Universitatea “Alexandru Ioan Cuza”
Facultatea de Informatică
Departamentul de Învățământ la Distanță

Prof. Dr. Henri Luchian
Prep. Drd. Mihaela Breabăn

ALGORITMI GENETICI

Anul III, Semestrul II

2006 - 2007

Introducere

Cursul de față prezintă o alternativă metodelor clasice de rezolvare a problemelor cu ajutorul calculatorului, introducând o serie de algoritmi a căror formulare a fost inspirată din natură şi care prezintă drept caracteristică centrală auto-adaptarea. Aceşti algoritmi au nevoie de mai puține aproximări în rezolvarea problemei deoarece lucrează în mod direct cu reprezentări ale soluțiilor în calculator eliminând necesitatea obiectelor matematice intermediare. Decizii sunt luate în mod probabilist şi mai multe soluții bune sunt returnate. Auto-adaptarea urmăreşte descoperirea şi exploatarea proprietăților specifice instanței problemei.

În acest context sunt prezentați algoritmii genetici - metode generale de căutare inspirate din evoluția naturală care îşi găsesc o largă utilizare în optimizare, învățare automată şi proiectarea sistemelor complexe.

O introducere în calculul evolutiv prezintă sursa de inspirație şi motivația utilizării unor astfel de algoritmi. Este prezentată apoi schema generală a algoritmilor genetici precum şi unele abateri de la aceasta. Sunt descrise detalii de implementare pentru rezolvarea de probleme din diferite domenii şi sunt expuse rezultate experimentale. Rezultate teoretice, precum teorema schemelor, explică modul în care algoritmii genetici efectuează căutarea şi ajung la convergență. Sunt evidențiate proprietăți ale problemelor pentru care algoritmii genetici ar trebui să depăşească prin performanță alți algoritmi; în acest context sunt prezentate funcțiile Royal Roads. Sunt trecute în revistă unele modificări ale algorimului genetic clasic: adaptarea reprezentării, scheme cu diferite presiuni de selecție, operatori genetici modificați.

Capitolul 1
Strategii în rezolvarea problemelor

Goldstein şi Levin (1987) au definit rezolvarea problemelor ca fiind procesul cognitiv de ordin înalt care necesită modulația şi controlul mai multor capacități/îndemânări fundamentale, considerând-o cea mai complexă dintre toate funcțiile intelectuale. Rezolvarea problemelor apare atunci când un organism sau un sistem ce posedă inteligență artificială nu cunoaşte drumul de la o stare dată, către o stare scop dorită.

Procesul rezolvării problemelor diferă în funcție de domeniul de cunoaştere şi de nivelul de expertiză. Rezultate obținute în laboratoare de multe ori nu pot fi extinse pentru situații reale, complexe. Din acest motiv, ultimele studii se concentrează pe probleme cât mai complexe iar utilizarea calculatoarelor a devenit o necesitate.

În rezolvarea de probleme cu ajutorul calculatorului sunt utilizate două strategii: **abordarea deterministă** (care cuprinde algoritmi determinişti exacți şi euristici) şi **abordarea probabilistă.**

Algoritmii determinişti produc aceeaşi soluție şi urmează aceeaşi secvență de stări pe o instanță dată la execuții repetate.

Algoritmii determinişti exacți obțin soluții exacte dar nu sunt tot timpul utilizabili în practică deoarece, de cele mai multe ori realizează o căutare exhaustivă în spațiul problemei. Considerând pentru rezolvare clasa algoritmilor determinişti, a fost stabilită ierarhia de complexitate a problemelor. Astfel, o problemă are complexitatea dată de cel mai bun algoritm determinist exact cunoscut a o rezolva. Clasa problemelor NP-complete cuprinde probleme pentru care nu se cunoaşte încă un algoritm determinist exact care să o rezolve în timp polinomial. Exemple de probleme NP-complete sunt: problema rucsacului, problema comisului voiajor, problema clicii maximale. Pentru aceste probleme algoritmii determinişti exacți devin intractabili.

Euristicile sunt algoritmi determinişti care obțin soluții aproximative. Cu cât precizia soluției este mai bună, cu atât complexitatea timp este mai mare. Utilizând această clasă de algoritmi, complexitatea problemelor se modifică: de exemplu, problema rucsacului devine polinomială pentru orice precizie iar problema comis-voiajorului devine polinomială pentru o precizie de 50%. Pentru problema clicii maximale nu se cunoaşte nici o euristică care

să dea soluții aproximative (oricare ar fi precizia) în timp polinomial.

Algoritmii probabilişti implică măcar un pas în care decizia se ia în mod aleatoriu. Aceştia sunt algoritmi impredictibili, la repetări succesive se obțin soluții diferite datorită unor paşi ce depind de valori generate aleatoriu în timpul execuției. Soluțiile obținute sunt aproximative.

Un algoritm probabilist poate fi privit ca o distribuție de probabilitate peste o mulțime de algoritmi determinişti. Dacă este uşor să se găsească o instanță a unei probleme care să ridice dificultăți pentru unul sau mai mulți algoritmi determinişti din această mulțime, este dificil să se găsească o singură intrare cu şanse mari de a bate un algoritm ales aleatoriu. Această paradigmă stă la baza succesului oricărui algoritm probabilist (R. Motwani, P. Raghavan).

Din clasa algoritmilor probabilişti fac parte algoritmi precum Monte Carlo, Las Vegas şi metaeuristicile (algoritmi evolutivi, călirea simulată, căutarea tabu, etc.). Complexitatea timp pentru astfel de algoritmi creşte odată cu precizia soluțiilor. Algoritmii genetici utilizează o schemă polinomială. Unele Strategii Evolutive au probabilitatea egală cu 1 să găsească soluția exactă.

Calitatea unei soluții aproximative se apreciază cu ajutorul a doi indici: *acuratețea* - care măsoară distanța față de optim a valorii acelei soluții, şi precizia – care măsoară apropierea variabilelor soluției de punctul în care se obține optimul.

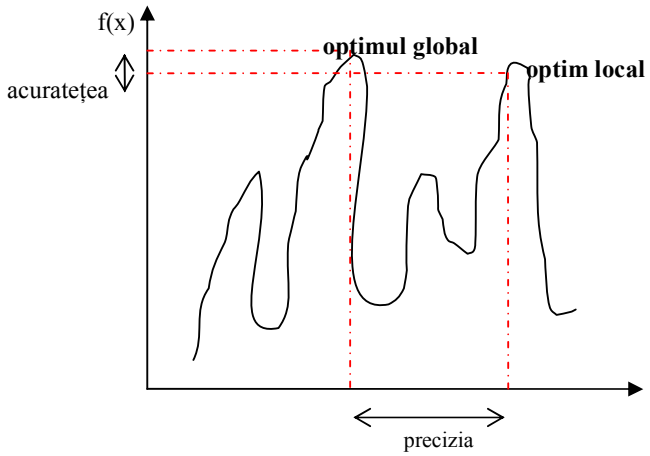

Fig1: Precizia şi acuratețea unei soluții aproximative: f-funcția de evaluare; x - parametrii funcției.

Metodele convenționale de rezolvare de probleme precum calculul diferențial, calculul integral, analiza numerică şa. sunt denumite tehnici ***hard computing***. Aceste modele sunt rigide, statice, fără posibilitate de adaptare; ele sunt complet formulate în prealabil şi nu se pot modifica cu nimic pe parcursul rezolvării. Doar sisteme relativ simple pot fi descrise şi analizate cu astfel de metode.

Existența sistemelor complexe din domenii precum biologia, medicina, economia a dus la apariția metodelor ***soft computing*** care prezintă posibilitate de *auto-adaptare*. Aceste metode sunt bazate pe feed-back-ul intern al algoritmului şi al structurilor de date iar proprietățile individuale ale instanței problemei sunt exploatate la maxim. Spre deosebire de metodele hard-computing care pun accent pe exactitate, tehnicile soft-computing exploatează adevărul parțial făcând uz de modelarea şi raționarea probabilistă.

Principalele modele soft-computing sunt rețelele neuronale artificiale, sistemele fuzzy (vagi) şi calculul evolutiv. Dacă primele două modele sunt dezvoltări moderne ale unor tehnici din statistică şi teoria mulțimilor, calculul evolutiv a adus o abordare şi idei fundamental noi în modelare, în conceptul de calcul şi în paradigma generală de rezolvare a problemelor de optimizare.

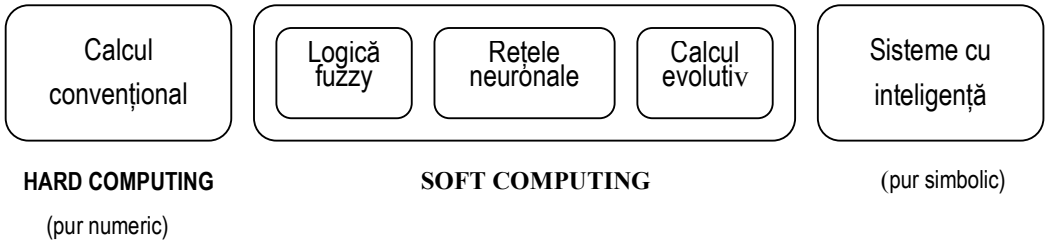

Fig. 2: Calculul soft este situat între calculul convențional (hard) şi sistemele de inteligență artificială

Capitolul 2
Algoritmi genetici

2.1 Calcul evolutiv

Calculul evolutiv este o subramură a Inteligenței Artificiale care cuprinde paradigme soft-computing pentru rezolvarea de probleme în domenii precum învățarea automată, proiectarea sistemelor complexe şi în optimizare. Aceşti algoritmi sunt algoritmi probabilişti inspirați cu precădere din biologie şi au în comun următoarele elemente:

- mențin o *populație* de reprezentări de *soluții candidat*
- care evoluează de-a lungul unor *generații*/iterații
- sub controlul unei *funcții fitness* care măsoară meritul individual.

Algoritmii evolutivi cu cele trei direcții - strategiile evolutive, programarea evolutivă şi algoritmii genetici - sunt cei mai vechi algoritmi de acest tip şi au ca sursă de inspirație evoluția biologică. Ulterior s-au dezvoltat metode care au la bază studiul comportamentului colectiv în sisteme decentralizate precum coloniile de furnici (Ant Colony Optimization) şi stolurile de păsări (Particle Swarm Optimization), metode care formează paradigma **Swarm intelligence**.

În calculul evolutiv paradigma de rezolvare a problemelor se modifică. Dacă în hard-computing paşii necesari rezolvării unei probleme sunt:

1. înțelege problema:
2. construieşte un model matematic şi studiază teoretic acel model;
3. modelează un algoritm de rezolvare a problemei pe baza rezultatelor teoretice;
4. implementează metoda şi execută programul;
5. obține soluția;

în calculul evolutiv această paradigmă devine:

1. înțelege problema;
2. construieşte reprezentarea, funcția fitness;
3. modelează un algoritm de calcul evolutiv pentru rezolvarea ei;
4. implementează metoda şi execută experimente;
5. obține mai multe soluții;

ceea ce înseamnă că rezolvarea unei probleme nu mai trebuie să fie precedată neapărat de obținerea unor rezultate teoretice asupra modelului matematic corespunzător.

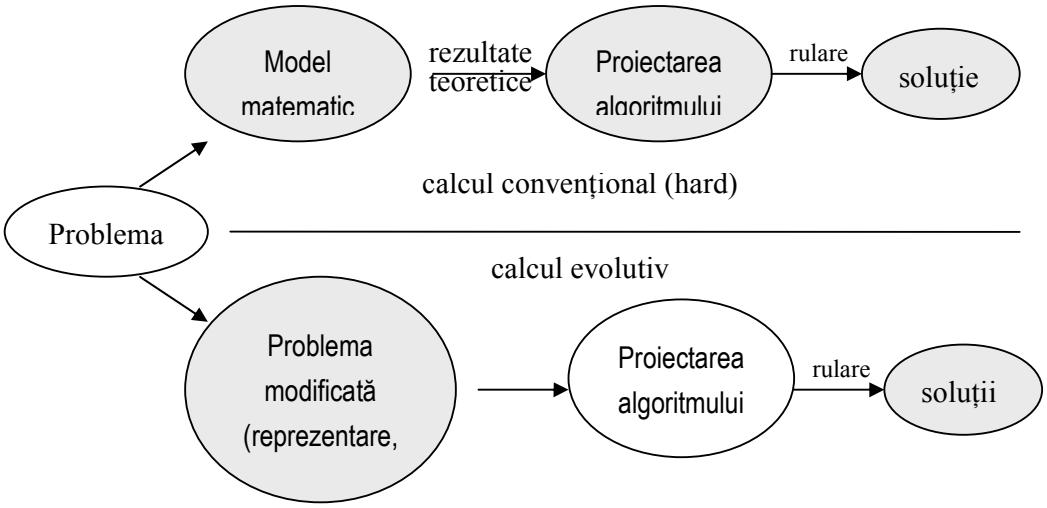

Fig. 3: Etapele rezolvării unei probleme în hard computing şi calcul evolutiv; etapele haşurate reprezintă aproximări de la problema reală

Tehnicile calculului evolutiv sunt metode slabe de optimizare; ele nu sunt, în principiu specializate, ci sunt aplicabile unor clase largi de probleme.

Pentru a putea fi abordată cu aceste tehnici, problema de rezolvat trebuie pusă sub forma unei probleme de optimizare.

2.2 Terminologie

Algoritmii evolutivi utilizează un vocabular împrumutat din genetică. Un algoritm genetic simulează evoluția printr-o succesiune de **generații** (proces iterativ) ale unei **populații** (mulțime) de soluții candidat. O soluție candidat este reprezentată intern ca un şir de **gene** (**genotip**) şi poartă numele de **cromozom**. Gena este informația atomică dintr-un cromozom. Poziția pe care o ocupă o genă se numeşte **locus** iar toate valorile posibile pentru o genă formează setul de **alele** ale genei. Populația menținută de algoritmul genetic evoluează prin aplicarea unor **operatori genetici** care simulează elemente considerate de geneticieni ca fiind fundamentale: **mutația** care constă în modificarea aleatoare a unei gene şi **încrucişarea** (**recombinarea**) care are ca scop schimbul de informație genetică între doi sau mai mulți cromozomi. Cromozomul asupra căruia se aplică un operator genetic poartă numele de părinte iar cromozomul rezultat în urma acelui operator este numit descendent. Un proces (aleator) de **selecție** oferă indivizilor mai bine adaptați şanse mai mari pentru a supraviețui în generația următoare. Gradul de adaptare la mediu al indivizilor este măsurat de **funcția fitness** obținută

pornind de la funcția matematică de optimizat. **Soluția** returnată de un algoritm genetic este cel mai bun individ din ultima generație.

2.3 Direcții în algoritmii evolutivi

Excelenta adaptare a ființelor vii în ceea ce priveşte structura, forma, funcțiile şi stilul de viață, a făcut ca mulți cercetători să împărtăşească ideea că natura posedă soluții optimale la problemele sale, soluții superioare oricăror performanțe tehnologice. S-a demonstrat chiar matematic optimalitatea unor subsisteme biologice precum: raportul diametrelor ramificațiilor arterelor, valoarea hematocritului, poziția punctului de ramificare a vaselor de sânge, etc. În consecință, au apărut încercări de imitare a procesului de evoluție naturală.

Strategiile evolutive au luat naştere în 1970 când, având de rezolvat o problemă de mecanica fluidelor, Hans-Paul Schwefel şi Ingo Rechenberg au căutat o nouă tehnică de optimizare întrucât metodele cunoscute până în acel moment nu duceau la soluție. Ideea lor a întruchipat conjectura lui Rechenberg, rămasă până astăzi justificarea fundamentală a aplicării tehnicilor evolutive: "*Evoluția naturală este, sau cuprinde, un proces de optimizare foarte eficient, care, prin simulare, poate duce la rezolvarea proceselor dificile de optimizare".* Metoda astfel concepută pentru a rezolva probleme de optimizare de variabilă continuă are la bază utilizarea operatorului de mutație aleatoare şi selecția celui mai bun individ.

