţie potenţială a problemei de optim (Eiben, Smith, 2003). Cea mai simplă cale de a asigura îndeplinirea acestei codiţii este de a utiliza un operator mutaţie care să permită modificarea oricărei alele dintr-un cromozom cu orice variantă posibilă, cu o probabilitate nenulă. În literatura de specialitate există însă şi opinii conform cărora rezultatele teoretice relative la comportamentul EA au o importanţă practică redusă şi multe implementări EA nu posedă proprietăţile cerute de acestea.

**Operatorul de recombinare**

Un operator de variaţie binar (cu aritate 2) este numit operator de recombinare sau încrucişare şi are ca efect obţinerea unuia sau a două genotipuri urmaş direct prin combinarea informaţiei purtate de două genotipuri părinte. Recombinarea este un operator stochastic: alegerea acelor părţi ale genotipurilor părinţi care vor fi combinate şi modalitatea de recombinare rezultă în urma unor extrageri aleatoare. Rolul recombinării diferă de la o clasă de algoritmi evolutivi la alta: în cadrul algoritmilor genetici este cel mai utilizat operator de variaţie (probabilitatea de efectuarea a unei încrucişări este în general mult mai mare decât probabilitatea apariţiei unei mutaţii), în programarea genetică este în general unicul operator de variaţie folosit, în timp ce în programarea evolutivă nu este implementat.

În dezvoltări de tip EA pot fi folosiţi şi operatori de recombinare de aritate mai mare decât 2 (în generarea urmaşilor sunt folosiţi mai mult de două genotipuri părinte). Astfel de operatori sunt uşor de implementat dar nu au corespondent biologic. Deşi o serie de studii indică utilitatea acestora în tratarea unor probleme particulare, aceşti operatori sunt rar folosiţi.

Prin împerecherea a două genotipuri părinte cu caracteristici diferite şi superioare calitativ pot fi obţinute progenituri care să îmbine caracteristicile celor doi părinţi. Acest principiu are un fundament biologic extrem de solid: a fost utilizat de cultivatorii de plante şi crescătorii de animale pentru a produce specii cu randament superior sau care să prezinte caracteristici îmbunătăţite. Aplicarea EA determină crearea de urmaşi direcţi prin încrucişări aleatoare, fiind acceptată ideea că unii dintre aceştia pot avea însuşiri nedorite, majoritatea pot fi calitativ similari sau chiar inferiori părinţilor şi doar o mică parte dintre ei pot avea caracteristici superioare părinţilor.

**Mecanismul de selectare a supravieţuitorilor (înlocuirea populaţiei curente)**

Rolul acestui operator, numit şi selecţia mediului sau strategia de înlocuire a populaţiei curente, este de a diferenţia indivizii în funcţie de calitatea lor. Din acest punct de vedere este similar procesului de selecţie a părinţilor dar este utilizat într-o etapă diferită a evoluţiei unui EA. Mecanismul de selecţie a membrilor următoarei generaţii este aplicat după generarea progeniturilor indivizilor populaţiei curente şi, deoarece dimensiunea populaţiei este în general constantă în timp, revine la aplicarea unei funcţii de decizie fiecărui individ aparţinând populaţiei curente sau mulţimii progeniturilor. Funcţia de decizie aplicată unui individ exprimă proprietatea acestuia de a fi selectat pentru includerea în populaţia următoare (proprietatea de a fi supravieţuitor) şi este de obicei construită pe baza funcţiei de evaluare, luând în calcul calitatea fiecărui individ şi, în unele situaţii, factorul vârstă (de câte generaţii este menţinut u individ).

În general selecţia mediului este un proces determinist. Obţinerea generaţiei următoare poate fi realizată, de exemplu, fie prin ordonarea indivizilor multisetului obţinut prin reuniunea populaţiei curente cu multisetul progeniturilor şi selectarea celor mai buni indivizi (funcţie de decizie bazată pe funcţia de evaluare), fie prin selectarea indivizilor exclusiv din multisetul urmaşilor direcţi (funcţie de decizie bazată pe factorul vârstă).

**Iniţializarea**

În majoritatea EA, crearea populaţiei iniţiale este realizată prin generare aleatoare de fenotipuri şi apoi obţinerea multisetului de genotipuri asociat. De asemenea, în funcţie de problema particulară de rezolvat, generarea populaţiei iniţiale poate fi realizată şi pe baza unor euristici care să asigure obţinerea unor indivizi cu adaptabilitate ridicată.

**Condiţia terminală**

Condiţia terminală în EA este stabilită în funcţie de tipul de problemă de rezolvat, în felul următor. Dacă problema are o valoare de optim cunoscută, atunci un posibil criteriu de oprire este atingerea acelei valori sau atingerea acelei valori cu o eroare dată . Dar, deoarece algoritmii evolutivi sunt stochastici şi nu garantează atingerea valorii optime, criteriul poate să nu fie satisfăcut la nici o iteraţie, deci el trebuie reformulat. Cele mai utilizate opţiuni sunt:

* atingerea unui număr maxim de iteraţii (generaţii);
* atingerea unui număr maxim de evaluări ale calităţii indivizilor;
* pentru o anumită perioadă de timp (un număr de iteraţii specificat sau un număr de evaluări specificat) calitatea populaţiei curente nu este semnificativ îmbunătăţită (este sub un prag dat);
* diversitatea populaţiei scade sub un prag dat.

În situaţia în care problema de rezolvat nu are un optim cunoscut, poate fi utilizată oricare din variantele menţionate mai sus.

* 1. **Evoluţia căutare directă-căutare stochastică. Metodele hill climbing şi simulated annealing**

Evoluţia căutare directă-căutare stochastică cuprinde două tehnici care reduc din dezavantajele căutărilor directe, şi anume metode de tip “hill climbing” şi “simulated annealing”.

Metodele de tip “hill climbing” utilizează o tehnică de iterativitate îmbunătăţită. Aceasta se aplică unui singur punct din spaţiul de căutare. La o iteraţie este selectat un nou punct aflat într-o vecinătate a punctului curent procesat. Dacă acest punct determină o valoare mai bună (din punct de vedere al criteriului de optim considerat) pentru funcţia obiectiv, el devine punct curent. În caz contrar, este selectată o altă vecinătate a punctului curent, procesul desfăşurându-se ulterior similar. Algoritmul se încheie când nici un punct vecin celui curent nu aduce îmbunătăţiri valorilor funcţiei obiectiv. Metodele de acest tip conduc de obicei la valori de optim local, depinzând de punctul de start. În plus, nu se pot furniza informaţii referitoare la eroarea relativă a soluţiei calculate. Pentru a creşte performanţele unor astfel de modele, acestea se utilizează pentru un număr mare de punct de start.

Metodele de tip“simulated annealing” elimină o mare parte din dezavantajele algoritmilor “hillclimbing”, în sensul că soluţiile nu depind de punctul de start şi sunt de obicei apropiate de punctul de optim global. Pentru aceasta, este considerată o probabilitate de acceptare a punctului selectat drept următor punct curent, egală cu 1 dacă noul punct furnizează o valoare mai bună pentru funcţia obiectiv considerată. În unele situaţii, probabilitatea de a accepta un nou punct este o funcţie cu valori corespunzătoare funcţiei obiectiv pentru punctul curent şi noul punct selectat. De asemenea, faţă de tehnica “hill climbing”, este considerat un parametru de tip temperatura sistemului, care influenţează probabilitatea de acceptare a unui nou punct ca punct curent: cu cât acest parametru este mai scăzut, cu atât şansele de acceptare sunt mai mici. Pe parcursul execuţiei algoritmului, temperatura sistemului scade; algoritmul se încheie pentru o temperatură mică, pentru care nu se mai acceptă nici o modificare a soluţiei (probabilitatea de acceptare a unui nou punct este 0).

