Raport tema nr. 2

1. Descrierea problemei

Date fiind niste functii matematice, scrieti un algoritm care sa gaseasca punctul de minim global al acestor functii. Deoarece nu exista un algoritm determinist ce poate rezolva o astfel de cerinta, trebuie sa dezvoltam o modalitate de a gasi aceasta solutie intr-un timp acceptabil. Gasirea punctului de minim pentru o astfel de functie printr-o cautare exhaustiva poate duce la o complexitate timp de ordinul anilor, ceea ce nu ne avantajeaza.

Asadar, recurgem la o metoda euristica sustinuta de ideea: decat ceva perfect in 10 ani, mai bine ceva acceptabil in cateva secunde. Vom implementa astfel niste algoritmi ce imbunatatesc pe parcurs o solutie initiala, „evoluand" dupa niste reguli bine definite.

2. Algoritmul utilizat

Pentru rezolvarea acestei probleme, vom implementa un algoritm genetic. Acesta are la baza o strategie evolutiva, plecand de la o populatie initiala de cromozomi care, pe masura ce avanseaza, **evolueaza** cu ajutorul unor **mutatii**. Doar cei mai puternici indivizi (cromozomi) vor supravietui timp de mai multe **generatii**.

Terminologie

O **populatie** este formata dintr-o multitudine de cromozomi. O **generatie** este reprezentata de o populatie la un anumit moment.

In contextul algoritmului, un **cromozom** este o multime de n-dimensiuni pe care functiile matematice sunt testate. Un cromozom este format din mai multe **gene,** iar o gena reprezinta de fapt un bit in programul nostru. Fiecare dimensiune (variabila a functiei) are o lungime de reprezentare a ei in acest cromozom, iar toate dimensiunile puse cap la cap formeaza cromozomul.

Pe parcursul programului, aceasta populatie evolueaza si se cromozomi din ce in ce mai buni, in sensul ca functiile matematice, evaluate pe acesti cromozomi, vor da valori din ce in ce mai mici.

In primul rand, sunt **selectati** cromozomii ce vor forma generatia viitoare. Acest lucru se realizeaza printr-o selectie de tipul „Roata Norocului". Apoi, cromozomii sunt supusi **incrucisarii** si **mutatiei.**

Pseudocod:

Genereaza o populatie initiala, random

Cat timp populatia evolueaza

1. Evalueaza populatia curenta
2. Selecteaza cromozomii pentru urmatoarea generatie (Roata Norocului)
3. Incruciseaza cromozomii
4. Supune cromozomii mutatiei

Returneaza cea mai buna solutie gasita.

Detalii de implementare

Populatia va fi reprezentata in memorie printr-o matrice care retine mai multi cromozomi, fiecare cromozom fiind un sir de biti.

```
bool chromosomes[DMAX][DMAX * sizeof(int)];
```

In functie de un **discreteFactor**, pentru reprezentarea binara a fiecarei dimensiuni vor fi alocati un anumit numar de biti pentru a putea retine si acea precizie.

Pentru fiecare dimensiune, calculam numarul de biti necesar reprezentarii astfel:

```
for (lg = 0; (acceptedVals[1] - acceptedVals[0]) * pow(10, discreteFactor) > pow(2, lg) - 1;)
	++lg;

lgReprez[pos] = lg;
```

Aici, acceptedVals[0, 1] retine valoarea minima, respectiv maxima pe care o poate lua valoarea dimensiunii, iar discreteFactor ne spune ce precizie sa avem la calcul. lgReprez[i] = cati biti sunt folositi pentru reprezentarea celei de-a i-a dimensiune

Decodificarea numerelor se face astfel:

```
for (int i = 0; i < nrDims; ++i) {
	aux = BinaryToInt(chromosome + sumLgReprez, lgReprez[i]);
	doubleVals[i] = acceptedVals[i][0] + aux * (acceptedVals[i][1] - acceptedVals[i][0]) /
(pow(2, lgReprez[i]) - 1);

	sumLgReprez += lgReprez[i];
}
```

