Proiectul 3

Probleme de optimizare cu algoritmi de inspirație naturală

Cuzic Gabriela info3 – grupa 2

Probleme de optimizare discretă: Problema rucsacului

1. Descrierea problemei

Fie n obiecte, fiecare având o valoare și o greutate. Trebuie găsit un set de obiecte din cele n, ce pot fi plasate într-un rucsac ce are o capacitate dată, astfel încât valoarea totală a obiectelor alese să fie cât mai mare.

S-au folosit ca date de test 20 de obiecte prezentate în tabelul de mai jos, și un rucsac de capacitate 879. Datele vor fi citite din fișier. **Soluția optimă este 1025**.

#	Valoare	Greutate
1	91	84
2	72	83
3	90	43
4	46	4
5	55	44
6	8	6
7	35	82
8	75	92
9	61	25
10	15	83
11	77	56
12	40	18
13	63	58
14	75	14
15	29	48
16	75	70
17	17	96
18	78	32
19	40	68
20	44	92

2. Detalii de aplicare a algoritmilor

Obiectele au fost reprezentate prin intermediul clasei *Obiect,* cu un atribute pentru greutate și valoare.

Soluțiile candidat au fost reprezentate prin intermediul clasei *Solutie,* cu un atribut de tip BitSet.

Am definit o clasă abstractă *InstantaRucsac,* în care am definit atributele ce țin de datele de test, dar și metodele comune celor doi algoritmi.

Evaluarea unei soluții va fi realizata cu metoda *evaluare()*, și va reprezenta suma valorilor obiectelor ce alcătuiesc soluția respectivă.

Am stabilit ca vecin al unei soluții candidat (reprezentată printr-un BitSet), un BitSet ce diferă de cel al soluției candidat printr-un singur Bit.

a) Algoritmul Hillclimbing

Algoritmul este implementat în metoda *gasesteSolutie()* a clasei *HC_Rucsac* ce extinde clasa abstractă *InstantaRucsac.*

Voi genera random o soluție candidat, apoi, pentru **100 de iterații**, obțin cel mai bun vecin al soluției candidat. Dacă acest vecin are o evaluare mai bună (mai mare) decât cea a soluției candidat, vecinul respectiv devine noua soluție candidat.

Soluția finală este obținută la sfârșitul celor 100 de iterații.

b) Algoritmul Simulated Annealing

Algoritmul este implementat în metoda *gasesteSolutie()* a clasei *SA_Rucsac* ce extinde clasa abstractă *InstantaRucsac.*

Am stabilit **temperatura de plecare 1000**, iar numărul **maxim de iterații 100**. Până ajung la temperatura 0, realizez câte 100 de iterații, astfel: obțin un vecin random al soluției candidat. Dacă acest vecin are o evaluare mai bună (mai mare) decât cea a soluției candidat, vecinul respectiv devine noua soluție candidat. Dacă nu, i se mai acordă o șansă de a deveni noua soluție candidat prin **funcția P**.

După cele 100 de iterații, apelez **funcția g** de actualizare a temperaturii : g(T,t)= T-0.000001.

Soluția finală este obținută în momentul în care temperatura devine 0.

3. Rezultate experimentale

a) Algoritmul Hillclimbing

- itMax = 100

#Rulare	Evaluarea soluției
1	323
2	616
3	464
4	358
5	632
6	555
7	464
8	599
9	442
10	307
11	445
12	625
13	539
14	626
15	700
16	746
17	524
18	480
19	601
20	480
21	666
22	542
23	464
24	541
25	396
26	578
27	529
28	525
29	728
30	579

b) Algoritmul Simulated Annealing

- itMax = 100
- T = 1000
- g(T,t)= T-0.000001

#Rulare	Evaluarea soluției
1	633
2	646
3	499
4	459
5	606

6	579
7	463
8	633
9	594
10	662
11	475
12	516
13	924
14	555
15	576
16	428
17	664
18	536
19	681
20	630
21	678
22	656
23	582
24	503
25	723
26	456
27	667
28	483
29	738
30	431

4. Comparații între algoritmi

a) Algoritmul Hillclimbing

Am obținut:

- valoare minimă: 307
- valoare maximă: 746
- valoare medie: 535.8

b) Algoritmul Simulated Annealing

Am obținut:

- valoare minimă: 428
- valoare maximă: 924
- valoare medie: 589.2

Probleme de optimizare continuă: Funcția De Jong 1

1. Descrierea problemei

Fie funcția:

$$f_1(x) = \sum_{i=1}^{n} x_i^2 \qquad -5.12 \le x_i \le 5.12$$

, cu n=2

Trebuie găsit x = (x_1, x_2) pentru care valoarea lui f_1 este minimă.