**Structurile algoritmilor de tip “hill climbing” şi “simulated annealing”**

Fie *f* funcţia obiectiv care se doreşte maximizată (dacă principiul de optimalitate este minimul, atunci problema minimizării funcţiei obiectiv *f* în anumite condiţii de constrângere revine la maximizarea funcţiei *–f*) . Considerând reprezentarea cromozomială a unei soluţii potenţiale de tip binar, ca un şir de *nr* biţi, punctele curent respectiv nou fiind desemnate prin *vc* respectiv *vn*, algoritmii „hill climbing” şi „simulated annealing” sunt descrişi după cum urmează.

Fie MAX numărul punctelor de start. Procedura următoare calculează *v*, un cel mai bun punct din cele MAX puncte obţinute în urma aplicării tehnicii de tip “hillclimbing” (memorate în vectorul *V*).

#### procedure hillclimbing

*begin*

*t=0*

*repeat*

*local=false*

*selectează aleator un punct curent vc*

*evaluează vc*

*repeat*

*selectează nr puncte din vecinătatea lui vc prin modificarea fiecărui bit al lui vc*

*selectează un punct vn dintre cei nr generaţi anterior, cu funcţia obiectiv maximă*

*if f(vc)<f(vn) vc=vn*

*else local=true*

*until local*

*V(t)= vc*

*t=t+1*

*until t=MAX*

*end.*

Variabila *local* are rolul de a semnala eventuala apropiere de un punct de maxim local, determinat atunci când nu se mai fac modificări ale punctului curent (când nici un vecin al punctului curent nu furnizează o valoare îmbunătăţită a funcţiei obiectiv).

#### procedure simulated\_annealing

*begin*

*t=0*

*iniţializează temperatura sistemului T*

*selectează aleator un punct curent vc*

*evaluează vc*

*repeat*

*repeat*

*selectează vn în vecinătatea lui vc (prin modificarea unui bit din vc)*

*if f(vc)<f(vn) vc=vn*

*else*

*if random[0,1) <exp{(f(vn)-f(vc)/T} vc=vn*

*until (condiţie de terminare dat de T)*

*t=t+1*

*T=g(T,t)*

*until (criteriu de stop)*

*end.*

Funcţia *random[0,1)* returnează un număr aleator generat uniform în intervalul [0,1). Condiţia de terminare verifică dacă sistemul a atins “echilibrul termic”, adică dacă distribuţia de probabilitate corespunzătoare noilor şiruri selectate ating distribuţia Boltzman. La fiecare repetare a selecţiei temperatura sistemului descreşte (*g(T, t)<T,* pentru orice *t*). Algoritmul se încheie pentru o valoare mică a lui *T*, corespunzătoare stării de “îngheţ”, adică dacă nu se mai acceptă virtuale modificări asupra punctului curent.

**Observaţie**. Reprezentarea cromozomială a unui număr din domeniul poate fi realizată prin intermediul unui şir binar pe baza următorului procedeu. Fie *nz* numărul de zecimale al reprezentării (precizia dorită); domeniul *D* este divizat în *(b-a)10nz* intervale egale. Fie *m* cel mai mic număr natural cu proprietatea că . Reprezentarea oricărei variabile reale este un şir binar de *m* biţi, notat cu *s*, în care şi reprezintă valoarea zecimală a şirului binar, unde

şi [*t*] este partea întreagă a umărului *t*.

Dacă funcţia de optimizat depinde de mai multe variabile, fiecare dintre ele este reprezentată conform observaţiei de mai sus, un cromozom fiind constituit din concatenarea fiecărei reprezentări binare corespunzătoare unei variabile.

**Exemple**. Fie . Lungimea intervalului este 15,1. Dacă precizia dorită este de 4 zecimale, atunci domeniul considerat trebuie divizat în 15,1104=151000 intervale, deci *m=18.*

1. Fie . Atunci

Rezultă

şi

Într-adevăr,

1. Fie . Atunci

Obţinem

şi

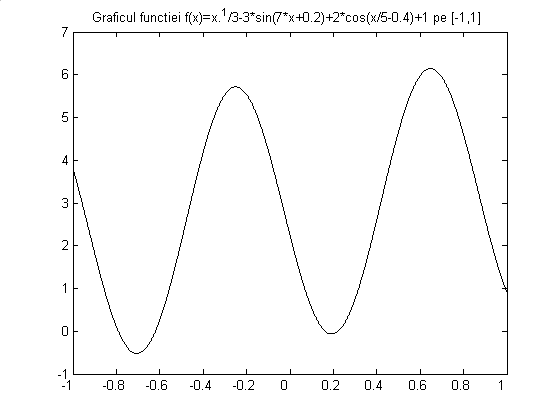
Într-adevăr,

**Exemple de aplicare a algoritmului hillclimbing. Rezolvarea unei probleme de optimizare a unei funcţii de o variabilă**

1. Fie , definită prin

Problema este de a calcula valoarea maximă a funcţiei pe intervalul .

Graficul funcţiei este prezentat în figura următoare.



Implementarea algoritmului “hill climbing” poate fi realizată astfel.

% functia obiectiv

function [val]=f\_obiectiv(x);

val=x.^1/3-3\*sin(7\*x+0.2)+2\*cos(x/5-0.4)+1;

end

% reprezentarea cromozomiala y, pe m biti, a unui numar x din [a,b] cu

% nz zecimale exacte

function [y,m]=repr\_sir\_bin(x,a,b,nz);

nr=(b-a)\*(10^nz);

m=fix(log2(nr))+1;

z=fix((x-a)\*(2^m-1)/(b-a));

y=bitget(z,m:-1:1);

end

% obtinerea fenotipului corespunzator

function [t]=repr\_reale(y,m,a,b);

x=0;

for i=1:m

x=bitset(x,m-i+1,y(i));

end;

t=a+x\*(b-a)/(2^m-1);

end

% implementarea algoritmului

function [val,v]=hillclimbing(a,b,nz,MAX);

% aici a=-1 si b=1, pentru a rezolva problema data

[y,m]=repr\_sir\_bin(0,a,b,nz);

V=[];

%m este numarul de biti pe care este reprezetat un numar din [a,b] cu

%precizia de nz zecimale

for t=1:MAX

local=0;

vc=unifrnd(a,b);

valm=f\_obiectiv(vc);

if (t==1)

val=valm;

v=vc;

end;

while(local==0)

%calculul vecinilor, insotit de valorile functiei obiectiv

[y,m]=repr\_sir\_bin(vc,a,b,nz);

valc=f\_obiectiv(vc);

ny=zeros(m,m+1);

for i=1:m

ny(i,1:m)=y(1:m);

ny(i,i)=not(y(i));

vn=repr\_reale(ny(i,1:m),m,a,b);

ny(i,m+1)=f\_obiectiv(vn);

end;

nys=sortrows(ny,m+1);

if(nys(m,m+1)>valc)

vc=repr\_reale(nys(m,1:m),m,a,b);

valm=nys(m,m+1);

else

local=1;

end;

end;

if(valm>val)

val=valm;

v=vc;

timp=t;

end;