Fiecare cromozom este evaluat in functie de **fitness-ul** lui (cat de bun este). In cazul functiei Rastrigin, fitness-ul este 1/F + E, unde F este valoarea functiei si E ceva foarte mic, sa evitam impartirea la 0. Pentru functia schwefel, fitness-ul este abs (1(F + C)), unde C este o constanta foarte mare.

```
if (testFunction == &Schwefel7)
	rez.fitness = abs(1 / (rez.functionValue + C));
else
	rez.fitness = (1 / (rez.functionValue + EPSILON));
```

Selectia se realizeaza prin Roata Norocului. Pentru aceasta, calculam fitness-ul fiecarui cromozom si suma totala a acestor fitness-uri, S. Calculam apoi probabilitatile individuale si cumulate astfel:

```
for (int i = 0; i < popSize; ++i)
	prob[i] = chromosomeFitness[i] / sumFitness;

probCumulated[0] = 0;
for (int i = 1; i < popSize; ++i)
	probCumulated[i] = probCumulated[i - 1] + prob[i - 1];
```

Evident, probCumulated[0] = 0 si probCumulated[n+1] = 1, unde n este numarul total de cromozomi, adica marimea populatiei.

Apoi, timp de n iteratii, generam un numar random intre (0, 1). Va intra in generatia urmatoare acel cromozom j pentru care probCumulated[j] <= randomNumber <= probCumulated[j+1].

```
for (int i = 0; i < popSize; ++i) {
	randomNr = RandomDouble(0, 1);
	for (int j = 0; j < popSize; ++j)
		if (probCumulated[j] < randomNr && randomNr <= probCumulated[j + 1])
			chosen[i] = j;
}
```

Construim **urmatoarea generatie:**

```
for (int i = 0; i < popSize; ++i)
	Copy(auxChromosomes[i], chromosomes[i], sumLgReprez);

int i, aux = popSize;
popSize = 0;
for (i = 0; i < aux; ++i)
	Copy(chromosomes[popSize++], auxChromosomes[chosen[i]], sumLgReprez);
```

Aplicam operatorii de mutatie asupra noii populatii. **Incrucisarea** are sansa de a fi aplicata fiecarui cromozom in functie de o probabilitate pe care o stabilim noi, P_CROSS.

Mutatia se aplica la nivelul fiecarei gene a fiecarui cromozom, fiecare gena avand probabilitatea P_MUTATION de a fi alterata.

Un „trick” pe care l-am facut aici pentru a obtine rezultate mai bune a fost sa aplic mutatia acelor cromozomi care nu intra in incrucisare.

```
for (int i = 0; i < popSize; ++i) {
	if (chromosomeWillBeCrossed[i]) continue;
	for (int j = 0; j < sumLgReprez; ++j) {
		randomNr = RandomDouble(0, 1);
		if (randomNr < P_MUTATION)
			chromosomes[i][j] = 1 - chromosomes[i][j];
	}
}
```

3. Rezultate experimentale

In tabelul de mai jos se pot observa valorile obtinute cu algoritmii HC si SA, in ambele lor variante (Best Improvement si First Improvement