Programarea evolutivă s-a cristalizat la Universitatea San Diego în 1966 când Fogel a generat programe simple sub formă de maşini cu număr finit de stări. Pe astfel de structuri (diagrame de stare-tranziție pentru maşini cu număr finit de stări) au fost aplicate mutații aleatoare iar cel mai bun individ este selectat pentru supraviețuire.

Algoritmii genetici au fost propuşi de John Holland în 1973 după mulți ani de studiere a ideii de simulare a evoluției. Aceşti algoritmi modelează moştenirea genetică şi lupta Darwiniană pentru supraviețuire.

Complexitatea problemelor din lumea reală a dus la dezvoltarea de noi direcții care implementează sau utilizează tehnicile specifice algoritmilor genetici în mod nestandard: **algoritmi genetici messy** (Goldeberg), **programarea genetică** (Koza), **co-evoluția** (Hillis), **algoritmii memetici/culturali**, algoritmii micro-economici. De asemenea s-au realizat diverse studii statistice în ceea ce priveşte compatibilitatea între reprezentare şi funcția fitness, operatorii, etc.

2.4 Elemente comune ale algoritmilor evolutivi

Auto-adaptarea în algoritmii evolutivi se bazează pe simularea evoluției prin adaptare şi competiție din lumea vie. Toți algoritmii evolutivi sunt procese iterative şi pot fi descrişi prin următoarea schemă generală: mențin o populație de reprezentări de soluții candidat care sunt îmbunătățite de-a lungul unor generații succesive prin intermediul unor operatori specifici (cum sunt mutația şi încrucişarea) şi a selecției bazate pe meritul individual care este asignat indivizilor prin evaluare cu ajutorul funcției fitness.

Deosebirile între direcțiile menționate în algoritmii evolutivi apar la nivel de reprezentare a soluțiilor, implementare a operatorilor şi a schemelor de selecție.

2.5 Structurile de date

Strategiile evolutive lucrează cu reprezentări în virgulă mobilă.

Programarea evolutivă utilizează ca reprezentări maşini cu număr finit de stări.

Algoritmii genetici înregistrează o mare variație în ceea ce priveşte reprezentarea soluțiilor candidat. Algoritmii genetici standard reprezintă cromozomii ca şiruri de biți, şiruri de numere întregi sau reale sau permutări, în funcție de problema propusă spre rezolvare. În ultimii ani s-au raportat diverse abateri de la reprezentarea sub formă de şir pentru a îngloba informație specifică problemei; sunt utilizate structuri bidimensionale precum matrici sau grafuri, reprezentări bazate pe vecinătăți (insule) sau distribuite geografic, etc. Se folosesc de asemenea reprezentări de dimensiune variabilă.

În programarea genetică se optimizează o populație de programe-calculator reprezentarea unui individ realizându-se cu o structură de tip arbore.

Condițiile pe care trebuie să le satisfacă o bună reprezentare sunt: independența relativă a genelor (lipsa epistaziei), includerea cât mai multor restricții ale problemei direct în reprezentare, complexitate scăzută a codificării/decodificării cromozomilor. În general, pentru modelarea unei probleme se poate alege între mai multe reprezentări candidat.

2.6 Modele teoretice pentru algoritmii genetici

Modelul teoretic fundamental propus pentru studiul algoritmilor genetici este cel propus de Holland şi denumit modelul schemelor. Acesta explică modul în care un algoritm genetic standard converge la soluție fără a face

însă predicții asupra producerii sau nu a convergenței, a acurateței soluțiilor găsite sau asupra vitezei de convergență. Rezultatul central este teorema schemelor însă ipotezele acesteia o fac aplicabilă doar pentru algoritmul genetic clasic.

Teorema schemelor a fost extinsă mai târziu (Vose 1991) la diferite tipuri de reprezentări şi diferiți operatori de încrucişare.

În studiul algoritmilor genetici au fost utilizate şi tehnici din statistică precum lanțul lui Markov şi modele Bayesiene.

2.7 Elementele unui algoritm genetic

Un algoritm genetic trebuie să aibă definite următoarele elemente pentru rezolvarea unei probleme:

- o reprezentare (genotip) a soluțiilor candidat;
- o procedură de inițializare (creare) a populației inițiale de soluții candidat;
- o funcție de evaluare (funcție fitness) care joacă rolul mediului şi care este utilizată pentru a măsura calitatea soluțiilor în termeni de potrivire/ acomodare;
- o schemă de selecție (înlocuirea generațiilor);
- operatorii genetici (mutația, încrucişarea, …);
- parametrii numerici.

Reprezentarea clasică în algoritmii genetici este cea binară: fiecare soluție candidat (cromozom) este reprezentat ca un şir de biți, fiecare bit constituind o genă. Algoritmii genetici au evoluat ei înşişi, setul reprezentărilor extinzându-se la numere întregi şi reale, permutări,arbori, reprezentări bi- şi multi-dimensionale. Pentru orice reprezentare alta decât cea binară nu există rezultate teoretice care să explice aplicațiile spectaculoase ale algoritmilor genetici la probleme de optimizare dificilă.

Procedura de inițializare este una aleatoare. Există şi abordări în care soluțiile inițiale sunt obținute cu o euristică (ex. greedy) şi apoi supuse evoluției cu un algoritm genetic.

Funcția fitness se obține plecând de la funcția matematică de optimizat. Complexitatea funcției de evaluare poate creşte complexitatea intrinsec polinomială a algoritmului genetic.

Selecția este un element comun mai multor tehnici de optimizare (hill-climbing, simulated annealing). Selecția în algoritmii genetici este una probabilistă. Esențial este caracterul neextinctiv: şi cel mai slab (ca fitness) individ are şansă nenulă de supraviețuire. În acest mod, informație locală (un

număr redus de biți) aparținând unui individ cu fitness slab poate fi transmisă în generațiile următoare pentru a se regăsi în final în structura soluției optime. Au fost propuse şi studiate numeroase tipuri de selecție: scheme bazate direct pe valorile fitness, scheme bazate pe rangul în ierarhie, selecția proporțională, sistem turneu, etc.

Operatorii genetici de bază – încrucişarea şi mutația - apar în forme diverse, fiind necesară în prealabil adaptarea lor la problemă prin încorporarea proprietăților/informației specifice problemei de rezolvat.

Încrucişarea clasică este cea cu tăiere într-un singur punct şi interschimbarea segmentelor celor doi cromozomi implicați.

$$x_1\, x_2\, x_3 \,|\, x_4\, x_5\, x_6\, x_7\, x_8 \longrightarrow x_1\, x_2\, x_3\, y_4\, y_5\, y_6\, y_7\, y_8$$
$$y_1\, y_2\, y_3 \,|\, y_4\, y_5\, y_6\, y_7\, y_8 \qquad\qquad y_1\, y_2\, y_3\, x_4\, x_5\, x_6\, x_7\, x_8$$

În literatură au apărut însă şi încrucişări cu două sau mai multe puncte fixe, încrucişarea uniformă, cu mai mulți părinți, încrucişări specifice pentru permutări, grafuri/arbori sau pentru alte reprezentări multidimensionale.

Operatorul clasic de **mutație** neagă valoarea unei gene (bit) alese aleatoriu.

$$x_1x_2x_3x_4x_5x_6x_7x_8 \longrightarrow x_1x_2x_3x_4x_5x_6\overline{x_7}x_8$$

Mai există mutații specifice problemelor de permutare şi mutații euristice.

În proiectarea unui algoritm genetic trebuiesc luate decizii şi în ceea ce priveşte valorile unor **parametrii** precum dimensiunea populației, rata (probabilitatea de aplicare) mutației şi încrucişării şi stabilirea unei condiții de oprire a algoritmului. În afara unor considerații generale (de exemplu rata mutației crescută la începutul vieții populației şi scăzută spre sfârşit, cu evoluția complementară a ratei încrucişării), găsirea valorilor optime ale parametrilor unui algoritm genetic ține deocamdată mai mult de empirism decât de un studiu teoretic abstract.

2.8 Optimizarea unui algoritm genetic

Pentru o problemă dată pot fi proiectați mai mulți algoritmi genetici, mai mult sau mai puțin asemănători, prin variații de la schema generală şi

existența mai multor posibilități de definire a elementelor constitutive enumerate mai sus.

Reprezentarea soluțiilor candidat şi funcția fitness sunt dependente de problemă iar pentru o problemă dată pot fi construite diverse reprezentări şi pot fi formulate mai multe funcții fitness.

Operatorii pot fi selectați dintre cei standard însă este necesară şi pentru aceştia o prealabilă adaptare la problemă astfel încât să înglobeze informație specifică problemei. De exemplu, pentru probleme de permutare s-au formulat diverşi operatori de încrucişare care păstrează structura de permutare a soluției.

De asemenea există mai multe tipuri de selecție (în jur de 10 scheme studiate) care favorizează mai mult sau mai puțin supraviețuirea indivizilor cu grad ridicat de adaptare (cu valori mari ale funcției fitness).

Alegerea valorilor parametrilor numerici care intervin în structura algoritmului este o altă problemă de decizie în proiectarea algoritmului genetic iar numărul de posibilități este de ordinul zecilor sau poate chiar al sutelor.

În urma unui calcul succint, numărul de algoritmi genetici diferiți care pot fi scrişi pentru rezolvarea unei probleme este de ordinul sutelor de mii. Dintre aceştia, majoritatea nu vor converge la soluția optimă sau nu vor fi eficienți din punct de vedere al spațiului/timpului de lucru necesar algoritmului pentru a atinge convergența; o parte dintre ei însă, vor depăşi alți algoritmi existenți. Determinarea celui mai bun algoritm genetic care poate fi scris pentru o problemă dată este o problemă de optimizare în sine.

Există două abordări pentru optimizarea unui algoritm genetic: optimizarea empirică şi utilizarea unui algoritm genetic supervizor.

Optimizarea empirică a parametrilor unui algoritm genetic constă din rulări multiple ale acestuia, cu setări diferite ale parametrilor, respectiv cu alegeri pentru elementele algoritmului, sistematic alese dintre cele posibile.

Algoritmul genetic supervizor SAG (meta-algoritm genetic) optimizează parametrii numerici ai unui algoritm genetic de bază AG. Elementele SAG sunt cele uzuale, cu excepția meta-cromozomilor care conțin câte o meta-genă pentru fiecare parametru de optimizat al algoritmului AG (dimensiunea populației, rata mutației, rata încrucişării, un bit pentru elitism). SAG este foarte costisitor ca timp deoarece evaluarea unui meta-cromozom constă din rularea repetată a algoritmului AG cu parametrii setați conform meta-genelor, fitnessul acestuia fiind egal cu performanța medie a AG astfel setat (fitnessul celui mai bun individ sau media valorilor fitness a celor mai buni indivizi obținuți în urma unui număr de rulări consecutive ale GA). De

obicei, meta-algoritmii genetici au populații de dimensiuni mici şi număr redus de generații.

2.9 Schema generală a unui algoritm genetic

Un algoritm genetic efectuează o căutare multi-dimensională prin utilizarea unei populații de soluții candidat (cromozomi) care evoluează de-a lungul unui număr de generații în care efectuează schimb de informații. La iterația *t* fiecare individ din populația curentă P(*t*) este evaluat. Apoi o nouă populație P(*t+1*) este formată prin selectarea celor mai buni indivizi. Câțiva membri din noua populație sunt supuşi recombinării prin intermediul mutației şi încrucişării pentru a forma noi soluții. Acest proces este ilustrat de următorul pseudo-cod:

```
INIȚIALIZARE.   begin t:=0
                generează P(t)
                evaluează P(t)
ITERARE         while (not COND_OPRIRE) do begin
                        t:=t+1
                        selectează P(t) din P(t-1)
                        recombină P(t)
                        evaluează P(t)
                end
```

Această schemă se aplică pentru majoritatea algoritmilor evolutivi.

Fiind algoritmi de tip probabilist, un generator de numere aleatoare este necesar în mai multe etape ale algoritmului: generarea populației inițiale dacă aceasta se face aleator, selecția indivizilor pentru supraviețuire, alegerea indivizilor pentru recombinare, aplicarea operatorilor genetici.

Cel mai bine adaptat individ din ultima generație reprezintă soluția pe care algoritmul genetic o găseşte în rularea respectivă. Din cauza caracterului nedeterminist , comportamentul unui algoritm genetic este considerat în medie, pentru mai multe execuții diferite, cu inițializări diferite ale generatorului de numere aleatoare.

Schema generală a unui algoritm genetic este polinomială dacă se consideră instrucțiunile din descrierea de mai sus ca fiind elementare. Singurul pas care poate modifica această complexitate polinomială este evaluarea. Funcția fitness obținută din modelul matematic al problemei poate duce la evaluare exponențială existând astfel probleme de complexitate

exponențială la aproximarea prin algoritmi genetici.

Există numeroase variații şi adaptări ale schemei generale pentru diverse clase de probleme. Unele dintre acestea privesc doar structura şi evoluția populației:

- Algoritmii genetici **steady-state** (cu stare stabilă), în care diferența dintre populațiile a două generații succesive constă doar din modifcarea a doi indivizi;
- Algoritmi genetici cu reprezentare pe niveluri diferite („**delta-coding**", „dynamic parameter encoding");
- Algoritmi genetici **distribuiți**, inclusiv cu distribuire spațială, în care un număr de sub-populații evoluează relativ independent;
- Modele **co-evolutive** – "pradă-victimă" – pentru optimizarea prin acelaşi algoritm a două probleme duale;
- Algoritmi genetici **paraleli**, etc.

2.10 Algoritmi genetici - exemple

În ceea ce urmează sunt descrişi trei algoritmi genetici utilizați în rezolvarea unor probleme din domenii diferite diferite. Un prim exemplu prezintă un algoritm genetic pentru optimizarea unei funcții simple de variabilă reală. Al doilea exemplu aplică un algoritm genetic pentru o problemă combinatorie NP-dificilă, problema comis-voiajorului. Al treilea exemplu ilustrează utilizarea algoritmilor genetici în domeniul învățării automate pentru determinarea unei strategii într-un joc simplu.

2.10.1 Un exemplu numeric simplu[1]

Problema propusă spre rezolvare în această secțiune este formulată în felul următor:

Să se determine valoarea variabilei x în intervalul [-1; +2] care maximizează funcția $f(x) = x \cdot \sin(10\pi \cdot x)+1$. Precizia cu care exprimăm rezultatul se doreşte a fi de 6 zecimale.

[1] Exemplu preluat din (Michalewicz 1996)

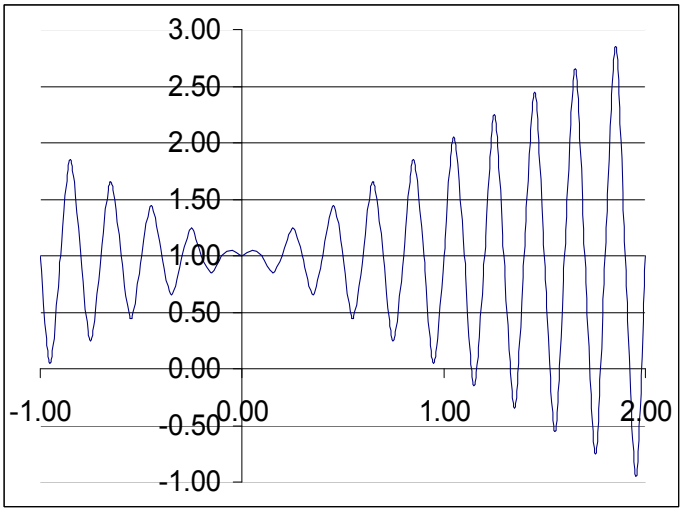

Fig. 4: Graficul funcției f(x) = x·sin(10π·x)+1.

Elementele unui algoritm genetic care rezolvă această problemă sunt descrise în continuare.

Reprezentarea unui cromozom (soluție candidat) se realizează sub forma unui vector binar al cărui lungime depinde de precizia cerută. Domeniul variabilei x are lungime 3; restricția de precizie implică o divizare a intervalului [-1; 2] în cel puțin $3 \cdot 10^6$ subintervale de lungime egală. Aceasta înseamnă ca sunt necesari 22 biți pentru reprezentare: $2097152 = 2^{21} < 3 \cdot 10^6 < 2^{22} = 4194304$.