V=[V vc];

end;

disp(v);

disp(val);

plot\_obiectiv(V,timp,a,b);

end

function []=plot\_obiectiv(V,timp,a,b);

figure

x=a:0.001:b;

plot(x,x.^1/3-3\*sin(7\*x+0.2)+2\*cos(x/5-0.4)+1,'k-');

hold on

[xx dim]=size(V);

disp(dim);

for i=1:dim

x=V(i);

y=f\_obiectiv(V(i));

plot(x,y,'rs');

hold on

end;

x=V(timp);

y=f\_obiectiv(V(timp));

plot(x,y,'bs');

end

La un apel

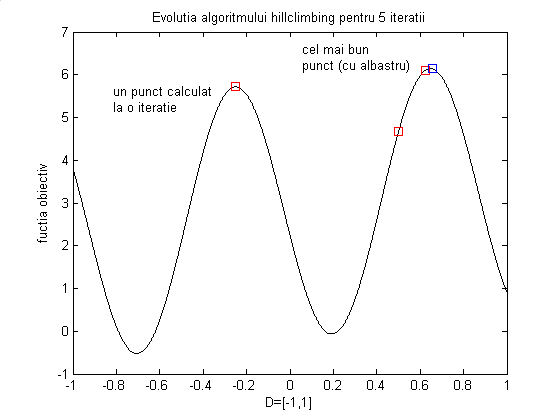
hillclimbing(-1,1,4,5);

pot fi obţinute rezultatul:

punctul în care este atins maximul 0.6476

maximul calculat 6.1425

şi evoluţia



La un apel

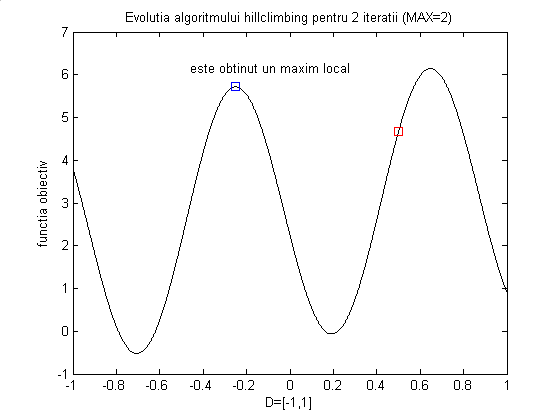
hillclimbing(-1,1,4,2);

pot fi obţinute rezultatul:

punctul în care este atins maximul -0.2495

maximul calculat 5.7169

şi evoluţia



La un alt apel

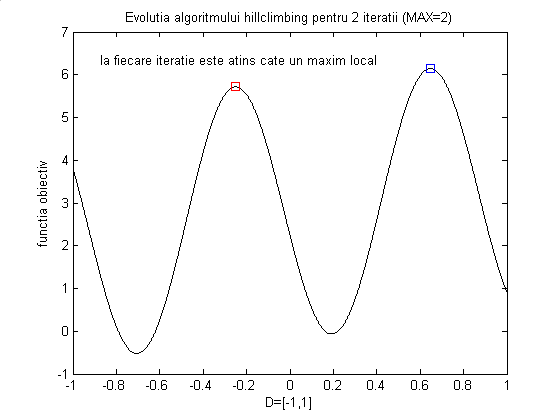
hillclimbing(-1,1,4,2);

pot fi obţinute rezultatul:

punctul în care este atins maximul 0.6476

maximul calculat 6.1425

şi evoluţia



1. Optimizarea unei funcţii de mai multe variabile.

Fie , definită prin

Problema este de a calcula valoarea maximă a funcţiei pe .

Implementarea algoritmului hill climbing în acest caz poate fi realizat astfel.

function [val,v]=hillclimbing(a,b,nz,MAX);

[y,m]=repr\_sir\_bin(0,a,b,nz);

V=[];

%m este numarul de biti pe care este reprezetat un numar din [a,b]x[a,b]

%cu precizia de nz zecimale

for t=1:MAX

local=0;

vc=unifrnd(a,b,1,2);

valm=f\_obiectiv(vc(1),vc(2));

if (t==1)

val=valm;

v=vc;

end;

while(local==0)

%calculul vecinilor, insotit de valorile functiei obiectiv

for i=1:2

[y((i-1)\*m+1:i\*m),m]=repr\_sir\_bin(vc(i),a,b,nz);

end;

valc=f\_obiectiv(vc(1),vc(2));

ny=zeros(2\*m,2\*m+1);

for i=1:2\*m

ny(i,1:2\*m)=y(1:2\*m);

ny(i,i)=not(y(i));

vn(1)=repr\_reale(ny(i,1:m),m,a,b);

vn(2)=repr\_reale(ny(i,m+1:2\*m),m,a,b);

ny(i,2\*m+1)=f\_obiectiv(vn(1),vn(2));

end;

nys=sortrows(ny,2\*m+1);

if(nys(2\*m,2\*m+1)>valc)

vc(1)=repr\_reale(nys(2\*m,1:m),m,a,b);

vc(2)=repr\_reale(nys(2\*m,m+1:2\*m),m,a,b);

valm=nys(2\*m,2\*m+1);

else

local=1;

end;

end;

if(valm>val)

val=valm;

v=vc;

timp=t;

end;

V=[V;vc];

end;

disp(v);

disp(val);

disp(timp);

plot\_obiectiv(V,timp,a,b);

end

function [val]=f\_obiectiv(x,y);

val=exp(-x^2-y^2)+y\*cos(5\*x)-x\*sin(3\*y);

end

function []=plot\_obiectiv(V,timp,a,b);

figure

[X,Y] = meshgrid([a:0.01:b]);

Z = exp(-X.^2-Y.^2)+Y.\*cos(5\*X)-X.\*sin(3\*Y);

plot3(X,Y,Z,'y');

grid on

hold on

[dim xx]=size(V);

disp(dim);

for i=1:dim

x=V(i,1);

y=V(i,2);

z=f\_obiectiv(x,y);

if(i==timp)

plot3(x,y,z,'ks');

hold on

else

plot3(x,y,z,'g.');

hold on

end;

end;

end

Funcţiile de reprezentare repr\_reale şi repr\_sir\_bin sunt similare primului exemplu (asigură transformarea număr real din şir binar).

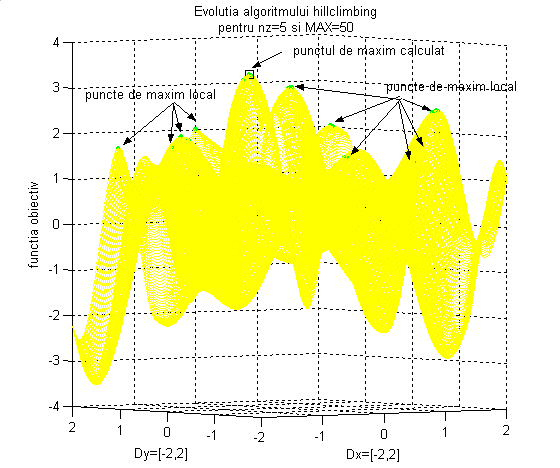
La un apel hillclimbing(-2,2,5,75);

pot fi obţinute rezultatele:

x=-1.9102, y=-1.6250

valoarea maximă: 3.4989.