Pentru n=2 funcția devine:

$f_1(x)$ = sum(x_i^2) , i=1:2, -5.12<= x_i <=5.12.

$f_1(x) = x_1^2 + x_2^2$

Caut perechea (x_1, x_2) din [-5.12, 5.12] care minimizeaza f_1, cu precizie de 6 zecimale. Soluția optimă este x = (0, 0).

2. Detalii de aplicare a algoritmilor

Reprezentare soluții:

Soluțiile candidat au fost reprezentate prin intermediul clasei *SolutieDJ,* cu un atribut de tip BitSet , reprezentând concatenarea a doua bitString-uri corespunzătoare perechii (x_1, x_2).

Soluții x_i posibile in [-5.12, 5.12] => (5.12+5.12) *10^6 = 10.24*10^6 = 10.240.000

8.388.608 = 2^23 < 10.24*10^6 < 2^24 = 16.777.216

=> sunt necesari 24 biti pentru x_i => sunt necesari 48 biti pentru x.

Decodificare:

Metoda *Decodificare* întoarce valorile perechii (x_1, x_2) în baza 10, primind un BitSet reprezentând concatenarea a doua bitString-uri (în baza 2) după următoarea formulă: $x = s + x' * (d - s) / (2^{B+1} - 1)$

În cazul nostru: s = -5.12, d = 5.12, B = 23

$x' = b_B*2^B + b_{B-1}*2^{B-1} + b_{B-2}*2^{B-2} + \ldots + b_1*2^1 + b_0$

$0 <= x' <= 2^{24}-1$ / : (2^{24}-1)

$0 <= x' / (2^{24}-1) <= 1$ /* 10.24

$0 <= x' * 10.24 / (2^{24}-1) <= 10.24$ /-5.12

$-5.12 <= x' * 10.24 /(2^{24}-1) – 5.12 <= 5.12$

$x = x' * 10.24 / (2^{24}-1) – 5.12$

Evaluarea unei soluții va fi realizata cu metoda *evaluare()*, și va reprezenta valoarea funcției în punctul x.

a) Algoritmul Hillclimbing

Algoritmul este implementat în metoda *gasesteSolutie()* a clasei *HC_DeJong.*

Voi genera random o soluție candidat, apoi, pentru **100 de iterații**, obțin cel mai bun vecin al soluției candidat. Dacă acest vecin are o evaluare mai bună (mai mică) decât cea a soluției candidat, vecinul respectiv devine noua soluție candidat.

Soluția finală este obținută la sfârșitul celor 100 de iterații.

b) Algoritmul Simulated Annealing

Algoritmul este implementat în metoda *gasesteSolutie()* a clasei *SA_ DeJong*.

Am stabilit **temperatura de plecare 1000**, iar numărul **maxim de iterații 100**. Până ajung la temperatura 0, realizez câte 100 de iterații, astfel: obțin un vecin random al soluției candidat. Dacă acest vecin are o evaluare mai bună (mai mică) decât cea a soluției candidat, vecinul respectiv devine noua soluție candidat. Dacă nu, i se mai acordă o șansă de a deveni noua soluție candidat prin **funcția P**.

După cele 100 de iterații, apelez **funcția g** de actualizare a temperaturii : g(T,t)= T-0.000001.

Soluția finală este obținută în momentul în care temperatura devine 0.

c) Algoritmul genetic

Algoritmul e implementat în metoda *gasesteSolutie()* a clasei *GA_ DeJong*.

Am stabilit **dimensiunea populației 50**, rata de **încrucișare 0.25**, rata de **mutație 0.01**, numărul de **generații 500**.

Evaluarea unei soluții va fi realizata cu metoda *fitness()*, și va reprezenta valoarea funcției în punctul x.

Inițializez o primă populație în mod aleator, apoi timp de 500 de generații, calculez fitness-ul tuturor indivizilor din populație, dar și suma acestora,

pentru a putea obține probabilitatea fiecărui individ de a fi selectat pentru a se reproduce (cu metoda s*electie()*).

Populației selectate îi aplic metoda *mutatie(),* urmând să aleg în mod aleator câte doi indivizi pentru a produce doi urmași cu metoda *incrucisare().*

Soluția finală este obținută în momentul în care ajung la generația 500.