Decodificarea unui şir binar într-un număr real din intervalul [-1; 2] este realizată în doi paşi: convertirea şirului din baza 2 în baza 10 şi determinarea numărului real x corespunzător:

1. $(b_{21}\ b_{20}\ \ldots b_0) \rightarrow \sum b_i \cdot 2^i = x'$
2. $x = -1 + x' \cdot (3/(2^{22} - 1))$

De exemplu, cromozomii

v1=(1000101110110101000111)
v2=(0000001110000000010000)
v3=(111000000**0**111111000101)

corespund valorilor numerice

x_1= +0.637197
x_2= -0.958973
x_3= +1.627888.

Limitele domeniului, -1 şi 2, corespund cromozomilor

(00000000000000000000) şi respectiv (11111111111111111111) .

Funcția de evaluare a cromozomilor este identică cu funcția studiată *f*. Pentru cei trei cromozomi valorile funcției de evaluare sunt după cum urmează:

$eval(v_1) = f(x_1) = 1.586245;$
$eval(v_2)=f(x_2) = 0.078878;$
$eval(v_3)=f(x_3) = 2.250650.$

Operatorii genetici utilizați sunt cei standard. **Mutația** modifică cu o probabilitate dată de valoarea ratei mutației bitul de pe o poziție selectată aleatoriu. De exemplu, să presupunem că a cincia genă a cromozomului v_3 a fost selectată. După aplicarea mutației cromozomul v_3 devine:

v'3 = (11100000**1**111111000101)

În urma evaluării se constată că s-a obținut un cromozom mai bun:

$x'_3 = +1.630818 \rightarrow eval(v_3') = f(x_3') = 2.343555 > f(x_3) = eval(v_3)$

Operatorul de **încrucişare** utilizat este cel cu un singur punct de tăiere. Dacă utilizăm cromozomii v_2 şi v_3 drept părinți iar punctual de tăiere a fost ales aleatoriu între genele 5 şi 6 obținem descendenții v'_2 şi v'_3

v_2 = (00000 | 01110000000010000)
v_3 = (11100 | 00000111111000101)
v'_2=(00000|00000111111000101)→x'_2=-0.99811
v'_3=(11100|01110000000010000)→x'_3=+1.66602

În urma evaluării celor doi cromozomi descendenți se constată că unul din aceştia este mai bine adaptat decât ambii părinți:

$eval(v'_2) = f(x'_2) = 0.940865 > eval(v_2)$
$eval(v'_3) = f(x'_3) = 2.459245 > eval(v_3) > eval(v_2)$

Principalii **parametrii** ai algoritmului genetic au luat următoarele valori: dimensiunea populației pop_size = 50, rata încrucişării $p_c = 0.25$, rata mutației $p_m = 0.01$.

În urma execuției algoritmului genetic cel mai bun cromozom la iterația 50 a fost v_{max} = (11110011010001000000101) corespunzător valorii reale x_{max} = 1.850773; valoarea funcției în acest punct este $eval(v_{max}) = f(x_{max})=$

2.850227.

Următorul tabel prezintă evoluția fitnessului celui mai bun individ curent pe parcursul generațiilor.

1	10	40	100
1.441942	2.250363	2.345087	2.849246

2.10.2 Un exemplu combinatorial

Utilizarea algoritmilor genetici în optimizarea combinatorie este ilustrată pe **problema comis-voiajorului**: dat costul drumului între oricare două oraşe, obiectivul este determinarea celui mai puțin costisitor itinerariu care trece o singură dată prin fiecare oraş şi se încheie în punctul de plecare.

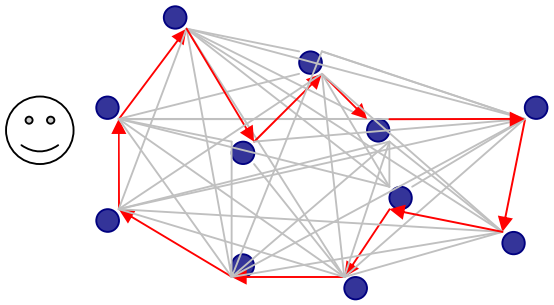

Fig. 5: Problema comis-voiajorului: cel mai scurt circuit Hamiltonian într-un graf complet

În ultimii ani s-au înregistrat mai multe încercări de a aproxima această problemă cu algoritmii genetici. În ceea ce urmează este prezentată una din metodele utilizate.

Restricția ca oraşele să fie vizitate o singură dată este înglobată în **reprezentarea** soluțiilor candidat prin utilizarea de permutări de numere întregi. Devine astfel necesară utilizarea de operatori genetici specializați care să nu construiască reprezentări ilegale.

Mutația este definită ca inversarea a două gene într-un individ.

Încrucişarea, al cărei scop este de a combina informația genetică din doi sau mai mulți cromozomi bine adaptați păstrează structura de permutare a descendenților prin următoarea procedură (încrucişare OX): se copie o subsecvență din alcătuira unui cromozom părinte şi se completează păstrând

ordinea de apariție din cel de al doilea.

Spre exemplu, dacă părinții sunt

P: (1 2 3 **4 5 6 7** 8 9 10 11 12) (7 3 1 **11 4 12 5** 2 10 9 6 8)

şi se selectează pentru copiere genele de pe pozițiile 4-7 rezultatul aplicării încrucişării îl constituie doi cromozomi în care se regăsesc relații structurale cu ambii părinți:

O: (3 1 11 **4 5 6 7** 12 2 10 9 8) (1 2 3 **11 4 12 5** 6 7 8 9 10)

Rezultatele obținute în urma a 20 de rulări pe o instanță generată aleatoriu cu 100 de oraşe indică o valoare a întregului tur cu 9,4% mai mare decât valoarea turului optim.

2.10.3 Un exemplu din învățarea automată

În această secțiune este descris un algoritm genetic utilizat pentru învățarea unei strategii într-un joc simplu cunoscut sub numele de “dilema prizonierului”.

Doi prizonieri sunt ținuți în celule separate fără a avea posibilitatea de a comunica unul cu celălalt. Fiecărui prizonier i se cere să-l trădeze pe celălalt. În funcție de acțiunile lor, prizonierii sunt recompensați sau pedepsiți după cum urmează:

- dacă doar un singur prizonier trădeaza, el este recompensat iar celălalt este pedepsit;
- dacă ambii prizonieri trădează, ambii sunt pedepsiți;
- dacă nici un prizonier nu trădează, vor primi amândoi o recompensă moderată.

Dilema prizonierului este de a lua o decizie: să trădeze sau să coopereze cu celălalt prizonier.

Tabelul de mai jos exprimă scorul obținut de fiecare jucător în funcție de deciziile luate:

Jucător1	Jucător2	Punctaj1	Punctaj2
Trădează	Trădează	1	1
Trădează	Cooperează	5	0
Cooperează	Trădează	0	5
Cooperează	Cooperează	3	3

Problema este aşadar de a învăța o strategie care să maximizeze numărul de puncte obținute.

În ceea ce urmează sunt prezentate elementele algoritmului genetic

utilizat de Axelrod(1987) pentru învățarea unei strategii.

Pentru simplitate sunt considerate strategii deterministe bazate pe ultimele 3 mişcări. Pentru fiecare mişcare sunt posibile 4 variante de răspuns din partea jucătorului ceea ce duce la $4^3 = 64$ istorii diferite pentru ultimele trei mişcări. O strategie poate fi specificată indicând ce mişcare trebuie făcută pentru fiecare din aceste istorii posibile. Aşadar în reprezentarea se poate utiliza un şir de 64 biți care păstrează decizia luată de fiecare jucător: trădare sau cooperare. La acest şir se mai adaugă 6 biți care specifică ultimele 3 mişcări din evoluția jocului. Aceasta duce la o **reprezentare** pe 70 biți a unui cromozom care specifică decizia luată de jucător în fiecare circumstanță posibilă. Utilizând această reprezentare pot fi definite un total de 2^{70} strategii.

Populația inițială formată din şiruri a câte 70 biți este generată aleator.

La fiecare iterație are loc evaluarea fiecărui jucător (cromozom) luând ca adversari strategii predefinite sau fiecare individ din populație. Valoarea **fitnessului** este considerat ca fiind media tuturor jocurilor la care participă.

Procedura de **selecție** este următoarea: un individ ce a obținut un scor mediu este ales o singură dată pentru reproducere; un individ ce a obținut un scor mai mare cu o deviație standard decât media este selectat de două ori; indivizii care au obținut un scor cu o deviație standard sub media populației nu sunt selectați deloc.

Indivizii selectați sunt supuşi mutației şi încrucişării.

Experimentele s-au desfăşurat pe un număr de 20 de indivizi, iar populația a evoluat de-a lungul a 50 generații.

Într-un prim experiment s-a considerat un mediu fix: indivizii au fost evaluați utilizând cele mai bune 8 strategii construite de om. Deşi au fost vizitate maxim 20*50=1000 strategii din 2^{70} posibile rezultatele obținute au fost la fel de bune sau chiar mai bune cu cele înregistrate de strategia „Ochi pentru ochi” (începe prin a coopera şi apoi fă ce a făcut adversarul la pasul anterior).

Într-un al doilea experiment mediul utilizat a fost dinamic: fiecare individ a fost evaluat luând ca adversar pe fiecare din cei 20 membrii ai populației. În primele generații s-au obținut strategii necooperative. După 10-20 generații s-au înregistrat tot mai multe strategii care funcționează în maniera „Ochi pentru ochi”.

Cele trei exemple prezentate ilustrează domeniul larg de aplicabilitate al algoritmilor genetici şi în acelaşi timp oferă câteva indicii asupra dificultăților ce pot apărea în proiectarea acestor algoritmi: reprezentarea soluțiilor nu este

întotdeauna imediată, funcția de evaluare nu este clar definită (vezi problema de învățare), operatorii trebuie adaptați la problemă pentru a îngloba cunoştințe specifice acesteia.

2.11 Cum lucrează algoritmii genetici

Etapele implementării şi utilizării unui algoritm genetic sunt următoarele:

- definirea elementelor algoritmului (reprezentarea, funcția fitness, mecanismul de selecție, operatorii genetici, parametrii)
- proiectarea experimentului
- execuția experimentului
- interpretarea rezultatelor

Analiza unui algoritm evolutiv se face empiric, pe baza rezultatelor unor experimente ce urmăresc fie performanța absolută de calcul a algoritmului studiat, fie compararea algoritmului genetic studiat cu un alt algoritm ce rezolvă aceeaşi problemă (studiu relativ). De aceea, în faza de proiectare a experimentului trebuie avută în vedere optimizarea algoritmului genetic şi, pentru al doilea caz, considerați alte tipuri de algoritmi decât cei genetici pentru efectuarea de comparații.

În algoritmul genetic clasic, funcția de evaluare trebuie să fie strict pozitivă, iar asupra ei să se realizeze maximizare. Ambele condiții sunt uşor de statisfăcut atunci când funcția fitness este dată de o funcție reală: funcția de evaluare poate fi uşor modificată prin translarea cu o constată iar, dacă este cazul, problema de minimizare poate fi exprimată ca problemă de maximizare prin înmulțirea cu –1 a funcției fitness.

Nu orice problemă poate fi exprimată ca problemă de optimizare a unei funcții reale. Pentru situații în care funcția fitness nu are o expresie algoritmică sau este necunoscută - mai ales în unele aplicații de inteligență artificială – se poate folosi evaluarea interactivă, în care utilizatorul stabileşte on-line fitnessul fiecărui individ sau ierarhia generației.

Algoritmii genetici clasici sunt formulați pentru optimizarea uni-criterială. În practică însă sunt numeroase probleme în care trebuiesc urmărite mai multe obiective. Un exemplu este problema orarului în care, în afara constrângerilor de natură materială care trebuiesc satisfăcute (suprapunerea în aceeaşi sală a două cursuri, resurse materiale limitate –videoproiector, laptop – distribuite între profesori, la o oră dată un profesor ține / un student participă la cel mult un curs), sunt necesare şi optimizări din punct de vedere al timpului alocat (cât mai puține ferestre în orarul unui profesor/student).

Soluția preferată în rezolvarea unor astfel de probleme este construirea unui criteriu global (modele liniare sau neliniare) în care fiecărui subcriteriu i se acordă mai multă sau mai puțină importanță.

Algoritmii genetici sunt algoritmi care îmbunătățesc soluția pas cu pas de-a lungul mai multor generații. Există probleme însă de tipul „acul în carul cu fân" care prin formulare nu permit o îmbunătățire pas cu pas. Un exemplu în acest sens este problema satisfiabilității în care, dată o formulă în logica Booleană peste un număr de *k* variabile booleene se cere o asignare a acestora astfel încât întreaga formulă sa fie satisfiabilă (rezultatul evaluării să fie 1 echivalentă valorii booleene adevărat). Pentru această problemă poate fi scris un algoritm genetic clasic în care reprezentarea soluțiilor se face sub forma unui şir de *k* biți, iar operatorii genetici sunt cei standard. Funcția fitness dă valoarea de adevăr a expresiei booleene sub asignările date de cromozomi. Dificultatea rezidă în faptul că evaluând cromozomii cu o funcție fitness ce are doar două valori nu se poate face îmbunătățirea pas cu pas iar învățarea devine imposibilă. Trebuie găsită o modalitate de a ierarhiza indivizii nesatisfiabili iar acest lucru este posibil utilizând cunoştințe suplimentare din domeniul problemei. O soluție este exprimarea problemei în formă normală conjunctivă echivalentă; pentru o astfel de formulare a problemei, îmbunătățirea soluțiilor poate fi realizată pas cu pas considerând ca funcție fitness numărul de termeni care se evaluează la valoarea booleană adevărat.

Capitolul 3

Algoritmii genetici în optimizarea numerică

Optimizarea numerică se referă la studiul problemelor în care se încearcă minimizarea sau maximizarea unei funcții reale prin alegerea valorilor variabilelor reale sau întregi dintr-o mulțime permisă.

Forma generală a unei probleme în optimizarea numerică este următoarea:

Dată o funcție $f : R^k \rightarrow R$ pozitivă $(f > 0)$ să se determine un element x^{opt} din R^k cu restricțiile $x_i^{opt} \in [a_i ; b_i]$ astfel încât $f(x^{opt}) \geq f(x)$ oricare ar fi $x \in R^k$.

O problemă de minimizare a unei funcții g poate fi exprimată în forma de mai sus considerând $f = -g$. În acest caz soluția problemei de minimizare a funcției g este aceeaşi cu soluția problemei de maximizare a funcției f:

$$\min g(x) = \max \{-g(x)\} = \max f(x).$$

O problemă de maximizare a unei funcții g care nu este pozitivă poate fi exprimată în forma de mai sus considerând funcția f de forma $f = g + C$ unde C este o constantă. Soluția problemei de maximizare a funcției g este soluție a problemei de maximizare a funcției f:

$$\max g(x) = \max \{g(x) + C\} = \max f(x).$$

În rezolvarea unei probleme de optimizare numerică cu algoritmi genetici este necesară fixarea apriori a preciziei soluțiilor. Să considerăm că optimizarea se face cu precizia de 6 zecimale.

Pentru a obține această precizie este necesară divizarea spațiului de căutare într-un număr de $(b_i - a_i)\cdot 10^6$ subintervale de lungime egală. Se caută cel mai mic întreg m_i astfel încât $(b_i - a_i)\cdot 10^6 \leq 2^{mi} - 1$. Fiecare componentă x_i a unei soluții va fi reprezentată ca un şir de m_i biți. Interpretarea acestui şir de biți se face după următoarea formulă de calcul:

$$x_i = a_i + decimal(d_{m_i-1}...d_1 d_0)\cdot \frac{b_i - a_i}{2^{m_i} - 1}$$

unde *decimal*(şir_binar) reprezintă valoarea zecimală a şirului.

Reprezentarea unei soluții candidat se face sub forma unui şir de biți rezultat din concatenarea celor k componente: lungimea necesară este $m = \sum_{i=1,k} m_i$, unde k este numărul de dimensiuni al spațiului de căutare iar m_i este numărul de biți necesari pentru a reprezenta o valoare din intervalul $[a_i; b_i]$ cu precizia dată.