Algoritm testat	Functie testata	Numar de dimensiuni	Nr. rulari	Minim	Medie	Maxim	Deviatie	Minin	Medie	Maxim	Deviatie
				Best improvement				First improvement			
HC	Rastrigin	5	40	9.745	17.09	24.254	10.66	7.22	16.83	23.92	11.77
		10	50	50.78	70.33	84.24	17.56	48.97	70.5	86.93	17.22
		30	100	314.97	348.11	376.061	24.15	280.80	349.06	386.03	30.41
SA		5	40	12.86	26.33	36.7	16.28	14	25.2	38.04	17.85
		10	50	56.924	86.244	100.764	24.32	63.03	85.89	106.73	19.85
		30	100	328.55	377.571	409.05	31.55	333.62	378.41	411.59	30.67
HC	De Jong	5	40	0.26	1.19	2.33	1.58	0.24	1.34	2.28	1.66
		10	50	4.11	12.81	20.24	6.98	7.78	13.61	18.26	5.22
		30	100	74.75	112.78	133.90	17.79	85.74	112.25	133.24	21.73
SA		5	40	0.76	2.86	5.23	3.58	1.11	3.13	5.63	3.32
		10	50	11.28	20.46	27.46	8.86	8.10	19.65	28.77	10.82
		30	100	82.27	130.79	190.94	24.73	92.10	132.09	159.17	25.65
HC	Schwefel 7	5	40	-1997	-1606	-1385	386	-1764	-1564	-1422	248
		10	50	-2904	-2336	-2043	400	-2891	-2318	-2082	430
		30	100	-5073	-4043	-3520	584	-5221	-4030	-3544	514
SA		5	40	-1709	-1377	-1172	394	-1577	-1349	-1177	301
		10	50	-2773	-1939	-1608	484	-2574	-1950	-1545	418
		30	100	-5407	-3396	-2731	764	-4418	-3436	-2749	679
HC	Six Hump Camel	2	40	-1.03163	-1.0294	-1.02406	0.01633	-1.031	-1.029	-1.019	0.015
SA				-1.03145	-1.0128	-0.9353	0.14309	-1.031	-1.01	-0.944	0.131

Sa analizam acum rezultatele obtinute cu algoritmul genetic.

Rezultatele au fost obtinute cu o populatie initiala de 4000 de cromozomi, pe 15 iteratii, si P_CROSS = 0.25 si P_MUTATION = 0.0075.

Algoritm testat	Functie testata	Numar de dimensiuni	Nr. rulari	Minim	Medie	Maxim
Algoritm Genetic	Rastrigin	5	15	0.0062	0.007	0.008
		10		0.0124	0.4177	2.53968
		30		0.037	20.227	94.0366
	De Jong	5		0.00003	0.00007	0.00013
		10		0.0004	0.00094	0.00176
		30		0.0065	0.0094	0.015
	Schwefel 7	5		-2088	-1993	-1837
		10		-4061	-3756	-3212
		30		-10767	-10505	-10265
	Six Hump Camel	2		-1.03156	-1.031	-1.0267

Observam o imbunatatire **MAJORA** la De Jong, Rastrigin si Schwefel 7 (am implementat bine!!!). Valorile obtinute sunt mult, mult mai bune. Adaptabilitatea algoritmului genetic face ca modul in care evolueaza popoulatia initiala sa rezulte in rezultate din ce in ce mai bune, ajungandu-se foarte aproape de minimul global al functiilor in discutie.

Totusi, o evolutie prea „agresiva” nu ne avantajeaza. Asta inseamna ori ca incrucisam prea multi cromozomi, ori ca mutatia se realizeaza prea des. Daca mutatia se realizeaza prea des, spre exemplu, un individ nu poate evolua pe aceeasi cale la fel de bine, pentru ca ar ajunge sa fie complet diferit daca ar suferi mutatii prea des. De asemenea, daca incrucisam cromozomii prea des, acelasi lucru se intampla.

Astfel, daca maresc P_MUTATION la 0.1 spre exemplu, obtin rezultate mai proaste. In cazul Schwefel7, minimul nu scade mai mult de -5000.

Un alt lucru crucial este cum definim „Cat timp populatia evolueaza”. Asta este ceea ce permite ca in viitor sa obtinem rezultate mai bune. Cu cat lasam populatia sa evolueze mai mult, cu atata o sa obtinem rezultate mai bune. Desigur, la schimbul unui timp de executie mai mare.

Am definit aceasta evolutie astfel:

```
bool PopulationIsEvoluating(int lastBestGeneration, int currentGeneration) {
	if (currentGeneration - lastBestGeneration >= 50)
		return false;
	return true;
}
```

Cat timp am o evolutie in ultimele 50 de generatii, continui algoritmul.

Este clar ca algoritmul genetic se comporta mult mai bine decat Hill Climbing si Simulated Annealing.Probabil ca o imbunatatire care s-ar putea realiza ar fi sa construiesc un algoritm „hibrid”. Dupa ce algoritmul genetic a gasit cea mai buna solutie, sa aplic HC/SA pe acea solutie.