În următoare figură este prezentat un exemplu de evoluţie posibilă a algoritmului hillclimbing aplicat pentru 75 de puncte de start.



* 1. **Exemple de aplicare a EA**

**Rezolvarea unei probleme de optimizare a unei funcţii de o variabilă**

Fie , definită prin

Problema este de a calcula valoarea maximă a funcţiei pe intervalul .

Spaţiul genotipurilor poate fi considerat mulţimea numerelor reale din intervalul . La fiecare moment de timp populaţia este constituită din *dim* numere reale din ; la momentul iniţial, populaţia conţine *dim* numere reale generate aleator pe intervalul .

Operatorul de mutaţie aplicat unui cromozom *x* determină obţinerea valorii *–x*. (în acest caz este posibil, deoarece, dacă ). Operatorul de recombinare aplicat pentru părinţii determină obţinerea cromozomului . Selecţia părinţilor este realizată astfel: sunt selectaţi jumătate din membrii populaţiei curente pe baza procedurii de tip turnir. Operaţia de încrucişare este aplicată pentru o împerechere aleatoare a câte 2 indivizi părinţi, cu probabilitatea *pc* (pentru fiecare pereche de părinţi selectată este generat un număr aleator în ; dacă acesta este inferior valorii *pc*, este efectuată încrucişarea). Progeniturile sunt supuse mutaţiei cu o probabilitate *pm*. Mecanismul de selectare a noii generaţii presupune ordonarea descrescătoare a multisetului format din indivizii populaţiei curente şi progeniturile obţinute prin operatorii de variaţie şi selecţie a părinţilor şi alegerea primilor *dim* indivizi pentru a forma populaţia următoare. Condiţia terminală este formulată astfel: a fost depăşit un prag al numărului de generaţii sau calitatea populaţiei, măsurată ca medie a funcţiei de evaluare în membrii populaţiei nu mai poate fi îmbunătăţită semnificativ.

În continuare sunt prezentate funcţiile MATLAB utilizate şi câteva exemple de aplicare a căutării evolutive descrise mai sus.

% generarea populaţiei iniţiale

function [pop]=genereza\_ini(dim);

pop=zeros(dim,2);

% fiecare membru al populatiei este un numar in [-1,1]

% la care este adaugata valoarea functiei obiectiv

for i=1:dim

pop(i,1)=unifrnd(-1,1);

pop(i,2)=f\_obiectiv(pop(i,1));

end;

end

%definirea functiei obiectiv

function [val]=f\_obiectiv(x);

val=x.^1/3-3\*sin(7\*x+0.2)+2\*cos(x/5-0.4)+1;

end

% operatorul mutatie

function [y]=mutatie(x);

y=x;

y(1)=-x(1);

y(2)=f\_obiectiv(y(1));

end

% operatorul de incrucisare

function [y]=crossover(x1,x2);

y=x1;

y(1)=(x1(1)+x2(1))/2;

y(2)=f\_obiectiv(y(1));

end

% mecanismul de selectie a parintilor

function [parinti]=selectie(pop);

[dim,xx]=size(pop);

d=round(dim/2);

% dim trebuie sa fie multiplu de 4, pentru ca d sa fie par

% va rezulta un singur copil din incrucisare

parinti=zeros(d,2);

for i=1:d

p1=unidrnd(dim);

p2=unidrnd(dim);

while(p2==p1)

p2=unidrnd(dim);

end;

if(pop(p1,2)>pop(p2,2))

parinti(i,:)=pop(p1,:);

else parinti(i,:)=pop(p2,:);

end;

end;

end

%obtinerea unei noi populatii popNou pe baza populatiei curente pop

%probabilitatea de incrucisare pc si probabilitatea de mutatie pm

% evalmed este valoarea medie a functiei obiectiv pentru noua generatie

function [popNou,evalmed]=trecere(pop,pc,pm);

[parinti]=selectie(pop);

[dim,xx]=size(pop);

[dd,yy]=size(parinti);

d=round(dd/2);

popN=zeros(d,2);

%aplica operatia de crossover

nr=0;

for i=1:d

j=2\*i;

prc=unifrnd(0,1);

if(prc<=pc)

nr=nr+1;

popN(nr,:)=crossover(parinti(j-1),parinti(j));

end;

end;

%aplica mutatia

for i=1:nr

prm=unifrnd(0,1);

if(prm<=pm)

popN(i,:)=mutatie(popN(i,:));

end;

end;

popNN=[pop;popN(1:nr,:)];

tt=sortrows(popNN,2);

popNou=zeros(dim,2);

popNou=tt(nr+1:dim+nr,:);

evalmed=mean(popNou(:,2));

end

function []=EA\_maxim(dim,pc,pm,Max,eps);

% dim este dimensiunea populatiei,multiplu de 4

% pc este probabilitatea de efectuare a

% unei incrucisari, pm este probabilitatea de efectuare a unei mutatii,

% Max este numarul maxim de iteratii(generatii)

%daca diferenta dintre valorile maxime ale

%functiei obiectiv de la o generatie la alta este in modul<eps --->stop

[pop]=genereaza\_ini(dim);

evalmed= mean(pop(:,2));

t=0;

V=[];

er=1;

plot\_obiectiv(pop);

while((t<Max)&&(er>eps))

[popN,evalmed1]=trecere(pop,pc,pm);

pop=popN;

er=abs(evalmed1-evalmed);

%disp(evalmed1);

%disp(er);

evalmed=evalmed1;

V=[V evalmed];

t=t+1;

if (t==20)

plot\_obiectiv(pop);

end;

end;

plot\_obiectiv(pop);

figure

i=1:t;

plot(i,V(i),'r-');

disp(t);

disp(popN(dim,:));

end

function []=plot\_obiectiv(pop);

figure

x=-1:0.001:1;

plot(x,x.^1/3-3\*sin(7\*x+0.2)+2\*cos(x/5-0.4)+1,'k-');

hold on

[dim,xx]=size(pop);

for i=1:dim

x=pop(i,1);

y=pop(i,2);

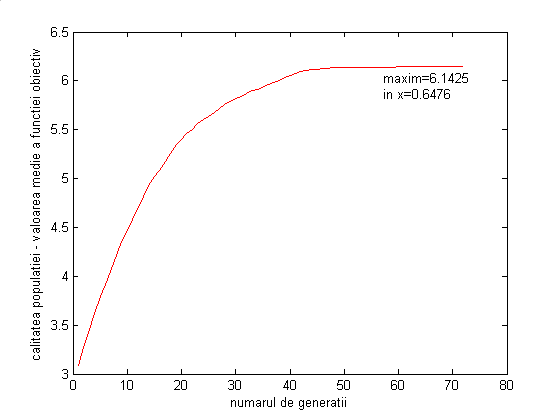
plot(x,y,'rs');

hold on

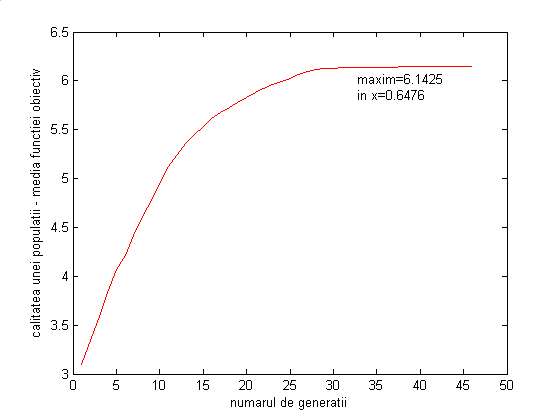
end;

end

Pentru pot fi obţinute următoarele rezultate.