Inițializarea populației se poate face în mod aleatoriu prin construirea a *pop_size* cromozomi prin setări aleatoare ale biților sau se pot utiliza soluții obținute cu alte euristici.

Pentru **evaluarea** cromozomilor se face mai întâi decodificarea acestora după procedeul descris mai sus; după obținerea pentru cromozomul (şirul) *v* a celor *k* componente ale soluției *x* din spațiul numerelor reale se calculează valoarea funcției *f* în punctul *x*: **eval(v)=f**(**x**) .

Algoritmul este oprit după un număr fixat de generații sau atunci când nu se observă nici o îmbunătățire în fitnessul soluțiilor candidat.

Procedura de selecție are loc în acord cu o distribuție de probabilități bazată pe fitnessul indivizilor. Cea mai bună procedură cunoscută este "roata norocului", în care, probabilitatea de selecție a fiecărui cromozom este direct proporțională cu valoarea fitnessului său. Paşii necesari în efectuarea unei selecții de acest tip sunt următorii:

- se calculează fitnessul fiecărui individ v_i, eval(v_i);
- se calculează fitnessul total ca suma fitnessului tuturor indivizilor:

$$F = \sum_{i=1}^{pop\_size} \mathbf{eval}(v_i)$$

- se calculează probabilitățile de selecție individuale: p_i=**eval**(v_i) / F
- se calculează probabilitățile de selecție cumulate:

$$q_i = \sum_{j=1}^{i} p_j$$

Procesul de selecție are loc prin învârtirea ruletei de *pop_size* ori: de fiecare dată este selectat pentru supraviețuire un cromozom din populația curentă şi pus în populația intermediară. Acest lucru este realizat în modul următor:

- se generează aleatoriu un număr real *r* în intervalul [0, 1];
- dacă ($q_{i-1} < r \leq q_i$) atunci selectează v_i.

În principiu, utilizând această schemă de selecție, populația intermediară va conține mai multe copii ale celor mai buni cromozomi şi aproximativ o copie a indivizilor de calitate medie, în timp ce cromozomii de calitate slabă dispar.

Populația intermediară obținută în urma selecției este supusă operatorilor genetici – încrucişarea şi mutația. Fiecare operator utilizează un parametru numeric care dă probabilitatea sa de a fi aplicat.

Dacă probabilitatea încrucişării este p_c, atunci numărul total estimat de indivizi care sunt supuşi încrucişării este $p_c \cdot pop\_size$. Indivizii sunt selectați în

modul următor din populația intermediară: pentru fiecare cromozom

- se generează un număr aleatoriu r în intervalul [0, 1];
- dacă $r < p_c$ individul curent este selectat.

Indivizii astfel selectați sunt împerecheați aleatoriu. Pentru fiecare pereche încrucişarea are loc în două etape: se generează aleatoriu un număr în mulțimea {1,...,m-1} care va constitui punctual de tăiere şi se înlocuiesc cei doi părinți cu descendenții lor rezultați în urma interschimbării fragmentelor de o parte şi de alta a punctului de tăiere. Procedeul este ilustrat în continuare: considerând cromozomii părinți P1 şi P2 şi punctul de tăiere *pos*, se obțin cromozomii descendenți O1 şi O2.

$P1 = (a_1\ a_2 \ldots a_{pos}\ a_{pos+1} \ldots a_m)$ $\quad O1 = (a_1\ a_2 \ldots a_{pos}\ b_{pos+1} \ldots b_m)$

$P2 = (b_1\ b_2 \ldots b_{pos}\ b_{pos+1} \ldots b_m)$ $\quad O2 = (b_1\ b_2 \ldots b_{pos}\ a_{pos+1} \ldots a_m)$

Operatorul de mutație se aplică pe populația intermediară rezultată în urma aplicării încrucişării. Probabilitatea mutației p_m dă numărul estimat de biți care vor fi supuşi mutației: $p_m \cdot m \cdot pop\_size$. Fiecare bit aceeaşi şansă de a fi supus mutației. Pentru fiecare cromozom din populația intermediară şi pentru fiecare bit din cromozom

- se generează un număr aleatoriu r în intervalul [0, 1];
- dacă $r < p_m$ valoarea bitului este negată.

Evoluția acestui algoritm genetic este ilustrată pe următoare problemă de maximizare cu două variabile:

$$\mathbf{f}(x_1,x_2) = 21.5 + x_1 \cdot \sin(4\pi x_1) + x_2 \cdot \sin(20\pi x_2),$$

cu restricțiile: $-3.0 \le x_1 \le 12.1$ şi $4.1 \le x2 \le 5.8$.

Se cere o precizie de 4 zecimale pentru fiecare variabilă ceea ce necesită o divizare a intervalului [-3.0, 12.1] în cel puțin $15.1 \cdot 104$ subintervale şi a intervalului [4.1, 5.8] în cel puțin $1.7 \cdot 104$ subintervale de lungime egală. Sunt astfel necesari 18 biți pentru reprezentarea variabilei x1 şi 15 biți pentru reprezentarea variabilei x2:

$217 < 151000 < 218$; $\quad 214 < 17000 < 215$

Lungimea totală a reprezentării unui individ este de 33 biți, primii 18 biți reprezentând codificarea primei variabile iar următorii 15 biți codificarea celei de a doua variabile. De exemplu, cromozomul

v = (010001001011010000.111110010100010)

corespunde perechii (1.0524, 5.7553). Fitnessul acestui cromozom este egal

cu

eval(v) = f(1.0524, 5.7553) = 20.2526.

În continuare este descris modul de desfăşurare al experimentului şi sunt prezentate rezultatele obținute pentru o rulare[2].

Parametrii algoritmului genetic utilizat sunt după cum urmează: dimensiunea populației *pop_size* = 20, probabilitatea încrucişării p_c = 0.25, probabilitatea mutației p_m = 0.01.

Populația inițială a fost creată aleatoriu. În urma evaluării s-a determinat cel mai bun individ ca fiind v_{15} iar cel mai slab ca fiind v2.

Fitnessul total al populației inițiale a fost F=(eval(v_i)=387.7768.

S-au calculat probabilitățile de selecție pi pentru fiecare cromozom.

S-au calculat valorile probabilităților cumulate q_i.

Prin rotirea roții norocului de 20 de ori s-a creat populația intermediară. Din aceasta s-au selectat indivizii care participă la încrucişare. Deşi numărul estimat este 5 doar 4 cromozomi au fost selectați. Aceştia sunt împerecheați aleatoriu pentru încrucişare. În cazul în care numărul de indivizi ar fi fost impar ar fi fost necesară selecția unui individ în plus sau eliminarea unuia deja selectat. De exemplu, prima pereche supusă încrucişării a fost formată din perechea

v_2'= (100011000.10110100111100000l110010)
v_{11}'= (111011101.10111000010001111011110)

perechea rezultată fiind

v_2''= (100011000. 10111000010001111011110)
v_{11}''=(111011101. 10110100111100000l110010)

Pentru aplicarea mutației este necesară efectuarea selecției de 660 ori. Numărul estimat de mutații este $p_m \cdot m \cdot pop\_size$=6.6. Numărul de gene afectate a fost 5. Următorul tabel reprezintă poziția genei alese pentru mutație (din 660 posibile), cromozomul afectat şi poziția bitului afectat în cromozom.

Poziția bitului (populație)	Numărul cromozomului	Poziția bitului (cromozom)
112	4	13
349	11	19

[2] Exemplu preluat din (Michalewicz 1996)

418	13	22
429	13	33
602	19	8

Doar 5 cromozomi au fost afectați de operatorul de mutație, iar din aceştia unul a suferit două mutații.

Valoarea fitnessului total al populației după prima iterație a fost F = 447.049, mult mai mare decât fitnessul populației anterioare.

După 1000 generații cel mai bun cromozom obținut a fost

v_{11}=(11010110000001001000110001011000 0)

iar fitnessul acestuia are valoarea

eval (v_{11}) = f (9.6236 , 4.4278) = 35.4779.

Într-o generație intermediară a algoritmului (generația numărul 396) s-a obținut însă un individ cu o valoare a fitnessului mai mare: **eval** (*best_so_far*) = 38.8275. Pentru a nu pierde astfel de indivizi cu valori mari ale fitnessului şi care au şansa de a deveni soluție este necesară memorarea lor. În final, algoritmul nu va raporta ca soluție cromozomul din generația finală cu cea mai bună valoare a fitnessului ci cel mai bun individ întâlnit pe parcursul căutării.

Motivul pentru care indivizi foarte buni sunt eliminați din populație îl constituie erorile stocastice de selecție (numere pseudo-aleatorii, populații finite, număr finit de generații).

Capitolul 4
Teorema schemelor

În studiul algoritmilor genetici este evidentă necesitatea unui model teoretic care să poată da răspunsuri unor întrebări precum:

- ce legi descriu comportamentul macroscopic al algoritmilor genetici
- ce efect au operatorii genetici (selecția, încrucişarea, mutația) asupra comportamentului macroscopic al algoritmilor genetici
- ce face o problemă potrivită pentru rezolvarea cu un algoritm genetic
- ce face o problemă nepotrivită pentru rezolvarea cu algoritmii genetici
- ce criterii de performanță sunt relevante

Primul demers în acest sens este cel al lui Holland(1975) care propune un model ce explică modul în care un algoritm genetic standard converge la soluție, fără a face însă predicții asupra producerii sau nu a convergenței, a acurateții soluțiilor găsite sau asupra vitezei de convergență. Rezultatul său central este teorema schemelor.

Investigând performanțele algoritmilor genetici, Forrest, Holland şi Mitchell au propus în 1992 funcțiile Royal Roads care ar fi trebuit să constituie un tip de probleme uşor de rezolvat pentru algoritmii genetici datorită structurii special definite a soluțiilor. În urma unor studii experimentale s-a constatat însă că euristici simple precum hill-climbing depăşesc algoritmii genetici şi numeroase observații în acest sens vin să întregească sfera de înțelegere a acestor algoritmi.

Au fost formulate şi modele matematice exacte care să descrie comportamentul unui algoritm genetic simplu: Vose (1991), Goldberg (1987), Davis (1991), Horn (1993), Whitley (1993).

S-au încercat de asemenea modelări din statistica mecanică: Prügel-Bennett (1994).

4.1 Formularea teoremei schemelor

În cele ce urmează este prezentat modelul lui Holland care dezvoltă mai multe argumente pentru a explica modul în care un algoritm genetic reuşeşte să efectueze o căutare complexă şi robustă. Modelul are la bază reprezentarea soluțiilor sub formă de şir binar şi noțiunea de *schemă*, o formalizare a noțiunii informale de *blocuri de construcție*.

O schemă este o mulțime de şiruri de biți descrisă printr-un cuvânt peste alfabetul {1, 0, *}. Mulțimea tuturor şirurilor care satisfac o schemă dată conține acele şiruri care sunt identice pe toate pozițiile exceptând cele pe care, în schemă, apare caracterul "*".

De exemplu, schemei S_1 de lungime 10 care conține un singur caracter "*"

S_1=(001100011*)

îi corespund cele două şiruri date de mulțimea H_1:

H_1={(0011000111), (0011000110)},

Schemei S_2 de lungime 10 dar cu două simboluri "*"

S_2=(00*100*111)

îi corespund cele 4 şiruri date de mulțimea H_2:

H_2={(0011000111), (0001000111), (0011001111),(0001001111)};

Schema de lungime 10 formată numai cu caracterul "*" reprezintă toate şirurile binare de dimensiune 10 în timp ce o schemă în care nu apare acest simbol reprezintă un singur şir: şirul identic.

În general, o schemă care conține *a* apariții ale simbolului "*"este descrisă de un hiperplan ce conține 2^a şiruri. Pe de altă parte, un şir de lungime *m* este reprezentat de (este instanță a) 2^m scheme. De exemplu, şirul de lungime 2 (01) este instanță a următoarelor scheme: 01, 0*, *1, **.

Pentru toate cele 2^m şiruri de lungime *m* există exact 3^m scheme. Într-o populație de dimensiune *pop_size* pot fi reprezentate între 2^m şi $pop\_size \cdot 2^m$ scheme.

Schemele sunt caracterizate de două proprietăți importante: ordinul schemei şi lungimea de definiție a schemei.

Pozițiile unei scheme care nu sunt ocupate cu caracterul „*" poartă numele de biți *definiți.*

Ordinul unei scheme S, o(S), este dat de numărul de biți definiți în schema respectivă (numărul pozițiilor fixe, ocupate cu 0 sau 1). Această caracteristică este o măsură a specificității schemei. De exemplu, considerând următoarele scheme de lungime 6

S_1 = (**1**0),
S_2 = (11**10),
S_3 = (01110*),

care au ordinele:

$o(S_1) = 2$, $o(S_2) = 4$, $o(S_3) = 5$,

cea mai specifică este schema S_3 care are cea mai mare valoare a ordinului.

Caracterizarea unei scheme prin ordinul ei este utilizată pentru a calcula probabilitatea de supraviețuire a schemei la mutații.

Lungimea de definiție a schemei *S*, $\delta(S)$, este distanța dintre primul şi ultimul bit definit din schemă. Este o măsură a compactității informației conținută în schemă. De exemplu, schemele de mai sus au următoarele lungimi de definiție:

$\delta(S_1) = 6-3 = 3$, $\delta(S_2) = 6-1 = 5$, $\delta(S_3) = 5-1 = 4$.

O schemă care are un singur bit definit va avea o lungime de definiție egală cu 0.

Lungimea de definiție a unei scheme intervine în calculul probabilității de supraviețuire la operații de încrucişare.

Fitnessul static al unei scheme S într-un algoritm genetic este definit ca media aritmetică a valorilor fitness a tuturor instanțelor str_i a schemei:

$$eval(S) = (1/2^r)\cdot\Sigma_i\ eval(str_i)$$

unde *r* este numărul de simboluri „*" din schemă.

În iterația *n* a algoritmului genetic *fitnessul unei scheme S* este definit ca media aritmetică a valorilor fitness a tuturor indivizilor din populație care sunt reprezentați de schema S:

$$\boldsymbol{eval}(S,n) = (1/k)\cdot\Sigma_j \boldsymbol{eval}(str_j),$$

unde k este numărul total de instanțe ale schemei S în populația P(n).

Schemele nu sunt explicit reprezentate şi evaluate într-un algoritm genetic. În timpul execuției unui algoritm genetic nu sunt calculate şi memorate estimări ale fitnessului vreunei scheme. Totuşi, numărul de instanțe ale schemelor reprezentate inițial, creşte sau descreşte în generații succesive ale algoritmului. Dacă inițializarea se face aleatoriu, în prima generație aproximativ jumătate din populație vor fi instanțe ale schemei S = (1*…*) iar cealaltă jumătate instanțe ale schemei T = (0*…*). Evaluările cromozomilor pot fi considerate ca fiind estimări ale fitnessului celor două scheme, eval(S) şi eval(T).

Fie H o schemă reprezentată în generația *t* de un număr $\eta(H,t)$ de instanțe. Fitnessul acestei scheme în această iterație este dat de fitnessul mediu observat *eval(H,t)*. Pentru a urmări evoluția schemei H de-a lungul unui algoritm genetic este necesară estimarea numărului de cromozomi care

satisfac această schemă în iterația t+1, **E**(η(**H**,**t**+1)). În acest sens este studiat efectul fiecărui operator asupra schemei. Algoritmul genetic considerat este cel clasic în care selecția este de tip roata norocului, operatorul de încrucişare este cel cu un singur punct de tăiere iar mutația este cea standard.