3. Rezultate experimentale

a) Algoritmul Hillclimbing

- itMax = 100

#Rulare	Val x_1	Val x_2	Val f_1
1	4.344730	3.320970	29.905529
2	3.236883	4.528214	30.982142
3	4.356111	-4.072993	35.564979
4	3.249133	3.396000	22.089688
5	-1.778281	-3.103127	12.791682
6	3.238485	-0.233448	10.542287
7	-2.570721	-1.664568	9.379398
8	-1.542273	1.612986	4.980335
9	3.964641	-3.688285	29.321832
10	-3.944084	-4.568838	36.430086
11	-3.008043	-2.520437	15.400928
12	2.844004	4.733984	30.498972
13	3.874789	-2.928532	23.590300
14	2.465250	-0.811735	6.736375
15	3.448082	4.669999	33.698170
16	-1.476223	2.992410	11.133754
17	-4.086957	0.181320	16.736102
18	-1.265207	-4.642545	23.153982
19	1.160453	-4.918414	25.537458

20	3.339682	-1.470515	13.315895
21	-2.693793	3.827055	21.902876
22	-1.794843	-4.444749	22.977263
23	-0.913839	1.187490	2.245236
24	4.573525	3.279633	31.673136
25	3.241104	3.802220	24.961639
26	-2.253043	-2.822759	13.044178
27	0.680372	0.479085	0.692429
28	1.272573	-3.368029	12.963066
29	-2.453498	3.606203	19.024358
30	-2.725277	-0.509326	7.686551

b) Algoritmul Simulated Annealing

- itMax = 100
- T = 1000
- g(T,t)= T-0.000001

#Rulare	Val x_1	Val x_2	Val f_1
1	-1.879955	-2.454975	9.561138
2	-0.457377	4.416486	19.714543
3	3.562636	-1.574955	15.172865
4	-0.848373	-1.962618	4.571610
5	-2.853688	-4.586459	29.179149
6	2.161644	4.599302	25.826292
7	-2.150367	-1.200357	6.064937
8	-1.095354	0.805951	1.849358
9	-1.264606	1.977305	5.508964
10	-3.040834	-0.377384	9.389095
11	1.490168	1.667485	5.001111

12	-2.846456	2.178645	12.848812
13	-2.500436	-1.120163	7.506951
14	-1.276902	2.293718	6.891622
15	-0.353572	-1.549588	2.526237
16	-5.041968	-1.515991	27.719681
17	-1.067592	-2.143507	5.734378
18	0.624863	-4.136801	17.503582
19	-4.718492	1.088857	23.449781
20	-3.716037	-2.086030	18.160456
21	-1.482051	4.204024	19.870300
22	0.442625	3.440140	12.030486
23	-1.807931	0.551400	3.572660
24	0.390197	1.794205	3.371426
25	1.351967	3.242884	12.344115
26	-2.879507	1.069480	9.435355
27	-2.207073	2.749346	12.430082
28	2.003965	1.780496	7.186045
29	-1.087323	2.711606	8.535080
30	-0.662607	-4.789374	23.377160

c) Algoritmul Genetic

- Nr Generatii = 500
- Nr Indivizi în generație= 50
- Rata mutație= 0.01
- Rata încrucișare= 0.25

#Rulare	Val x_1	Val x_2	Val f_1
1	-2.754525	1.705619	2.909137
2	0.015810	1.272673	1.619698
3	0.572489	1.608997	2.588871

4	-0.697703	1.414035	1.999495
5	4.270345	1.403969	1.971131
6	-2.220829	1.311483	1.719989
7	1.313281	1.451975	2.108234
8	4.711066	1.608086	2.585941
9	3.493624	1.082240	1.171244
10	-2.185535	1.235003	1.525233
11	-3.035368	1.371175	1.880123
12	0.823569	1.184491	1.403018
13	-1.136333	1.706041	2.910576
14	-2.324676	1.648229	2.716661
15	2.757819	1.425285	2.031439
16	-3.359839	1.642219	2.696885
17	-1.986666	1.170817	1.370814
18	4.542960	1.526840	2.331240
19	2.529974	1.664093	2.769207
20	5.025275	1.454253	2.114852
21	2.911112	1.669037	2.785686
22	-0.567062	1.321764	1.747062
23	1.576316	1.474740	2.174859
24	0.207373	1.585744	2.514586
25	3.597413	1.558522	2.428993
26	-0.534289	1.556349	2.422223
27	-3.523824	1.480603	2.192188
28	3.844675	1.341314	1.799125
29	0.197067	1.189715	1.415423
30	1.690959	1.489497	2.218604

4. Comparații între algoritmi

a) Algoritmul Hillclimbing

Am obținut:

- valoare minimă: 0.692429
- valoare maximă: 36.430086
- valoare medie: 19.29868753

b) Algoritmul Simulated Annealing

Am obținut:

- valoare minimă: 1.849358
- valoare maximă: 29.179149
- valoare medie: 12.30248734

c) Algoritmul Genetic

Am obținut:

- valoare minimă: 1.171244
- valoare maximă: 2.910576
- valoare medie: 2.1374179