Pentru sunt obţinute următoarele rezultate.

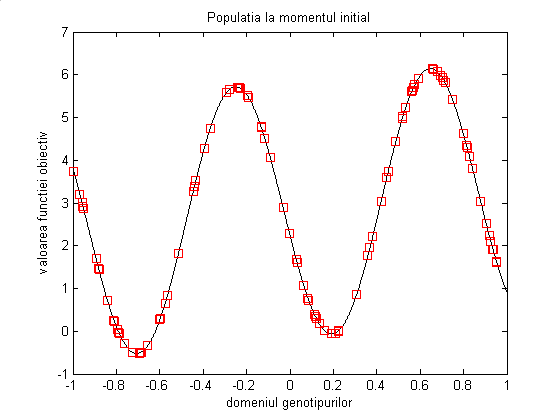


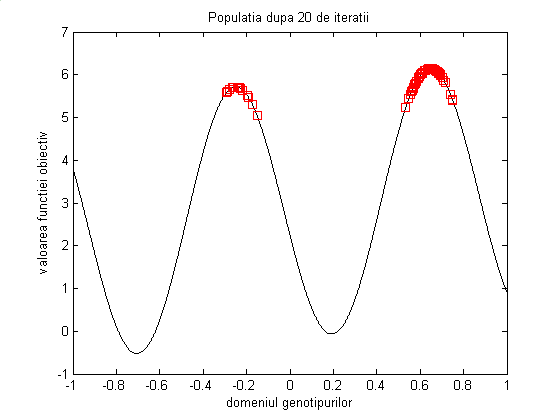
Valoarea maximă a funcţiei obiectiv este atinsă pentru şi este . Valorile obţinute sunt conforme cu graficul funcţiei prezentat în prima figură a acestei secţiuni.

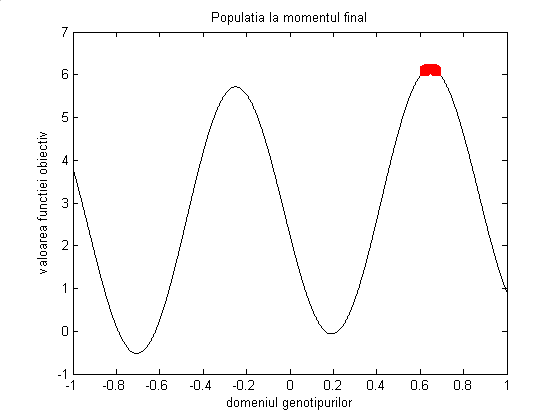
În următoarele trei figuri este reprezentată o posibilă distribuţie a membrilor populaţiei la momentul curent, după 20 de iteraţii şi la momentul final. Apelul este

*EA\_maxim(104,0.5,0.1,1000,0.0001);*

Iniţial, indivizii sunt distribuiţi aleator, pe între domeniul de definiţie al funcţiei obiectiv. După 20 de iteraţii, indivizii din populaţie sunt distribuiţi în vecinătatea unui punct de maxim local şi respectiv în vecinătatea punctului de maxim global. La terminarea căutării, EA reuşeşte să identifice maximul global al funcţiei obiectiv şi membrii populaţiei la momentul final sunt grupaţi într-o vecinătate a punctului de maxim global.







**Problema celor opt regine**

Problema este de a determina o configuraţie în care 8 regine pot fi plasate pe o tablă de şah astfel încât să nu se atace reciproc (nici o pereche de regine nu trebuie plasată pe aceeaşi linie, coloană sau diagonală). Aceasta poate fi generalizată imediat la cazul a *N* regine pentru o tablă de şah *N* x *N* -dimensională.

Deoarece pe fiecare linie şi coloană a tablei de şah trebuie plasată o singură regină, configuraţiile sunt reprezentate succesiv printr-un vector cu *N* componente. Valoarea componentei *i* dintr-un vector configuraţie reprezintă indicele coloanei în care este plasată regina aflată pe linia *i*. Din această reprezentare rezultă că două regine nu pot fi plasate pe aceeaşi linie a tablei de şah. De asemena, dacă vom presupune că elementele vectorului configuraţie sunt distincte (deci voctorul configuraţie corespunde unei permutări pe mulţimea ), rezultă că două regine nu pot fi plasate pe aceeaşi coloană a tablei de şah.

Rezolvarea clasică a unei astfel de probleme este prin căutare directă, utilizând metoda backtracking: iniţial este plasată o regină; după plasarea a *n* regine, este căutată o poziţie posibilă pentru plasarea celei de-a *n*+1-a regine (o poziţie din care noua regină să nu se afle în situaţie de atac cu nici una din celelalte *n* regine deja plasate pe tabla de şah). Dacă nu există nici o astfel de poziţie, cea de-a *n*-a regină este repoziţionată pe tabla de şah, dacă acest lucru este posibil.

O abordare evolutivă a acestei probleme este total diferită de căutarea directă şi este de tip non-incremental. Soluţiile candidat sunt complete, în sensul că fiecare vector configuraţie este instanţiat cu o permutare. Din punct de vedere al reprezentării, un fenotip, care reprezintă o configuraţie a tablei de şah astfel încât două regine să nu fie plasate pe aceeaşi linie sau aceeaşi coloană (este soluţie potenţială), este codificat prin intermediul vectorului configuraţie, permutare a mulţimii . Spaţiul genotipurilor este deci format din mulţimea permutărilor setului . Evident, nu toate elementele acestui spaţiu sunt soluţii admisibile (două regine pot fi plasate astfel încât să fie pe o aceeaşi diagonală a tablei de şah, deci să se afle în poziţie de atac reciproc).

Calitatea fiecărui genotip *x*, , poate fi evaluată în termenii numărului de perechi de regine care se află în poziţie de atac (numărul de perechi cu proprietatea că ): dacă acesta este 0 atunci genotipul este soluţie, în caz contrar valoarea calculată este strict pozitivă. Problema de optim care trebuie rezolvată este deci de minimizare funcţie *f* sau, echivalent, maximizarea funcţiei , unde este numărul maxim de perechi de regine care se pot afla în poziţie de atac (de exemplu în cazul ). În implementarea algoritmului, fiecărui genotip îi este asociată valoarea funcţiei de evaluare, deci datele cu care se lucrează sunt vectori *N*+1-dimensionali: primele *N* componente reprezintă configuraţia şi ultima componentă valoarea funcţiei de evaluare corespunzătoare acesteia.