În urma selecției un individ primeşte zero, una sau mai multe copii în populația intermediară în funcție de valoarea fitnessului său. Cromozomul x va avea în generația următoare un număr de urmaşi estimat de

$$f(x)/\bar{f}(t)$$

unde f(x) = eval(x) este fitnessul individului iar $\bar{f}(t)$ este fitnessul mediu în generația *t* curentă. Ținând seama de acest rezultat, numărul de cromozomi care satisfac schema H în generația t+1 este

$$E(\eta(\mathbf{H},\mathbf{t}+1)) = \sum_{x\in\mathbf{H}} \mathrm{eval}(x)/\bar{f}(t) = \eta(\mathbf{H},\mathbf{t})\cdot \mathrm{eval}(\mathbf{H},\mathbf{t})/\bar{f}(t),$$

Algoritmul genetic nu calculează explicit eval(H,t) dar această cantitate decide numărul de instanțe ale schemei H în generațiile următoare. Numărul de şiruri în populație creşte o dată cu raportul dintre fitnessul schemei şi fitnessul mediu al populației. Schemele cu fitness peste medie vor primi un număr mai mare de şiruri în generația următoare, schemele cu fitness mai mic decât media vor primi un număr mai mic de şiruri iar cele cu fitness mediu vor primi un număr aproximativ egal.

În urma operatorului de încrucişare vor fi distruse unele instanțe ale schemei H sau vor fi create altele noi. Dacă se consideră doar caracterul destructiv se obțin valori mai mici pentru **E**(η(**H**,**t**+1)). Punctul de tăiere pentru încrucişare este ales uniform din cele *m-1* posibile unde *m* este lungimea cromozomului. Probabilitatea de distrugere a unei scheme este dată de

$D_c(H) = \delta(H)/(m-1)$

ceea ce dă o probabilitate de supraviețuire la încrucişare de

$S_c(H) = 1- \delta(H)/(m-1)$.

Deoarece numărul de cromozomi care sunt supuşi încrucişării este dat de probabilitatea de încrucişare p_c, şi în urma unei încrucişări sunt şanse ca schema să nu fie distrusă deşi punctul de tăiere se află între poziții fixe (atunci când indivizii care participă la încrucişare satisfac aceeaşi schemă), probabilitatea de supraviețuire a schemei H după aplicarea operatorului de încrucişare satisface următoarea relație:

$S_c(\mathbf{H}) \geq 1- p_c\cdot(\delta(\mathbf{H})/(m-1))$.

Probabilitatea de supraviețuire la încrucişare este mai mare pentru schemele mai compacte, adică cele cu valori mici ale lungimii de definiție.

Dacă se consideră operatorul de mutație cu probabilitatea de modificare a unui bit dată de p_m, atunci probabilitatea de supraviețuire a unui singur bit este $1-p_m$. Deoarece mutațiile sunt independente între ele, probabilitatea de supraviețuire a schemei H în urma mutației, echivalentă cu probabilitatea ca nici un bit dintre cei definiți să nu fie mutat, este dată de:

$S_m(\mathbf{H}) = (1-p_m)^{o(\mathbf{H})}$.

Probabilitatea de supraviețuire a unei scheme supuse operației de mutație este mai mare pentru schemele de ordin redus.

Însumând efectul selecției cu efectele celor doi operatori – încrucişarea şi mutația – obținem creşterea minimală a unei scheme H de la o generație la următoarea:

$$E(\eta(H\quad,t+1)) \geq \frac{eval(H,t)}{\bar{f}(t)} \cdot \eta(H,t) \cdot (1-p_c \cdot \frac{\delta(H)}{m-1}) \cdot (1-p_m)^{o(H)}$$

Factorul de creştere este $\frac{eval(H,t)}{\bar{f}(t)}$.

Interpretarea relației de mai sus poartă numele de **teorema schemelor**: schemele scurte, de ordin redus, cu valori ale fitnessului constant peste medie vor primi un număr crescător exponențial de instanțe în generații succesive ale algoritmului genetic.

Teorema schemelor dă o limită inferioară deoarece neglijează "creativitatea" operatorilor.

Un rezultat imediat al teoremei schemelor este ideea că încrucişarea este sursa majoră a puterii algoritmului genetic, recombinând instanțe ale schemelor bune pentru a crea instanțe ale unor scheme cel puțin la fel de bune. Acest rezultat poartă numele de **ipoteza blocurilor de construcție** şi este formulat în modul următor: un algoritm genetic explorează spațiul de căutare prin juxtapunerea schemelor scurte, de ordin redus şi performanță ridicată, denumite blocuri de construcție.

Mutația furnizează diversitate în populație chiar atunci când populația tinde să conveargă. Dacă în populație un bit devine 0, numai mutația oferă şansa de a se face noi încercări cu valoarea 1.

Într-o generație, algoritmul genetic estimează fitnessul mediu al tuturor schemelor care au instanțe în acea generație şi creşte sau descreşte reprezentarea acestor scheme în generația următoare corespunzător valorii acestuia. Are loc aşadar o evaluare implicită a unui număr mare de scheme

utilizând numai *pop_size* cromozomi. Holland a arătat că cel puțin *pop_size*^{3} scheme sunt procesate şi a denumit această proprietate în 1975 drept paralelism intrinsec după care, pentru a evita confuzia cu terminologia din calculul paralel a utilizat termenul de *paralelism implicit*. Un număr de scheme mult mai mare decât dimensiunea populației sunt reprezentate, această explozie combinatorială constituind de data aceasta un avantaj.

În acelaşi timp, algoritmii genetici prin structura lor, se pretează la implementări paralele, proprietate denumită *paralelism inerent*.

În concepția lui Holland, un sistem adaptiv ar trebui să identifice, testeze şi încorporeze proprietăți structurale care au şanse să dea performanță mai bună într-un anumit mediu. Algoritmii genetici trebuie să păstreze un echilibru între explorare şi exploatare. Explorarea are loc prin căutarea de noi adaptări utile, în timp ce exploatarea desemnează utilizarea şi propagarea adaptărilor. Dacă explorarea este neglijată, se ajunge în situația de supra-adaptare, inflexibilitate la noutate, blocarea algoritmului. Insuficientă exploatare duce la sub-adaptare, puține proprietăți sunt câştigate.

4.2 Problema banditului cu două brațe

Pentru a studia modul în care algoritmul genetic utilizează schemele, Holland a utilizat o problemă intens studiată în contextul teoriei deciziilor statistice şi controlului adaptiv (Bellman 1961): problema banditului cu două brațe.

Scenariul este următorul: un jucător are la dispoziție N monede pentru a juca la o maşină cu două brațe etichetate A_1 şi A_2. Câştigul mediu la o încercare pentru cele două brațe este μ_1, respectiv μ_2, şi este stabil de-a lungul timpului (procese staționare, independente unul față de celălalt). Atât câştigurile cât şi variațele σ_1 şi σ_2 sunt necunoscute jucătorului. Acesta poate să le estimeze doar jucând pentru fiecare braț. Problema constă în determinarea unui model de alocare a monedelor pentru fiecare braț, date câştigurile observate, estimarea câştigurilor medii μ_1, μ_2 şi estimarea variațelor σ_1, σ_2. Scopul nu este aşadar determinarea celui mai bun braț, ci maximizarea câştigului în timp ce informația este câştigată în timpul alocării resurselor. Un astfel de criteriu de performanță este denumit on-line deoarece câştigul pentru fiecare încercare participă la evaluarea finală.

Pe măsură ce se obține mai multă informație prin alocări succesive, strategia optimală este de a creşte exponențial probabilitatea de alocare pentru brațul aparent mai bun în defavoarea brațului cu câştiguri mai mici. Ori, această strategie a fost evidențiată de Holland în teorema schemelor pentru

mecanismul de eşantionare a schemelor în algoritmii genetici. Cele 3^m scheme pot fi văzute ca 3^m brațe ale unei maşini cu mai multe brațe. Câştigul observat al unei scheme H este fitnessul mediu observat prin numărul de reprezentanți ai schemei H în populație.

Să presupunem că $\mu_1 \geq \mu_2$,ceea ce înseamnă că A_1 oferă un câştig mediu mai mare decât A_2. Dintr-un număr de N încercări fie n numărul de alocări către A_L şi N-n numărul de alocări către A_H, unde $A_H(N, N-n)$ este brațul cu cel mai mare câştig observat iar $A_L(N, n)$ brațul cu un câştig observat mai mic. Problema se reduce la determinarea numărului n care maximizează profitul de-a lungul a N încercări.

Ideal este ca alocarea monedelor să se efectueze doar la brațul A_1, câştigul estimat fiind $N\cdot\mu_1$.

O pierdere este considerată o încercare în care alocarea se face către brațul cu câştig real mai mic.

Dacă brațul observat ca fiind cel mai bun în urma încercărilor este şi în realitate cel care oferă cele mai mari câştiguri, în cazul nostru $A_H(N, N-n) = A_1$, de-a lungul celor n încercări alocate lui $A_L(N, n)$, jucătorul pierde din profitul estimat $n\cdot(\mu_1 - \mu_2)$.

Dacă dimpotrivă, brațul considerat ca fiind cel mai prost în urma încercărilor este în realitate cel mai bun ($A_L(N, n) = A_1$), pierderea din profitul estimat provine de la cele N-n încercări alocate brațului $A_H(N, N-n)$, şi are valoarea $(N-n)\cdot(\mu_1 - \mu_2)$.

Fie q probabilitatea ca primul caz să aibă loc, $q = Prob(A_L(N, n) = A_1)$ şi fie L(N-n, n) numărul de pierderi pentru cele N încercări. Atunci, următoarea relație are loc:

$$L(N-n,n) = q\cdot(N-n)\cdot(\mu_1 - \mu_2) + (1-q)\cdot n\cdot(\mu_1 - \mu_2)$$

Pentru a detrmina valoarea lui n^* care minimizează expresia lui L(N-n, n) se calculează derivata acesteia în raport cu n:

$$\frac{dL}{dn} = (\mu_1 - \mu_2)\cdot(1 - 2q + (N - 2n)\frac{dq}{dn}) = 0$$

$$\text{dar } q = Prob(\frac{S_2^{N-n}}{N-n} > \frac{S_1^n}{n}) = Prob(\frac{S_1^n}{n} - \frac{S_2^{N-n}}{N-n} < 0),$$

unde S_1^n este suma câştigurilor pentru încercările alocate lui A_1 iar S_2^{N-n} suma câştigurilor pentru încercările de alocare lui A_2.

Deoarece cele două sume sunt variabile aleatoare, diferența acestora este tot o variabilă aleatoare. Determinarea lui q se reduce la determinarea ariei de sub partea distribuției care e mai mică sau egală cu 0. Holland a

aproximat această arie prin utilizarea teoremei limitei centrale pentru distribuții normale. Dan Frantz a corectat această aproximare utilizând teoria marilor deviații şi a obținut următoarea valoare pentru n^*:

$$n^* \approx c_1 \cdot \ln(\frac{c_2 \cdot N^2}{\ln(c_3 \cdot N^2)})$$

unde c_1, c_2 şi c_3 sunt constante pozitive definite de acesta.

Cu ajutorul unor calcule algebrice se obține numărul optim de încercări alocate brațului cel mai bun observat:

$$N - n^* \approx e^{n^*/2c_1} \cdot \sqrt{\frac{\ln(c_3 N^2)}{c_2}} - n^*$$

Pe măsură ce n^* creşte, se poate face aproximarea

$$N - n^* \approx e^{cn^*}$$

unde $c = 1/2c_1$.

Concluzionând, alocarea încercărilor către brațul cel mai bun observat ar trebui să crească exponențial cu numărul de încercări alocate brațului mai prost observat.

Problema banditului cu două brațe ilustrează perfect dificultatea sistemelor adaptive de a echilibra explorarea şi exploatarea.

Teorema schemelor susține că, în condițiile stabilite, algoritmul genetic conține o versiune a strategiei optimale descrise mai sus: populația creşte exponențial în raport cu numărul de încercări alocate celor mai slabe scheme observate.

Interpretarea corectă a analogiei dintre banditul cu două brațe şi scheme nu este totuşi simplă. Grefenstette şi Baker ilustrează acest lucru prin următoarea funcție fitness:

$$f(x) = \begin{cases} 2 \text{ if } x \in 111*\ldots* \\ 1 \text{ if } x \in 0***\ldots* \\ 0 \text{ otherwise} \end{cases}$$

Fitnessul mediu static (media fitnessului tuturor instanțelor schemelor) calculat pentru schemele care au doar primul bit definit este:

eval(1*…*) = ½ (3 instanțe din 4 au fitnessul egal cu 0, şi doar una dă 2)

eval(0*…*) = 1

În urma selecției, instanțe ale schemei 111*…* vor fi puternic selectate

în populație. După un număr de n generații fitnessul observat pentru schema 111*…* este aproximativ egal cu 2.

eval(1*…*, n) → **eval**(111*…*, n) ≈ 2.

Instanțe ale schemei 111*…* sunt însă instanțe şi pentru schema 1*…*. Acestea din urmă vor fi selectate aşadar de mult mai multe ori decât instanțele schemei 0*…* deşi au un fitness mediu static mai mic.

Pentru problema banditului cu două brațe cele două variabile aleatoare care descriu fiecare braț sunt independente (deciziile nu sunt influențate de jocurile anterioare), în timp ce în algoritmul genetic, brațele (schemele) interacționează: câştigul observat pentru schema 1*...* este puternic influențat de câştigul observat pentru scheme 111*...*. Aşadar, algoritmul genetic nu eşantionează schemele în mod independent pentru a estima adevăratul lor câştig.

În realitate algoritmul genetic nu simulează jocul banditului cu 3^m brațe, în care toate schemele au rolul brațelor, ci banditul cu 2^k brațe pentru fiecare schemă de ordin k. Algoritmul genetic alocă un număr exponențial de instanțe celei mai bune sub-scheme observate dintr-o schemă şi nu celei mai bune scheme viitoare. Strategia algoritmului genetic va fi (aproape) optimală dacă valorile fitness ale schemelor în poulație au o distribuție uniformă. Exemplu anterior al lui Grefenstette şi Baker este ilustrativ în acest sens.

În ceea ce priveşte evoluția banditului cu 2^k brațe, valoarea lui k creşte pe parcursul algoritmului genetic: o dată cu derularea generațiilor, se obțin scheme de ordin tot mai mare. Totuşi, la fiecare generație selecția introduce noi erori în modalitatea de eşantionare a schemelor, rezultatul fiind necorelarea fitnessului mediu static cu fitnessul mediu observat.

Strategia de alocare exponențială utilizată pentru problema banditului cu două brațe este potrivită problemelor care necesită adaptare şi în care performanța este măsurată on-line, precum problemele de control în timp real (controlul automat al maşinilor) şi probleme de învățare automată (de ex. navigarea într-un mediu necunoscut, prezicerea piețelor financiare). În astfel de probleme algoritmul genetic reuşeşte să ofere soluții aproximative satisfăcătoare, în timp scurt. Spre deosebire de *optimizare* (determinarea celei mai bune soluții), algoritmul genetic reuşeşte să maximizeze, într-un timp relativ scurt dat, cantitatea de informație legată de sub-spațiile cu şanse mari de a furniza soluții bune. Obținerea de soluții în acest mod poate fi denumită *satisfacere*. Pentru o reală optimizare, este necesară hibridizarea algoritmului genetic cu alte strategii (precum hill climbing) care să realizeze o căutare de tip gradient. Algoritmii genetici realizează o cătare globală, fiind

capabili să descopere în timp scurt regiuni promițătoare în spațiul de căutare, în timp ce algoritmii de tip hill climbing realizează o căutare locală în acele regiuni.

4.3 Înşelarea unui algoritm genetic

În algoritmii genetici selecția dirijează procedura de eşantionare către instanțe ale schemelor cu fitnessul estimat peste medie. Acuratețea acestei estimări creşte odată cu vârsta populației. Există totuşi contraexemple în care, scheme scurte de ordin redus şi cu fitness peste medie pot înşela algoritmul genetic cauzând convergența către puncte suboptimale (Bethke 1980). Un exemplu în acest sens este următorul: toate schemele în care biții definiți au valoarea 1, au valori mari ale fitnessului static; excepție face schema care are 1 pe primele două poziții şi pe ultima, 11…1; în locul acesteia din urmă, fitness ridicat obține schema 00…0. Funcțiile de acest gen au fost denumite *complet înşelătoare* (termenul înşelotor – în engleză *deceptive* – a fost introdus în 1986 de către Goldberg): scheme de ordin redus dau informații false cu privire la localizarea optimului. Exemplul este ilustrat în continuare: schemele S_1, S_2, S_3

S_1 = (11******)
S_2 = (*******1)
S_3 = (00*****0)

au fitnessul peste medie dar combinația schemelor S_1 şi S_2 dată de S_4

S_4 = (11*****1)

are fitness scăzut.

Dacă soluția optimă este şirul

s = (11111111)

algoritmul genetic va întâmpina dificultăți în a converge către aceasta deoarece prezintă tendința de a converge către puncte precum (00111110).

Există grade diferite de înşelare a unui algoritm genetic. Bethke a utilizat transformări Walsh (asemănătoare transformărilor Fourier) pentru a crea funcții fitness cu diferite grade de înşelare.

Înşelarea este un factor imortant atunci când algoritmii genetici sunt utilizați pentru optimizare (pentru determinarea optimului global). Dacă însă ei sunt utilizați pentru satisfacere (maximizarea câştigului), înşelarea poate fi neglijată

Problema ridicată de înşelare poate fi tratată prin utilizarea de

reprezentări alternative (în exemplul nostru o codificare în care primii doi biți şi ultimul să ocupe poziții adiacente), reprezentări care necesită însă cunoştințe apriori despre funcția obiectiv şi natura decepției. O altă soluție este utilizarea unui al treilea operator – inversia – care selectează două puncte în şir şi inversează ordinea biților.

4.4 Critica teoremei schemelor

Teorema schemelor nu explică în totalitate comportamentul algoritmilor genetici. Ea nu oferă o garanție pentru producerea convergenței într-un algoritm genetic care respectă setul de condiții specificat, dar explică cum se produce convergența atunci când are într-adevăr loc. Nu dă informații nici în ceea ce priveşte viteza de convergență.

Teorema schemelor lucrează numai cu scheme care apar în prima iterație a algoritmului, nu explică apariția de noi scheme pe parcursul algoritmului şi nici nu oferă estimări ale schemelor ce vor fi descoperite.

Teorema schemelor lucrează cu fitnessul observat al unei scheme, obținut din indivizii prezenți în populație; valoarea acestuia poate diferi însă de fitnessul static, obținut luând în calcul toți reprezentanții schemei.

Schemele sunt potrivite pentru blocurile de construcție rezultate în urma încrucişării cu un singur punct de tăiere. Alte structuri sunt însă necesare pentru a lucra cu alți operatori.

Predicțiile făcute pe termen îndelungat sunt nerealiste, evoluția populației este corect apreciată doar pentru un pas al algoritmului.

Capitolul 5
Funcțiile Royal Roads

Teorema schemelor a evidențiat efectele pozitive ale selecției; în ceea ce priveşte operatorii genetici a luat însă în considerare numai efectele negative/ destructive ale acestora.

Teorema schemelor a sugerat că drumul către soluția optimă într-un algoritm genetic este croit pornind de la scheme scurte, de ordin redus şi bine adaptate care, prin combinare, dau naştere unor scheme de ordin intermediar cu fitness ridicat; acestea, la rândul lor, dau naştere unor scheme cu valori şi mai ridicate ale fitnessului, procesul repetându-se până la convergență.

Pornind de la aspectele evidențiate de teorema schemelor, Stephanie Forrest, John Holland şi Melanie Mitchell au formulat în 1992 o funcție care să respecte ipoteza blocurilor de construcție şi care să ilustreze puterea constructivă a operatorului de încrucişare. Această funcție este construită utilizând o listă de scheme s_i şi o mulțime de coeficienți c_i asociați acestor scheme. Funcția R_1 este dată de următoarea sumă:

$$R_1(x) = \sum_i c_i \cdot \delta_i(x),$$

în care $\delta_i(x)$ = "**if** ($x \in s_i$) **then** 1 **else** 0" verifică satisfacerea schemei s_i de către soluția x.

Funcția R_1 are o structură sub formă de blocuri de construcție; ipoteza blocurilor de construcție ar trebui să contureze un drum "regal" către şirul optim pentru algoritmul genetic. Aceste funcții studiază de fapt procesarea şi recombinarea schemelor, testând abilitatea algoritmilor genetici de a propaga blocuri de construcție folositoare..

O funcție de acest gen este ilustrată în continuare; ea este construită cu ajutorul a 8 scheme de ordin egal cu 8; coeficienții c_i sunt egali cu ordinul schemelor:

s_1 = 11111111**********…**********, c1=8
s_2 = ********11111111********…**********, c2=8
s_3 = **********11111111…**********, c3=8
s_4 = **********…**********, c4=8
s_5 = **********…**********, c5=8
s_6 = **********…**********, c6=8
s_7 = **********…11111111********, c7=8
s_8 = **********…********11111111, c8=8.

Valoarea maximă a funcției este egală cu 64 şi se obține în punctual de optim:

s_{opt}= 11111111111111111111111…1111111111111111.

Pentru această funcție a fost proiectat un algoritm genetic standard cu următoarele valori ale parametrilor: pop_size = 128, pc = 0.7, pm = 0.005. Inițializarea populației s-a făcut aleatoriu.

Schema de selecție utilizată a fost σ-selecția, în care numărul estimat de copii ale unui individ este stabilit prin comparație între fitnessul individului şi fitnessul mediu raportat la deviația standard:

$1 + (f(x_i) - F / pop\_size) / \sigma$

Această schemă are drept rezultat realizarea a zero, una sau două copii ale unui individ în generația intermediară. În alegerea schemei de selecție s-a avut în vedere faptul că pentru funcția R_1 unii indivizi pot avea valori mult mai mari ale fitnessului decât alți indivizi; datorită numărului mic de copii pe care un individ le poate primi în generația intermediară sub această schemă de selecție, convergența este încetinită şi în acest fel evitată convergența prematură.

Algoritmul genetic astfel proiectat a fost comparat cu trei algoritmi, variații ale algoritmului Hillclimbing standard.

În urma experimentelor, algoritmul genetic a convers prematur către o soluție diferită de soluția optimă, un punct de optim local. Convergența prematură nu a fost neapărat rezultatul a prea puține generații, ci mai degrabă a faptului că algoritmul genetic s-a îndreptat către o țintă greşită.

Cele trei variante ale algoritmului Hill Climbing utilizat au fost: steepest ascent hill climbing (SAHC), next-ascent hill climbing (NAHC) şi random mutation hill climbing (RMHC).

SAHC caută printre vecinii soluției curente pe cael care oferă cea mai bună îmbunătățire a funcției fitness:

1. alege aleatoriu un şir;
2. generează toți vecinii la distanță Hamming 1;
3. vecinul care dă cea mai bună îmbunătățire a fitnessului devine soluție curentă
4. dacă nu se obține nici o îmbunătățire a fitnessului atunci mergi la pasul 1, altfel mergi la pasul 2
5. când este atins numărul maxim de evaluări ale funcției fitness, returnează cea mai bună soluție găsită.

NAHC ia în considerare primul vecin care oferă o îmbunătățire şi

continuă căutarea din acel loc:

1. alege aleatoriu un şir; setează j=1;
2. generează cel mult m vecini la distanță Hamming 1 începând cu poziția j;
3. dacă la poziția k se obține o îmbunătățire a fitnessului, noul şir devine soluție curentă, actualizează j = (k+1)%m şi mergi la pasul 2;
4. mergi la pasul 1;
5. când este atins numărul maxim de evaluări ale funcției fitness, returnează cea mai bună soluție găsită.

RMHC, spre deosebire de cele două variante de mai sus, utilizează selecția aleatoare a vecinilor:

1. generează aleatoriu un şir;
2. alege aleatoriu o locație pentru modificare;
3. dacă fitnessul soluției obținute este cel puțin la fel de bun, selecteaz-o drept soluție curentă;
4. mergi la pasul 2 până numărul maxim de evaluări este atins;
5. returnează soluția curentă.

Fiecare algoritm a fost rulat de 200 ori cu inițializări diferite a generatorului de numere aleatoare. Numărul maxim de evaluări a fost setat la 256.000. Algoritmilor li s-a permis să ruleze până a fost găsit optimul iar numărul de evaluări a funcției a fost înregistrat.

Rezultatele experimentale sunt afişate în tabelul următor.

În ceea ce priveşte algoritmii SAHC şi NAHC rezultatele sunt cele estimate: nici unul nu reuşeşte să găsească optimul cu un maxim de 256000 evaluări ale funcției în timp ce algoritmul genetic găseşte soluția optimă după un număr mediu de 61334 evaluări. Lucru neaşteptat, RMHC este de aproximativ 10 ori mai rapid decât algoritmul genetic atingând optimul după un număr mediu de numai 6179 evaluări ale funcției.

Numărul de evaluări pentru 200 rulări	GA (σ/sqrt(200))	SAHC	NAHC	RMHC (σ/sqrt(200))
Media	61.334 (2.304)	⊥ >256,000	⊥ >256,000	6,179 (186)
Mediana	54208	⊥ >256,000	⊥ >256,000	5,775

Deşi funcțiile Royal Roads s-au dorit a fi un mediu special creat pentru a furniza drumuri regale către convergență pentru algoritmii genetici, rezultatele experimentale au dovedit că o procedură aleatoare obține o performanță mai bună.

Rămâne deschisă aşadar întrebarea referitoare la condițiile în care un algoritm genetic depăşeşte alți algoritmi, precum hill climbing.

O analiză a algoritmului RMHC a fost realizată pentru a explica rezultatele bune obținute de acesta pe funcțiile Royal Roads. Funcția considerată pentru analiză utilizează N blocuri adiacente, fiecare cu o lungime de K biți (pentru problema de mai sus N=8, K=8). Timpul necesar găsirii unui al doilea bloc este mai mare dacât cel necesar determinării primului bloc deoarece sunt efectuate mutații şi în cadrul primului bloc. Numărul de mutații folositoare, efectuate în afara primului bloc este egal cu (K·N-K)/K·N.

Pentru a estima numărul de evaluări ale funcției, efectuate de către algoritm până la găsirea soluției optime, se estimează, din aproape în aproape, numărul de evaluări necesare găsirii primului bloc, celui de al doilea bloc, samd., până când toate cele N blocuri au fost determinate. Dacă utilizăm notația E(K, I) pentru a desemna numărul estimat de evaluări ale funcției pentru determinarea a I blocuri, obținem următorul şir de relații:

$$E(K,2) = E(K,1) + E(K,1) \cdot (K \cdot N / (K \cdot N - K)) = E(K,1) \cdot N / (N-1)$$
$$E(K,3) = E(K,2) \cdot N / (N-2)$$
$$\dots$$
$$E(K,N) = E(K,N-1) \cdot N / (N-(N-1))$$
$$E(K,N) = E(K,1) \cdot N \cdot (1+1/2+1/3+\dots+1/N)$$

Deşi ultima formulă estimează numărul de evaluări necesare găsirii soluției optime funcție de numărul de evaluări necesar găsirii primului bloc, în realitate, E(K,N) depinde de numărul de evaluări necesar determinării blocului cel mai nefavorabil.

Aproximând ultimul termen, obținem următoarea relație:

$$E(K,N) \approx E(K,1) \cdot N \cdot (\ln N + \gamma)$$

unde γ este constanta lui Euler.

Pentru a estima numărul de evaluări necesar găsirii primului bloc printr-un şir de mutații succesive, se realizează o analiză statistică cu ajutorul lanțurilor lui Markov. Rezultatul obținut estimează numărul de evaluări la 2^k pentru determinarea primului bloc. Pentru cazul particular considerat, valoarea estimată este E(8,1) = 301,2.

Numărul estimat de evaluări ale funcției efectuate de algoritmul RMHC

până la găsirea optimului este dat aşadar de următoarea relație:

$$E(K,N) \approx 2^K \cdot N \cdot (\ln N + \gamma)$$

iar valoarea obținută pentru cazul particular considerat este E(8,8) = 6.549.

În realitate, în urma execuției repetate a algoritmului RMHC, media pentru 200 de rulări a numărului de evaluări necesare până la obținerea soluției optime a fost 6.179 apropiat de cel estimat. Aşadar, rezultatele teoretice obținute pentru algoritmul RMHC estimează corect performanța acestuia în practică.

Au fost căutate motivele pentru care algoritmul genetic dă rezultate mai slabe comparativ cu algoritmul RMHC.

Principalul motiv analizat a fost denumit *hitchhicking* (autostopist): o dată găsit un bloc (o schemă), scheme care conțin 0 pe alte poziții decât blocurile descoperite poartă acelaşi fitness ridicat în populație. Aceasta încetineşte descoperirea de noi scheme pe alte poziții, în special a celor care sunt apropiate de pozițiile definite ale schemelor cu fitness ridicat (probabilitatea ca punctul de tăiere al operatorului de încrucişare să fie în aceste poziții este mai mică).

Fenomenul *hitchhicking*, denumit şi corelație falsă, a fost studiat de Belew (1992), Whitley (1991), Schaffer (1991), şa. Paralelismul implicit al algoritmului genetic este limitat prin restricționarea, la anumite poziții, a numărului de scheme prelevate; selecția acestora nu este independentă. Eficacitatea încrucişării este limitată de convergența către scheme cu fitness ridicat dar greşite în primele iterații. O schemă bună care nu e prezentă în prima generație va fi defavorizată în generațiile următoare din cauza schemelor vecine prezente deja în populație, mai ales dacă ambele blocuri vecine sunt deja găsite în scheme diferite sau în aceeaşi schemă.

Datorită paralelismului implicit şi a încrucişării, algoritmul genetic ar trebui să dea rezultate mai bune pe funcțiile Royal Roads decât algoritmul RMHC. Dar, ca şi în cazul RMHC (fiecare şir diferă de cel anterior printr-un singur bit) selecția nu este independentă nici în algoritmul genetic. De exemplu, selecțiile în hiperplanul s_3 nu sunt independente de cele din hiperplanele s_2 sau s_4.

Conform ipotezei statice a blocurilor de construcție, dacă în algoritmul genetic procedura de selecție ar fi independentă în fiecare hiperplan şi cea mai bună schemă din fiecare hiperplan ar fi selactată, atunci încrucişarea va

combina rapid cele mai bune scheme într-un singur şir. Plecând de la aceste premize, Mitchell, Holland şi Forrest (1994) au formulat un algoritm genetic idealizat.

Algoritmul genetic idealizat (IGA) nu are o populație, ci lucrează cu un singur şir la un moment dat. IGA are drept intrare schemele dorite şi funcționează conform pseudo-codului următor:

repeat {**generează** şiruri aleatoriu}**until** (şirul generat conține cel puțin o schemă dorită) ;
memorează acel şir;
While (**not** condiție_oprire) **do** {
generează şiruri aleatoriu (cu probabilitate uniformă);
oricând este descoperit un şir care conține una sau mai multe scheme nedescoperite încă, aplică **încrucişarea** între acesta şi cel memorat;
înlocuieşte şirul memorat cu descendentul care conține toate schemele dorite descoperite până atunci
}

Analizând pseudo-codul, următoarele proprietăți captează elementele esențiale ale unui algoritm genetic şi satisfac ipoteza statică a blocurilor de construcție sunt evidente:

Şirurile sunt selectate complet independent, ceea ce rezultă într-o selecție independentă a schemelor (hiperplanelor)

Selecția (deterministă) este condusă de memorarea schemelor dorite / găsite

Este utilizată încrucişarea.

Schemele dorite sunt de fapt necunoscute apriori.

IDA oferă o limită inferioară pentru timpul (numărul de evaluări) necesar algoritmilor genetici să găsească şirul optim.

Pentru a compara performanța algoritmului genetic idealizat cu cea a algoritmului RMHC, este estimat numărul de evaluări ale funcției până la găsirea şirului optim. Considerând funcția cu N blocuri (scheme), fiecare de lungime K, probabilitatea de a găsi o instanță a unei scheme H în mod aleatoriu este $p = 1/2^K$. Probabilitatea ca schema H să nu fie găsită o notăm cu q = 1-p. Probabilitatea ca H să fie găsită până la momentul t este $p_1(t) = 1-q^t$ iar probabilitatea de a găsi toate cele N scheme până la momentul t este $p_N(t) = (1 - q^t)^N$.