La fiecare moment de timp, populaţia este constituită dintr-un multiset cu *dim* elemente (de exemplu, pentru , pentru ). Pentru această problemă, la fiecare generaţie sunt înlocuiţi câte 2 indivizi (cei mai slabi din punctul de vedere al funcţiei de evaluare) cu copii a doi părinţi. Cei doi părinţi sunt selectaţi aleator din populaţia curentă, . Operatorul de încrucişare este proiectat astfel încât progeniturile să păstreze proprietatea de a fi permutări: pentru doi indivizi şi pentru o poziţie sunt obţinute progeniturile astfel:

* copiază primele *poz* elemente din , respectiv în , respectiv
* copiază în ultimele *dim-poz+1* poziţii din , respectiv , elementele din , respectiv , începând cu primul şi până când au fost completate poziţiile din , respectiv , astfel încât , respectiv să nu conţină dubluri

Operatorul mutaţie este aplicat cu probabilitatea de 0.8 uneia din cele două progenituri rezultate după aplicarea operatorului de recombinare şi revine la selectarea aleatoare a două poziţii diferite din genotipul modificat şi interschimbarea valorilor din acele poziţii.

Condiţia terminală este formulată astfel: a fost atinsă o configuraţie soluţie sau a fost depăşit pragul dat pentru numărul maxim de generaţii.

În continuare sunt prezentate funcţiile MATLAB utilizate şi câteva exemple de aplicare a căutării evolutive descrise mai sus.

function [pop]=genereaza\_ini(N,dim);

pop=zeros(dim,N+1);

% fiecare membru al populatiei este o permutare pe 1,2,..,N

% la care este adaugata valoarea functiei obiectiv N-numarul de perechi

% de regine care se ataca reciproc

for i=1:dim

x=gen\_perm(N);

pop(i,1:N)=x;

pop(i,N+1)=valoare(x);

end;

end

% generarea aleatoare a unei permutari pe {1,2,…,N}

function [y]=gen\_perm(N);

y=zeros(1,N);

for i=1:N

gata=0;

while(~gata)

v=unidrnd(N);

if(~ismember(v,y))

y(i)=v;

gata=1;

end;

end;

end;

disp(y);

end

% functia de evaluare calculata pentru un genotip

function [val]=valoare(x);

[m,N]=size(x);

v=0;

for i=1:N-1

for j=i+1:N

if(abs(i-j)==abs(x(i)-x(j)))

v=v+1;

end;

end;

end;

val=N\*(N-1)/2-v;

end

% recombinarea

function [y1,y2]=crossover(x1,x2,poz);

[m,p]=size(x1);

N=p-1;

y1=x1;y2=x2;

y1=copiaza\_rest(y1,x2,poz,N);

y2=copiaza\_rest(y2,x1,poz,N);

y1(N+1)=valoare(y1(1:N));

y2(N+1)=valoare(y2(1:N));

%disp(y1);

%disp(y2);

end

function [y]=copiaza\_rest(y,x,poz,N)

i=poz+1;

while(i<=N)

for j=1:N

if(~ismember(x(j),y(1:i-1)))

y(i)=x(j);

i=i+1;

break;

end;

end;

end;

end

% mutatia

function [y]=mutatie(x,poz1,poz2);

[m,p]=size(x);

N=p-1;

y=x;

y(poz1)=x(poz2);

y(poz2)=x(poz1);

y(N+1)=valoare(y(1:N));

end

% selectarea parintilor – cate doi/popuatie

function [parinti]=selectie(pop);

[dim,p]=size(pop);

N=p-1;

parinti=zeros(2,N+1);

p1=unidrnd(dim);

p2=unidrnd(dim);

while(p2==p1)

p2=unidrnd(dim);

end;

parinti(1,:)=pop(p1,:);

parinti(2,:)=pop(p2,:);

end

% trecerea de generatia curenta la generatia urmatoare

function [popNou,evalmax]=trecere(pop,pm);

[parinti]=selectie(pop);

[dim,m]=size(pop);

N=m-1;

poz=unidrnd(N-1);

%aplica operatia de crossover

y=zeros(2,N+1);

[y(1,:),y2(2,:)]=crossover(parinti(1,:),parinti(2,:),poz);

%aplica mutatia

prm=unifrnd(0,1);

if(prm<=pm)

i=unidrnd(2);

poz1=unidrnd(N);

poz2=unidrnd(N);

while(poz2==poz1)

poz2=unidrnd(N);

end;

y(i,:)=mutatie(y(i,:),poz1,poz2);

end;

popN=[pop;y(1,:);y2(2,:)];

tt=sortrows(popN,N+1);

%disp(tt);

popNou=zeros(dim,N+1);

popNou=tt(3:dim+2,:);

evalmax=popNou(dim,N+1);

%disp(popNou);

end

% algoritmul evolutiv

function []=EA\_Queens(N,dim,pm,Max);

% N este umarul de regine, dim este dimensiunea populatiei, pm este

% probabilitatea de efectuare a unei mutatii si Max este numarul maxim

%de iteratii(generatii)

[pop]=genereaza\_ini(N,dim);

evalmax=0;

t=0;

V=[];

while(evalmax<N\*(N-1)/2 && t<Max)

[popN,evalmax]=trecere(pop,pm);

pop=popN;

disp(popN(dim,:));

V=[V popN(dim,N+1)];

t=t+1;

end;

figure

i=1:t;

plot(i,V(i),'ro:');

axis([0 t V(1)-1 (N\*(N-1)/2)+5]);

figure

for i=0:N

plot( [0 N],[i i],'k-');

plot( [i i],[0 N],'k-');

hold on

end;

hold on

for i=1:N

x=i;

y=pop(dim,i);

k=x-1:0.1:x;

l=y-1:0.1:y;

plot(k,l,'bv');

hold on

k=x:-0.1:x-1;

l=y-1:0.1:y;

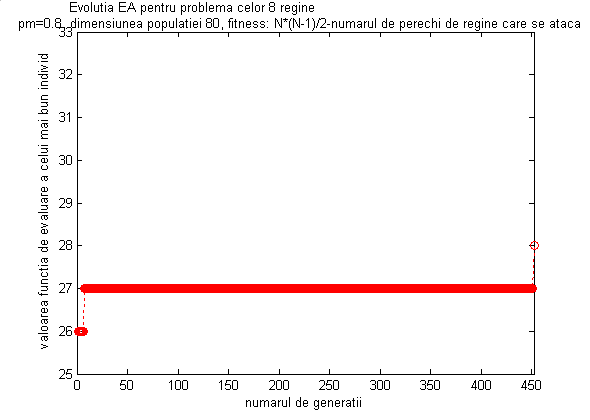
plot(k,l,'bv');

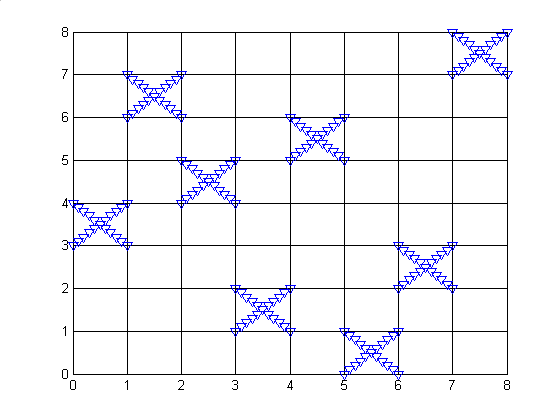
hold on

end;