Pentru a estima însă timpul necesar găsirii tuturor celor N scheme este nevoie de probabilitatea $P_N(t)$ de a găsi ultima schemă exact la momentul t:

$\boldsymbol{P}_N(t) = \boldsymbol{P}_N(t) - \boldsymbol{P}_N(t\text{-}1) = (1 - q^t)^N - (1 - q^t\text{-}1)^N$

Pentru a estima timpul utilizând această probabilitate se sumează după toate momentele de timp:

$$E_N = \sum_{t=1}^{\infty} t \cdot \boldsymbol{P}_N(t) = \sum_{t=1}^{\infty} t \cdot ((1-q^t)^N - (1-q^{t-1})^N)$$

Utilizând teorema binomială, diferențele de mai sus pot fi extinse astfel:

$$((1-q^t)^N - (1-q^{t-1})^N) =$$
$$= [C_N^1(1/q-1)q^t] - [C_N^2(1/q^2-1)q^{2t}] + ... + (-1)^{N-1}[C_N^N(1/q^N-1)q^{Nt}]$$

Pentru a suma după t de la 1 la ∞ considerăm suma infinită a primului termen, suma infinită a celui de al doilea termen, etc:

$$C_N^1(1/q-1)\sum_{t=1}^{\infty} t \cdot q^t = C_N^1(1/q-1) \cdot q(q + 2q^2 + 3q^3 + ...) =$$
$$C_N^1(1/q-1) \cdot q \cdot \frac{d}{dq}(q + q^2 + ...) = C_N^1(1/q-1) \cdot q \cdot \frac{d}{dq}(\frac{q}{1-q}) = C_N^1 \frac{1}{1-q}$$

Suma celui de al n-lea termen este: $C_N^n \frac{1}{1-q^n}$.

Substituind 1-p cu q şi presupunând că valoarea lui K este suficient de mare pentru a putea face aproximarea $q^n = (1\text{-}p)^n \approx 1\text{-}\ n \cdot p$, obținem:

$$E_n \approx \frac{1}{p} \cdot (\frac{C_N^1}{1} - \frac{C_N^2}{2} + \frac{C_N^3}{3} - ... + (-1)^{n-1} \frac{C_N^N}{N})$$

Pentru valorile N=8 şi K=8 timpul estimat are valoarea $E_N \approx 696$. Experimentele effectuate de Mitchell dau exact această valoare pentru 200 de rulări ale algoritmului genetic idealizat (cu deviația standard 19,7).

Utilizând teorema binomială şi integrarea lui $(1+x)^N$ obținem:

$$\sum_{n=1}^{N} C_N^n \cdot \frac{x^n}{n} = \sum_{n=1}^{N} \frac{1}{n} \cdot ((1+x)^{n-1} - 1)$$

ceea ce duce la :

$$E_N \approx -\frac{1}{p}\sum_{n=1}^{N} C_N^n \cdot \frac{(-1)^n}{n} = \frac{1}{p}\sum_{n=1}^{N}\frac{1}{n} \approx \frac{1}{p}\cdot(\ln N + \gamma)$$

Pentru algoritmul genetic idealizat timpul estimat este aşadar

EN = O (2K · ln(N))

în timp ce pentru algoritmul RMHC timpul estimat este

E(K,N) = O (2K · N · ln (N)).

Algoritmul genetic idealizat este de N ori mai rapid decât RMHC în primul rând datorită paralelismului implicit perfect implementat: noi instanțe sunt date independent fiecărei scheme în algoritmul genetic în timp ce în RMHC fiecare şir nou dă oferă o instanță pentru doar o schemă. Selecția independentă ân IGA permite mai multor scheme să fie găsite într-un singur şir şi evită mutațiile nefolositoare, pe biți corecți.

Comparația făcută pentru IGA este valabilă considerând orice metodă de tip HC bazată pe mutația unui singur bit sau a unui număr redus de biți.

Algoritmul genetic idealizat lucrează în acest mod deoarece cunoaşte explicit care sunt schemele dorite. Algoritmul genetic standard nu dispune de această informație şi poate face doar estimări.

Unele caracteristici ale IGA, pot fi aproximate in algoritmul genetic standard.

Pentru o eşantionare independenta populația trebuie să fie suficient de mare, presiunea de selecție trebuie să fie suficient de scazuta (fitnessul relativ al schemelor dorite care nu se suprapun sa aiba valoare mica) iar rata mutației suficient de mare pentru a asigura ca nici un locus să nu fie fixat pe o singură valoare în fiecare şir din populatie sau intr-un numar foarte mare de siruri.

Pentru a inmagazina schemele dorite, presiunea de selecție trebuie sa fie suficient de mare ca sa permită supraviețuirea acestora.

Rata încrucişării trebuie să aibă o valoare suficient de mare ca sa permita recombinarea rapida a schemelor dorite.

Pentru ca algoritmul genetic să fie într-adevăr rapid comparativ cu metoda RMHC lungimea şirului trebuie sa fie suficient de mare ca factorul N sa fie semnificativ.

Nu toate aceste mecanisme sunt insa compatibile, unele au caracter opus, iar implementarea lor trebuie sa obțină echilibrul.

Capitolul 6

Implementarea algoritmilor genetici

După cum s-a văzut în capitolele anterioare, pentru rezolvarea unei probleme pot fi proiectați o mulțime largă de algoritmi genetici. Apar astfel dificultăți în alegerea celui mai potrivit algoritm genetic, teoria oferind prea puține informații în acest sens. Algoritmii genetici standard utilizează reprezentări pe biți, scheme de selecție bazate pe proporțiile fitnessului şi operatori simpli; acestea însă nu sunt cea mai bună alegere pentru orice caz particular. Implementarea unui algoritm genetic trebuie să țină cont de caracteristicile specifice problemei de rezolvat şi să exploateze aceste cunoştințe în special prin intermediul operatorilor.

În practică, algoritmii genetici s-au dovedit a fi o unealtă puternică însă au fost cazuri în care au înregistrat şi eşecuri, fiind depăşiți de alte metode. Nu există reguli stricte pentru a identifica situațiile în care algoritmii genetici sunt cea mai potrivită opțiune pentru a rezolva o problemă; intuitiv însă, un algoritm genetic are şanse mari de a depăşi alte metode dacă problema prezintă următoarele proprietăți:

- spațiul de căutare este suficient de mare încât o căutare exhaustivă ar fi practic imposibilă
- funcția de optimizat este
  - „ne-standard" sau cu zgomot, nefiind potrivită abordarea cu metode bazate pe un singur candidat (ex. Hillclimbing);
  - multi-modală, în cazul problemelor uni-modale fiind utilizate cu succes metode de tip gradient ascendent;
  - mai puțin înțeleasă; dacă spațiul problemei este foarte bine înțeles, metodele de căutare care utilizează euristici specifice domeniului pot fi uşor proiectate pentru a depăşi metodele cu scop general precum algoritmii genetici;
- nu este necesară determinarea optimului global ci găsirea unei soluții suficient de bune într-un timp relativ scurt.

Regulile de mai sus nu sunt suficiente pentru a prezice performanța unui algoritm genetic relativ la alte metode. Aceasta depinde în mare măsură de detaliile implementării algoritmului genetic, precum modul de codificare a soluțiilor candidat, operatorii, parametrii.

6.1 Reprezentarea

Reprezentarea soluțiilor candidat este considerată a fi factorul central de care depinde succesul sau eşecul algoritmilor genetici.

Reprezentarea cea mai des întâlnită este cea din algoritmul genetic clasic: şiruri de biți cu dimensiune şi ordine fixe. Principalul avantaj al unei astfel de codificări este dat de existența unui fundament teoretic care explică modul în care are loc căutarea până la convergență. Un alt avantaj, evidențiat de Holland este gradul ridicat de paralelism implicit în cadrul algoritmului genetic: pentru codificări relativ lungi binare numărul de scheme reprezentate este mai mare decât pentru codificări peste un număr mare de caractere dar de dimensiune mai mică. De asemenea, pentru astfel de codificări există numeroase studii care au evidențiat anumite euristici pentru setări ale.

Din păcate, de multe ori astfel de codificări nu sunt naturale, fiind greu de utilizat pentru multe probleme.

Extensii ale codificării binare au fost propuse precum codurile gray sau codificările binare diploide.

Naturale şi uşor de utilizat sunt codificările care utilizează un număr mai mare de caractere sau valori reale (un exemplu îl constituie reprezentarea cu numere reale pentru antrenarea rețelelor neuronale). Contrar argumentelor lui Holland, în practică pentru unele probleme se obține o performanță mai bună dacă se lucrează cu alfabete de dimensiune mai mare.

John Koza a introdus codificarea sub formă de arbore (soluțiile candidat sunt programe de calculator) punând bazele Programării Genetice. Spațiul căutării este nelimitat, în principiu dimensiunea unui arbore putând creşte foarte mult prin operații de mutație şi încrucişare. Un dezavantaj al acestei reprezentări îl constituie dificultatea de simplificare şi interpretare a arborilor de dimensiune mare.

Unele probleme se pretează la reprezentări multidimensionale (precum clasificarea nesupervizată – clustering), altele necesită reprezentări neomogene (de ex. pentru problema orarului codificarea instrucțiunilor care duc la construirea orarului). Decodificarea necesită uneori utilizarea unor euristici specifice problemei: de ex. pentru clasificarea nesupervizată (clustering) în determinarea claselor reprezentate de o soluțe candidat ce codifică doar centrele claselor se utilizează metoda celui mai apropiat vecin.

Nu există reguli stricte care să ne ajute la alegerea celei mai bune reprezentări pentru o problemă. Cu o bogată experiență în aplicarea algoritmilor genetici în lumea reală, Lawrence Davis sugerează utilizarea

codificării naturale a problemei şi proiectarea celorlalte elemente ale algoritmului genetic în funcție de aceasta.

Paradoxal însă, algoritmii genetici sunt potriviți pentru a rezolva probleme mai puțin înțelese; cum poate fi atunci cunoscută codificarea naturală apriori? Unele poziții sunt mai importante decât altele în schemele folositoare. Apare astfel fenomenul denumit *legare* (linkage) în care o mulțime de biți acționează ca alele coadaptate tinzând să se transmită ca un grup. Problema este de a reprezenta compact un grup de gene în raport cu o schemă, astfel încât aceasta să nu fie alterată de încrucişare.

Imposibilitatea cunoaşterii celei mai bune reprezentări a dus la ideea adaptării codificării pe parcursul algoritmului.

O primă schemă de adaptare ia în considerare lungimea cromozomului. Este cazul programării genetice în care lungimea cromozomului se modifică prin aplicarea operatorilor. O astfel de codificare cu lungime variabilă este necesară în rezolvarea problemelor de clasificare nesupervizată când numărul de clase este necunoscut apriori. De asemenea, în probleme de învățare a celei mai bune strategii (precum dilema prizonierului) este interesantă evoluția strategiilor pentru lungimi diferite a istoricului.

O altă modalitate de adaptare a reprezentării face uz la operatorii genetici: utilizarea unui nou operator - inversiunea şi identificarea punctelor de tăiere pentru operatorul de încrucişare astfel încât descendenții să fie mai bine adaptați.

Alături de încrucişare şi mutație, **inversiunea** este de multe ori considerată un operator genetic de bază. Acest operator a fost introdus de Holland (1975) pentru a modifica linkage-ul dintre biții unui cromozom de lungime fixă în aşa fel încât biți cu interacțiuni neliniare puternice să devină apropiați pe cromozom. Holland a accentuat ideea că un linkage corect este esențial pentru funcționalitatea operatorului de încrucişare.

Inversiunea este un operator de reordonare inspirat din genetica reală. Spre deosebire de algoritmii genetici, în genetica reală funcția unei gene este de cele mai multe ori independentă de poziția sa în cromozom iar inversarea unei porțiuni de cromozom are un efect minor asupra semanticii cromozomului inițial.

Inversiunea are drept efect modificarea ordinii biților într-un cromozom dar nu şi a semanticii cromozomului. Din acest motiv este necesară o modificare în reprezentarea cromozomului astfel încât codificarea să fie

independentă de poziție. De exemplu, se poate utiliza codificarea cu perechi în care primul număr este eticheta, indexând bitul, iar al doilea număr este valoarea bitului:

(00011010001)

→ ((1,0)(2,0)(3,0)(4,1)(5,1)(6,0)(7,1)(8,0)(9,0)(10,0)(11,1))

Inversiunea acționează asupra unei ordini aşa cum mutația acționează asupra biților. Tipic, inversiunea este implementată prin răsturnarea unui segment din cromozom ales aleatoriu. Dacă punctele de inversiune generate aleatoriu au valorile 3 şi 7, în urma aplicării inversiunii cromozomul de mai sus devine:

((1,0) (2,0) (3,0)|(7,1) (6,0) (5,1) (4,1)|(8,0) (9,0) (10,0) (11,1))

Prin modificarea linkage-ului cu ajutorul inversiunii se modifică probabilitățile de supraviețuire a schemelor la încrucişare. De exemplu, scheme precum (10*********01) pot supraviețui la încrucişare dacă inversiuni utile găsesc blocul de construcție ((1,1) (2,0) (13,1) (12,0) (11,*)…(3,*)). Inversiunea este aşadar o modalitate de a evita fenomenul înşelării în algoritmii genetici prin determinarea unui linkage corect.

Utilizarea acestui operator prezintă însă şi unele dezavantaje: datorită utilizării unor structuri de tip permutare, încrucişarea clasică cu un punct de tăiere poate duce la descendenți cu o structură lipsită de sens în care o genă apare de mai multe ori iar unele nu apar deloc. O soluție mai puțin agreată este restricționarea aplicării încrucişării doar pentru indivizii care au aceeaşi permutare a pozițiilor biților. O altă abordare este stăpân/sclav: un părinte este ales drept stăpân iar celălalt este reordonat temporar la aceeaşi permutare ca a stăpânului. După efectuarea încrucişării cel de al doilea cromozom (sclavul) este adus la ordonarea originală.

Rezultatele empirice obținute prin aplicarea acestui operator au evidențiat îmbunătățiri minore în algoritmul genetic. Nu există nici o dovadă clară (experimente sistematice sau rezultate teoretice) a beneficiilor aduse de inversiune. În plus, trebuie luată în calcul necesitatea unor resurse suplimentare de spațiu şi timp.

Schaffer şi Morishima introduc o abordare duală inversiunii: în locul determinării unei cele mai bune ordini, ei caută ***locurile fierbinți*** în care este permisă încrucişarea. În acest mod, nu ordinea biților în şir evoluează ci pozițiile unde poate fi aplicat operatorul de încrucişare. Astfel, în această abordare, fiecare cromozom are ataşat un vector de aceeaşi lungime în care

valoarea 1 se găseşte numai pe poziţiile în care este admisă încrucişarea. În exemplul următor, poziţiile marcate cu ! sunt cele în care este permisă ruperea cromozomului la încrucişare (încrucişarea este cea cu mai multe puncte de tăiere):

părinți: (1 0 0 1! 1 1 1! 1)
(0 0 0 0 0 0! 0 0)

descendenți: (1 0 0 1! 0 0! 1! 0)
(0 0 0 0 1 1 0 1)

Mutaţia acţionează atât asupra cromozomului cât şi asupra vectorilor ataşaţi.

Deşi numai soluţiile candidat sunt utilizate pentru evaluare, prin selecţie, încrucişare şi mutaţie evoluează nu numai acestea ci şi vectorii de încrucişare ataşaţi.

Ca şi în cazul inversiunii, îmbunătăţirile observate pe un set redus de funcţii nu au fost mari. Metoda nu a fost analizată pe o gamă mai largă de probleme.

Algoritmii genetici *messy* (dezordonaţi) au fost creaţi de Goldberg, Deb şi Korb în 1989 pentru a îmbunătăţi performanţa algoritmilor genetici în optimizarea funcţiilor. Ideea generală a fost motivată tot de evoluţia naturală: forme de viaţă simple au dat naştere la forme de viaţă complexe; genomul uman cu un număr de aproximativ $5.9*10^9$ perechi de nucleotide a avut ca precursori reprezentări mai simple.