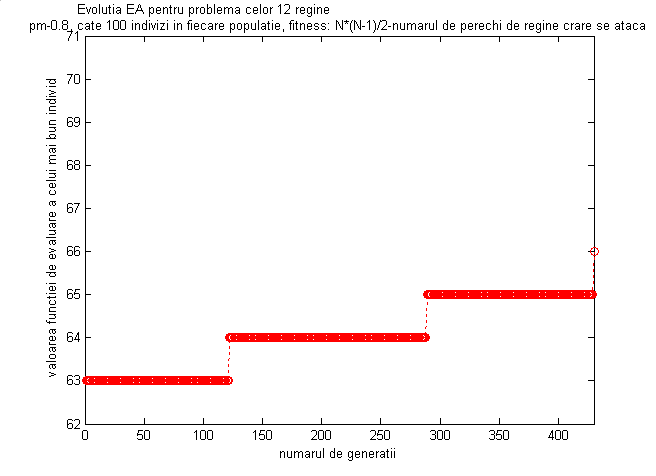
axis([0 N 0 N]);

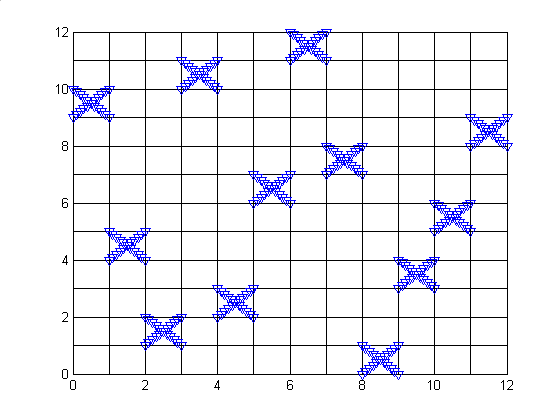
end





O configuraţie corectă obţinută prin aplicarea algoritmului de mai sus





O configuraţie corectă obţinută prin aplicarea algoritmului de mai sus

**Problema rucsacului de tip 0-1**

Problema rucsacului de tip 0-1 este enunţată în cele ce urmează. Fiind date *m* obiecte, fiecare având asociate o valoare şi respectiv un cost de selecţie, trebuie determinat un set de obiecte cu proprietatea că este de valoare maximă (unde valoarea unei mulţimi de obiecte este definită ca suma valorilor obiectelor ce o compun) şi costul (definit ca suma costurilor obiectelor setului) este sub un prag dat, *Cmax*. Problema este de tip 0-1 pentru că nu pot fi selectate fracţiuni dintr-un obiect, ci întregul obiect (dacă prin această operaţie nu este depăşit *Cmax*).

O soluţie posibilă este dată de un set de obiecte selectate astfel încât costul lui este inferior valorii *Cmax*. Dacă *val* este vectorul valorilor asociate celor *m* obiecte şi *cost* este vectorul costurilor, reprezentarea unui candidat la soluţie poate fi realizată printr-un vector de *m* elemente, *v*, unde

şi

Funcţia de evaluare asociată lui *v* calculează valoarea asociată selecţiei reprezentate de *v*,

Este evident că fiecare genotip *v* corespunde unui număr şi *v* este reprezentarea binară a lui *R-1* şi cu proprietatea că nu este depăşit pragul *Cmax*.

La fiecare moment de timp, populaţia este formată din *N* indivizi reprezentaţi aşa cum a fost explicat mai sus. La momentul iniţial, este generat aleator un număr , genotipul corespunzător lui fiind reprezentarea binară a lui *R-1*; *R* este selectat pentru reprezentare numai dacă nu este depăşit pragul *Cmax*. Procedeul continuă până la generarea a *N* cromozomi.

Selecţia părinţilor este realizată prin următorul procedeu. De *N* ori sunt alese perechi de cromozomi din populaţia curentă şi este selectat cel mai bun dintre ei, din punctul de vedere al funcţiei de evaluare. Recombinarea este realizată cu o probabilitate *pc* () şi este proiectată astfel încât progeniturile să rămână admisibile (costul asociat să fie inferior lui *Cmax*): pentru doi indivizi şi pentru o poziţie generată aleator sunt obţinute progeniturile astfel:

* copiază primele *poz* elemente din , respectiv în , respectiv
* copiază în ultimele *m-poz+1* poziţii din , respectiv , ultimele *m-poz+1* elementele din , respectiv

Dacă astfel obţinuţi nu îndeplinesc proprietatea de admisibilitate, este selectată o altă pereche de indivizi şi este aplicat acelaşi mecanism, până când este obţinută o pereche de genotipuri (soluţii admisibile ale problemei de rezolvat). Dacă o pereche de părinţi nu este selectată pentru încrucişare, aceasta este menţinută în populaţia următoare.

Operatorul mutaţie este aplicat cu o probabilitate pm (în general ) unui individ *v* şi revine la selectarea aleatoare a unei poziţiei şi modificarea valorii şi efectuarea operaţiei (valoarea genei poz este schimbată din 0 în 1 sau invers). Dacă rezultatul nu este admisibil, este selectat un alt individ pentru mutaţie.

Mecanismul de înlocuire a populaţiei revine la selectarea acelor indivizi aleşi ca părinţi dar pentru care nu s-a realizat operaţia de încrucişare (valoarea generată aleator este inferioară valorii *pc*) şi pentru progeniturile (eventual mutante) rezultate. Fiecare generaţie are deci dimensiunea constantă, *N*.

Condiţia terminală controlează numărul de iteraţii efectuat. Alternativ, poate fi implementată şi o variantă în care să fie controlată şi calitatea populaţiei, evaluată ca medie a funcţiei de evaluare sau ca valoarea maximă a funcţiei de evaluare calculată pentru fiecare din indivizii populaţiei curente.

În continuare sunt prezentate funcţiile MATLAB pentru implementarea căutării evolutive. Datele sunt preluate din fişierul nume şi sunt reprezentate de costurile şi valorile asociate celor *m* obiecte. Testele au fost efectuate pentru *m*=16 şi *m*=32.

% citirea datelor din fisier

function [val,cost,costM]=citeste\_date(nume);

f=fopen(nume);

m=fscanf(f,'%d',1);

val=fscanf(f,'%lg',[m,1]);

cost=fscanf(f,'%lg',[m,1]);

costM=fscanf(f,'%lg',1);

fclose(f);

end

% generarea populaţiei iniţiale

function [pop,val,cost,costM]=genereza\_ini(nume,N);

[val,cost,costM]=citeste\_date(nume);

[m,p]=size(val);

pop=zeros(N,m);

i=0;

while(i<N)

%se lucreaza pe m biti, generez R, un numar intre 1 si 2^m

%si consideram reprezentarea binara

R=unidrnd(2^m);

x=repr\_bin(R-1,m);

[ok,sum]=verifica(x,cost,costM);

if(ok)

i=i+1;

pop(i,:)=x;

end;

end;

end

% obtinerea reprezentarii binare a unui numar

function [y]=repr\_bin(x,m);

y=bitget(x,m:-1:1);

end

% verificarea admisibilitatii unei solutii potentiale

function [OK,sum]=verifica(x,cost,costM);

sum=x\*cost;

OK = sum<=costM;

End

% operatorul de mutatie

function [y]=mutatie(x,poz);

y=x;

y(poz)=not(x(poz));

end

% operatorul de incrucisare

function [y1,y2]=crossover(x1,x2,poz);

[m,p]=size(x1);

y1=x1;y2=x2;

y1(1:poz)=x1(1:poz);

y1(poz+1:m)=x2(poz+1:m);

y2(1:poz)=x2(1:poz);

y2(poz+1:m)=x1(poz+1:m);

end

% selectia parintilor

function [parinti]=selectie(pop,val,cost,costM);

[N,m]=size(pop);

parinti=zeros(N,m);

i=0;

for i=1:N

p1=unidrnd(N);

p2=unidrnd(N);

%functia de fitness - suma vi\*gi, gi=1,

%daca a fost ales obiectul i, 0 altfel

if(pop(p1,:)\*val>=pop(p2,:)\*val)

parinti(i,:)=pop(p1,:);

else

parinti(i,:)=pop(p2,:);

end;

end;

end

% inlocuirea populatiei curente cu generatia urmatoare

function [popN]=trecere(pop,val,cost,costM,pc,pm);

[parinti]=selectie(pop,val,cost,costM);

[N,m]=size(pop);

popN=zeros(N,m);

i=0;

%aplica operatia de crossover

for i=1:2:N

p1=unidrnd(N);x1=parinti(p1,:);

p2=unidrnd(N);x2=parinti(p2,:);

prc=unifrnd(0,1);

if(prc<=pc)

cont=1;

%trebuie ca progentiturile sa fie solutii admisibile

while(cont)

poz=unidrnd(m);

%disp(poz);

[y1,y2]=crossover(x1,x2,poz);

[OK1,sum1]=verifica(y1,cost,costM);

[OK2,sum2]=verifica(y2,cost,costM);

if((OK1==1)&&(OK2==1))

popN(i,:)=y1;

popN(i+1,:)=y2;

cont=0;

end;

end;

else

popN(i,:)=x1;

popN(i+1,:)=x2;

end;

end;

%aplica mutatia

for i=1:N

prm=unifrnd(0,1);

if(prm<=pm)

cont=1;

%trebuie ca progentitura sa fie solutie admisibila

while(cont)

poz=unidrnd(m);

x=popN(i,:);

[y]=mutatie(x,poz);

[OK,sum]=verifica(y,cost,costM);

if(OK==1)

popN(i,:)=y;

cont=0;

end;

end;

end;

end;

end

% calculul valorii maxime a functiei de evaluare

function [Q,x,V]=maximQ(pop,val);

[N,m]=size(pop);

Q=0;

V=zeros(N);

for i=1:N

qm=pop(i,:)\*val;

V(i)=qm;

if(qm>=Q)

Q=qm;

x=pop(i,:);

end;

end;

end

% implementarea cautarii evolutive

function []=GA\_Knapsack(nume,N,pc,pm);

%GA\_Knapsack('date4.txt',2000,0.9,1.0/8); - obtinerea solutiei exacte

%pentru 32 de componente

%pentru 16 componente GA\_Knapsack('date.txt',500,0.8,1.0/8); -solutia

%exacta

[pop,val,cost,costM]=genereaza\_ini(nume,N);

[Q,x,V]=maximQ(pop,val);

%disp(x);

%disp(Q);

figure;

i=1:N;

plot(i,V(i),'r-');

hold on;

for j=1:15

[popN]=trecere(pop,val,cost,costM,pc,pm);

pop=popN;

[Q,x,V]=maximQ(pop,val);

%disp(Q);

%disp(x);

end;

[Q,x,V]=maximQ(pop,val);

%disp(x);

i=1:N;

plot(i,V(i),'k-');

hold on

for j=1:50

[popN]=trecere(pop,val,cost,costM,pc,pm);

pop=popN;

[Q,x,V]=maximQ(pop,val);

%disp(Q);

%disp(x);

end;

[Q,x,V]=maximQ(pop,val);

disp('Solutia optima calculata');

disp(x);

disp('Valoarea optima calculata');

disp(Q);

i=1:N;

plot(i,V(i),'b-');

end

La apelul

GA\_Knapsack('date.txt',500,0.8,1.0/8);

fişierul *date.txt* conţine următoarele informaţii:

Numărul de componente:

16

Valoarea obţinută la alegerea fiecărei componente

4.5 6 8 5.4 10.2 3.2 4.2 8.3 3.2 4.5 9 10.9 5 6.2 7 8.2

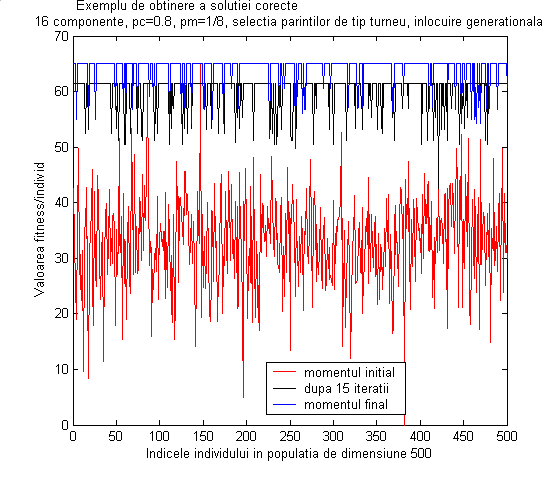
Costul alegerii fiecărei componente

3 2 2.4 6.8 5 6.2 5 4.8 5 0.2 3 4.8 7 7.1 9.7 4

Costul maxim permis

26.3

În continuare este prezentat un exemple de evoluţie a EA.



Solutia optima calculata

0 1 1 0 1 0 0 1 0 1 1 1 0 0 0 1

Valoarea optima calculata

65.1000

La apelul

GA\_Knapsack('date4.txt',2000,0.9,1.0/8);

fişierul *date4.txt* conţine următoarele informaţii:

Numărul de componente:

32

Valoarea obţinută la alegerea fiecărei componente

2.5 6 8 3.4 10.2 3.2 4.2 8.3 3.2 4.5 9 10.9 5 6.2 7 8.2 9.2 1.3 4 5 8.9 0.4 7.3 4.1 8

3 11 2.5 3 1.2 5 2.3

Costul alegerii fiecărei componente

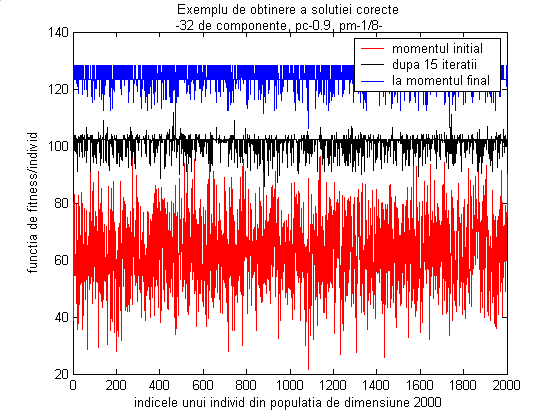
7 3.1 2.4 8.8 5 5.2 5 4.8 5 5.2 6 4.8 7 1.1 2.7 4.9 5 5.6 8 2.5 4 4 2.2 5.2 3.1 6.4

3.3 7 6.5 8 1.4 9.8

Costul maxim permis

56.6

În continuare este prezentat un exemple de evoluţie a EA.



Solutia optima calculata

0 1 1 0 1 0 0 1 0 0 1 1 0 1 1 1

1 0 0 1 1 0 1 0 1 0 1 0 0 0 1 0

Valoarea optima calculata

128.2000