Algoritmii genetici messy utilizează astfel reprezentări incomplete sau chiar contradictorii şi introduc noi operatori. Scopul este de a construi incremental şiruri cu fitness ridicat din blocuri de construcţie mai scurte. Reprezentarea soluţiilor candidat se face sub forma unor şiruri de biţi la care se ataşează şi locusul. Nu toate reprezentările au toate locusurile specificate (subspecificare) iar unele reprezentările pot avea mai multe valori specificate pentru acelaşi locus (supraspecificare). De exemplu, şirul {(1,0),(2,0),(4,1),(4,0)} nu are nici o valoare specificată pentru locusul 3 şi două valori pentru locusul 4.

Problema apare la evalurea unui astfel de cromozom. În cazul supraspecificării evaluarea ia în calcul valorile pentru gene de la stânga la dreapta. Cromozomul dat ca exemplu mai sus corespunde astfel schemei 00*1. Evaluarea unui cromozom (subspecificat) se reduce aşadar la evaluarea unei scheme. Două modalităţi au fost propuse pentru a evalua

cromozomii (schemele) în algoritmul messy. Prima variantă calculează fitnessul static al schemei corespunzătoare prin generarea aleatoare de cromozomi corepunzători schemei şi evaluarea acestora pentru a obține o medie. A doua modalitate utilizează un optim local determinat cu ajutorul unei euristici şi completarea schemei cu valori din această soluție.

Algoritmul genetic se abate de la schema generală a algoritmului genetic clasic prin utilizarea a două etape distincte: faza pimordială care are ca scop principal explorarea spațiului de căutare şi, şi faza de juxtapunere în care se efectuează exploatarea schemelor determinate prin combinarea lor.

În prima fază sunt determinate scheme relativ scurte dar promițătoare prin execuția următorilor paşi:

se ghiceşte valoarea celui mai mic ordin k astfel încât să se obțină scheme folositoare;

se enumeră toate schemele de ordin k; de exemplu, pentru un cromozom de lungime m=8 şi o valoare k=3 fixată schemele sunt:

{(1,0),(2,0),(3,0)};
{(1,0),(2,0),(3,1)};
…
{(1,1),(2,1),(3,1)};
{(1,0),(2,0),(4,0)};
{(1,0),(2,0),(4,1)};
…
{(6,1),(7,1),(8,1)}

se aplică selecția în modul următor: se crează un număr de copii ale cromozomilor proporțional cu valoarea fitnssului şi se şterge jumătate din populație la intervale regulate.

După un număr fixat de generații faza primordială ia sfârşit şi începe faza de juxtapunere. Dimensiunea populației rămâne fixă şi evoluția continuă cu ajutorul selecției şi a doi operatori de juxtapunere: tăierea şi îmbinarea. Tăierea este un operator unar care acționează prin divizarea unui cromozom messy părinte dând naştere la doi cromozomi messy descendenți. Îmbinarea este un operator binar care are drept rezultat alipirea a doi cromozomi părinți.

Tăierea:

$$\{(1,0),(2,0),(4,1),(4,0),(3,0)\} \rightarrow \begin{cases} \{(1,0),(2,0),(4,1)\} \\ \{(4,0),(3,0\} \end{cases}$$

Îmbinarea:

{(6,0),(3,1),(4,1),(3,0)}
{(2,1),(1,1)}
} {(6,0),(3,1),(4,1),(3,0), (2,1),(1,1)}

Experimente au fost efectuate pe o funcție fitness înşelătoare construită peste m=30 biți divizați în 10 segmente de lungime 3 care sumează scorul acestor segmente. Cel mai mare scor l-a primit segmentul 111; segmentul 000 a primit scorul imediat următor:

000→28; 001→26; 010→22; 011→0;
100→14; 101→0; 110→0; 111→30

Principala problemă în rezolvarea unor astfel de probleme cu algoritmii genetici messy îl constituie alegerea parametrului *k* (o valoare minimă a ordinului pentru a obține scheme folositoare) fără a dispune de cunoştințe apriori despre funcția de optimizat.

Odată fixată valoarea parametrului *k*, trebuiesc generate $2^k C_m^k$ scheme, ceea ce rezultă într-o explozie combinatorială. Dimensiunea populației creşte exponențial cu *k* iar problemele de dimensiune reală devin intractabile. Soluția este inițializarea complet probabilistă, în care sunt creați cromozomi de lungime cuprinsă între k şi m; paralelismul implicit va furniza multe scheme de ordin k pentru un şir.

6.2 Mecanisme de selecție

În pasul de selecție al algoritmilor genetici sunt create copii ale cromozomilor care participă la reproducere pentru a crea descendenți. Scopul selecției este de a alege pentru supraviețuire cei mai adaptați indivizi din populație cu speranța că descendenții acestora vor avea un fitness mai ridicat. În combinație cu variații ale operatorilor de încrucişare şi mutație, selecția trebuie să păstreze un echilibru între explorare şi exploatare. O presiune de selecție ridicată duce la crearea unei diversități scăzute în populația creată din indivizi bine adaptați dar suboptimali, ceea ce duce la o limitare a modificărilor şi implicit a progresului. Pe de altă parte, o presiune de selecție scăzută încetineşte evoluția. În continuare sunt prezentate cele mai întâlnite scheme de selecție în literatura de specialitate.

6.2.1. Roata norocului

Această este schema de selecție introdusă de Holland în algoritmul

genetic original. Numărul estimat de copii pe care le primeşte un individ este proporțional cu fitnessul său împărțit la fitnessul mediu al populației. Procedura de selecție a fost descrisă detaliat în capitolul (*Algoritmii genetici în optimizarea numerică).* Pentru populații relativ mici utilizate în algoritmul genetic, numărul estimat de descendenți pe care îl primeşte un cromozom este însă diferit de cel real. În plus, cu o probabilitate diferită de 0, toți descendenții pot fi alocați celui mai slab adaptat cromozom.

6.2.2. Selecție universală stocastică

James Baker propune o nouă metodă de selecție pentru a preveni variația mare de la valorile estimate. În această schemă, ruleta este învârtită o singură dată, nu de un număr de ori egal cu dimensiunea populației pop_size, dar cu pop_size indicatori egal spațiați. Următorul pseudo-cod descrie procedura de selecție:

```
ptr = Rand();
for (sum = i = 0; i < pop_size; i++)
  for (sum += Expected_Value(i,t); sum > ptr; ptr++)
            Select(i);
```

unde Expected_Value(i,t) este valoarea estimată a individului *i* la momentul *t*.

În această schemă, fiecare individ *i* este selectat la genarația *t* de cel puțin $\lfloor$Expected_Value(i,t)$\rfloor$ ori şi de cel mult $\lceil$Expected_Value(i,t)$\rceil$ ori.

Atât roata norocului cât şi selecția universală stocastică favorizează selecția unui număr mic de indivizi pentru reproducere în fazele de început ale algoritmului ceea ce favorizează explaotarea în detrimentul explorării. Acest fenomen duce la convergența prematură a algoritmului genetic.

6.2.3. Scalare sigma

În schemele anterioare rata evoluției depinde de variația fitnessului în populație. După cum s-a arătat, o variație prea mare duce convergență prematură. La capătul opus, o variație mică a fitnessului între indivizii din populație, duc la un comportament aproape aleatoriu al algoritmului.

Pornind de la aceste observații, Forrest (1985) a creat o schemă de selecție mai puțin influențată de varianță prin limitarea num[rului de copii pe care le poate primi un cromozom. Valoarea estimată Expected_Value (i,t) a unui individ *i* în generația *t* este funcție de valoarea fitnessului său f(i), media populației f_med(t) şi deviația standard a populației σ:

E_V(i,t) **= if** (σ(t) $\neq$ 0)

then (1+(f(i)-f_med(t))/2σ(t)) **else**

1.0

Expected_Value(i,t) **= if** (E_V(i,t)>=0)

then E_V(i,t) **else**

const (ex. const = 0.1).

6.2.4. Elitism

Elitismul, intordus de Ken DeJong (1975) este o schemă adițională oricărui mecanism de selecție al cărei scop este de a reține cei mai buni k indivizi la fiecare generație; astfel de idivizi ar putea altfel dispărea din cauză că nu au fost selecatți pentru supraviețuire sau fiindcă au fost distruşi prin aplicarea operatorilor. Studii experimentale au dovedit ca de multe ori elitismul îmbunătățeşte semnificativ performanța algortimului genetic.

6.2.5. Selecție Boltzmann

Spre deosebire de selecția sigma în care presiunea de selecție este constantă pe parcursul algoritmului, în selecția Boltzmann presiunea de selecție variază continuu funcție de un parametru T numit temperatură. Valori mari ale temperaturii la începutul algoritmului determină o presiune de selecție scăzută. Pe parcursul algoritmului temperatura scade, ceea ce determină o creştere gradată a presiunii de selecție. Asfel, nivelul de diversitate scade pe parcursul algoritmului; la începutul algoritmului este încurajată explorarea iar spre final spațiul căutării este limitat încurajând exploatarea.

În implementarea tipică, fiecărui individ *i* îi este asignată o valoare estimată:

$Exp\_Val(i,t) = e^{f(i)/T} / \langle\langle e^{f(i)/T} \rangle\rangle_t$,

unde $\langle\langle \cdot \rangle\rangle_t$ reprezintă media peste generația t. Pe măsură ce temperatura T descreşte, diferența pentru valorile Exp_Val(i,t) creşte între fitness-uri scăzute şi ridicate ceea ce duce la o creştere a presiunii de selecție.

6.2.6. Selecție bazată pe rang

Pentru a preveni convergența prematură ce apare la utilizarea schemelor de selecție proporțională cu fitnessul, Baker a intodus în 1985 o nouă schemă în care indivizilor li se atribuie un rang în funcție de fitness.

Valorile estimate depind în această schemă de rang şi nu de valorile fitness absolute. Presiunea de selecție este în acest mod scăzută în cazul în care varianța fitnessului este mare şi este crescută dacă varianța fitnessului este mică.

Baker a utilizat o metodă liniară pentru a atribui fiecărui individ un rang: indivizii sunt ordonați crescător în funcție de valoarea fitnessului iar fiecare individ primeşte drept rang poziția sa în şir. Primul individ din şir va fi etichetat aşadar cu rangul 1 iar ultimul cu rangul *pop_size*. Pentru rangul pop_size se alege o valoare estimată $2 \geq Max \geq 0$. Pentru rangul 1 valoarea estimată este $Min = 2 - Max$. Pentru orice individ *i* cu rangul rank(i,t) valoarea estimată se obține astfel:

$$Exp\_Val(i,t) = Min + (Max-Min)(rank(i,t)-1)/(pop\_size-1)$$

Valorile *Min* şi *Max* au fost astfel alese pentru a satisface condiția ca suma valorilor estimate a tuturor indivizilor din populație la momentul *t* să fie egală cu dimensiunea populației: $\sum Exp\_Val(i,t) = pop\_size$.

Selecția bazată pe rang poate fi implementată utilizând un mecanism asemănător celui utilizat în roata norocului.

O altă modalitate de a implementa selecția bazată pe rang introduce un parametru numeric *q* cu ajutorul căruia, fiecărui rang *i* i se ataşează o probabilitate de selecție a cromozomului corespunzător $prob(i) = q(1-q)^{i-1}$. În această abordare rangul 1 corespunde celui mai bun cromozom. De exemplu, pentru o populație de 50 de indivizi şi o valoare a parametrului q=0,04, se obțin următoarele valori ale probabilităților de selecție:

prob(1)=0.04; prob(2)=0.0384; prob(3)=0.036864; etc.

Selecția parametrului *q* se face astfel încât suma probabilităților de selecție a tuturor rangurilor să fie aproximativ egală cu 1.

$$\sum_{i=1}^{pop\_size} \text{prob}(i) = \sum_{i=1}^{pop\_size} q \cdot (1-q)^i \approx 1.$$

6.2.7. Selecție turneu

Selecția turneu este cea mai eficientă din punct de vedere al complexității timp. Din punct de vedere al presiunii de selecție se aseamănă selecției bazată pe rang.

În această abordare, doi indivizi sunt selectați la întâmplare din populație. Un număr $r \in [0;1]$ generat aleatoriu este comparat cu un

parametru *k* (de ex. k=0,9) al schemei ales în prealabil. Dacă r<k atunci este selectat pentru supraviețuire cel mai adaptat individ, altfel este ales cel mai puțin adaptat. Cei doi indivizi nu sunt eliminați din populația originală, putând astfel participa mai departe la selecție.

O analiză a acestei metode a fost efectuată de Goldberg şi Deb (1991).

6.2.8. Selecție stare stabilă (steady-state)

Majoritatea algoritmilor genetici descrişi în literatură sunt de tip generațional: noua populație este formată într-o iterație a algoritmului din descendenți rezultați în urma aplicării operatorilor genetici asupra părinților selectați din populația anterioară. Nici unul, puțini sau mai mulți părinți pot supraviețui nemodificați. De exemplu, schema de selecție elitistă prezentată mai sus păstrează o proporție din populație nemodificată pentru iterația următoare.

Proporția indivizilor noi din noua generație poartă numele de decalaj (gap) generațional (DeJong). În selecția stare stabilă, un număr mic de indivizi sunt înlocuiți în fiecare generație. De obicei sunt înlocuiți indivizii cei mai slabi cu descendenți rezultați în urma încrucişării şi mutației a celor mai buni indivizi. Acest tip de selecție este util în evoluția sistemelor bazate pe reguli în care învățarea incrementală este importantă (*machine classifier systems* - Holland 1986).

Selecția de tip stare stabilă a fost analizată de DeJong şi Sarma (1993).

Scheme de selecție – comparații

Selecția proporțională cu fitnessul este utilizată în mod tradițional, fiind propusă inițial de Holland şi utilizată în teorema schemelor.

Mecanisme de selecție alternative au arătat însă o îmbunătățire a convergenței în multe cazuri, realizând o mai bună echilibrare a explorării şi exploatării.

Metodele de selecție proporțională cu fitnessul necesită două iterări prin populație: una pentru a calcula valoarea medie a fitnessului şi una pentru a calcula valorile estimate.

Selecția bazată pe rang necesită o sortare a populației în funcție de valorile fitnessului, sortare care este consumatoare de timp.

Selecția turneu este mai eficientă din punct de vedere computațional şi se pretează paralelizării.

Din punct de vedere al dinamicii câmpurilor de probabilitate selecțiile

clasice sunt selecții dinamice, valoarea estimată a oricărui cromozom variind de-a lungul generațiilor; selecția bazată pe rang este o selecție de tip static, valorile estimte fiind fixe de-a lungul algoritmului.

O caracteristică importantă a schemelor de selecție este caracterul extinctiv al acestora. Probabilități de supraviețuire egale cu 0 determină eliminarea unor indivizi din populație: atunci când sunt eliminați cei mai buni cromozomi schema are caracter stânga-extinctiv iar daca sunt eliminați cei mai slabi cromozomi, dreapta-extinctiv. Schemele ne-extinctive folosesc numai probabilități diferite de 0, astfel încât oricărui individ i se dă şanse de supraviețuire.

Schemele care încurajează supraviețuirea celor mai buni indivizi sunt scheme elitiste.

6.3 Modelul insulelor

O modificare menită să îmbunătățească procesul de căutare în algoritmii genetici este împărțirea populației în grupuri de cromozomi care evoluează separat: selecția şi operatorii sunt aplicați numai în interiorul fiecărui astfel de grup din populație. Din timp în timp, cromozomi migrează dintr-un grup în altul cu o anumită probabilitate.

Avantajul unui astfel de model este dat de utilizarea mai multor istorii ale evoluției într-o singură rulare.

Bibliografie

- Z. Michalewicz – “Genetic Algorithms + Data Structures = Evolution Programs”, Springer Verlag, 1996.
- M. Mitchell – “An Introduction to Genetic Algorithms”, MIT Press, 1997.
- Lance D. Chambers – “Practical Handbook of Genetic Algorithms”, 1999,
- Kenneth A. De Jong - “Evolutionary Computation. A Unified Approach”, MIT Press, 2006.