Problema comis-voiajorului rezolvată de metode euristice

Lipșa Ondina-Patricia
grupa B1

Stanciu Victor-Nicolae
grupa B1

7 ianuarie 2021

Abstract. Acest raport surprinde modul de funcționare a metodei Simulated Annealing și a unui algoritm genetic pentru problema comis-voiajorului, precum și o analiză comparativă dintre cei 2 algoritmi, prin raportarea la rezultatele experimentale obținute. Experimentul a fost rulat de 30 de ori independent, pentru aceleași instanțe ale problemei și s-a putut observa că Simulated Annealing și algoritmul genetic returnează rezultate similare în cazul instanțelor cu 17, 36 și 124 de orașe, iar pentru testele cu peste 300 de orașe, rezultatele obținute prin metoda SA sunt mult mai apropiate de minimul cunoscut.

1 Introducere

Problema comis-voiajorului (PCV) este una dintre cele mai studiate probleme de optimizare și este definită astfel: "găsiți un ciclu de cost minim într-un graf cu n orașe care trece exact o singură dată prin toate nodurile, cunoscând distanța dintre oricare 2 orașe". Varianta asimetrică, folosită în cadrul acestui studiu, se bazează pe aceeași idee, cu precizarea că între 2 orașe A și B distanța de la nodul A la nodul B poate fi diferită de distanța de la nodul B la nodul A.

Aceasta este o problemă NP-dificilă, pentru care nu se cunoaște o rezolvare eficientă folosind algoritmi determiniști, din motive intuitive: având n orașe, există n! posibilități de construire a circuitelor care să viziteze toate orașele, deci pentru doar 10 orașe există peste 3,6 milioane de circuite diferite. Deoarece o căutare printre toate soluțiile posibile devine mult prea costisitoare pe măsură ce numărul de orașe crește, este preferabilă utilizarea algoritmilor euristici, ce aproximează soluția optimă și limitează spațiul de căutare.

2 Metode

2.1 Simulated Annealing

Metoda Simulated Annealing reprezintă o tehnică probabilistică ce generează o potențială soluție începând cu starea curentă și comparând-o cu starea vecinului.

Astfel, în cadrul algoritmului se inițializează temperatura T, ce descrește pe parcursul căutării până la atingerea unei stări relativ stabile (similar cu procesul tehnic al răcirii metalelor). Valorile alese arbitrar din domeniul de definiție, valorile vecinilor, respectiv soluția, au fost reprezentate ca fiind o permutare a orașelor (0, 1, 2, ..., n). Astfel, pentru reprezentarea soluției si calcularea distanțelor dintre orașe, nu este nevoie de decodificarea reprezentării, deoarece șirul decodificat coincide cu cel codificat.

```
Initializare ()
{for (i = 0; i < n; ++i)
        {f0.push_back(0);
        }
for (int i = 0; i < n; ++i)
        {ok = 0;
        do
        {poz = rand() % n;
        if (f0[poz] == 0)
        {ok = 1;
        f0[poz] = 1;
        }
        } while (!ok);
vc.push_back(poz);
}
f0.clear();
}
```

În ceea ce privește determinarea lungimii unui circuit, se utilizează o funcție care îl calculează cu ajutorul costului c[i][j] corespunzator distantei de la nodul "i" la nodul "j".

```
Cost (v)
{s = 0;
for (i = 0; i < n - 1; ++i)
        s += c[v[i]][v[i+1]];
s += c[v[n-1]][0];
return s;
}
```

Cu referire la metoda Simulated Annealing, se inițializează temperatura cu valoarea 1000 și, de asemenea, se inițializează în mod arbitrar o permutare. Pentru determinarea unui vecin al unei permutări, se interschimbă valorile corespunzătoare a două poziții alese arbitrar.

Dacă valoarea curentă nu a fost modificată după vecinilor după un număr de 1000000 de iterații, algoritmul verifică dacă există șansa ca pentru vecin să identifice un vecin al său căruia îi corespunde o lungime mai mică a circuitului cu ajutorul distribuției Boltzmann(ce reprezintă probabilitatea ca procesul să se afle într-o anumită stare referitoare la temperatură și energia abs(Cost(vn)-Cost(vc)). Stările cu o energie mai mică vor avea o probabilitate mai mare de

a fi ocupate. Dacă nici condiția referitoare la distribuția Boltzmann nu este îndeplinită pentru niciunul din vecinii valorii curente, temperatura descrește (T=T*0.999) și tinde să se stabilizeze.

```
void AlgoritmSimulatedAnnealing ()
{T = 1000;
nr_de_iteratii = 0;
vc.clear();
vn.clear();
InitializareAlternativa();
do
{do
        {local = 0;
         do {poz1 = rand() % n;
             poz2 = rand() % n;
             } while (poz2 == poz1);
        vn=vc;
        vn[poz1] = vc[poz2];
        vn[poz2] = vc[poz1];
        Value = Cost (vc);
        min = Value;
        Neighbor = Cost (vn);
        if (Neighbor < min)
        {local = 1;
        vc=vn;
        }
        else
        {
        prob = rand(0,1);
        if (prob < exp(-abs(Neighbor - Value) / T))
        {
        local = 1;
        vc.assign(vn.begin(), vn.end());
        }
}
        ++nr_de_iteratii;
        vn.clear();
        } while (!local && nr_de_iteratii <= 1000000);
T = T * 0.999;
} while (T >= 0.000000000000000001);
CostFinal=Cost(vc);
vn.clear();
}
```

Condiții de oprire: 1) A fost atins un număr de 1000000 de iterații sau probabilitatea aleasă arbitrar < probabilitatea generată de distribuția Boltzmann exp(-abs(Cost(vn)- Cost(vc))/T)); 2) T>=0.000000000000000001.

2.2 Algoritmi genetici

Un algoritm genetic este o metodă meta-euristică inspirată din procesul natural de selecție al algoritmilor evolutivi, care menține o populație de indivizi(genetați inițial aleator). Acești indivizi reprezintă soluțiile candidat ale problemei, iar scopul algoritmului este de avea o populație de indivizi din ce în ce mai adaptați pe măsură ce este iterat algoritmul (fiecare iterație se numește *generație*).

Reprezentarea Indivizii sunt reprezentați printr-un vector ce conține toate cele n orașe, denumit *cromozom*, iar acesta conține mai multe *gene*, fiecare genă fiind un anumit oraș vizitat pe o anumită poziție din ordinea orașelor. Această poziție a vizitării unui oraș se numește *locus*. Ordinea orașelor în cromozom este chiar reprezentarea drumului, cu precizia că ultimul oraș este și primul, pentru a avea un circuit.

```
begin
        generate_population();
        evaluate_population();
        while(nrGenerations<10000) do
                rouleteWheel();
                mutation();
                crossover();
                evaluate_population();
                nrGenerations=nrGenerations+1;
end
```

Algoritmul generează inițial o populație de 100 de indivizi, care sunt evaluați și ulterior selectați în funcție de calitatea lor, iar pe baza reprezentărilor existente se formează soluții noi prin aplicarea operatorilor genetici de mutație și crossover.

Procesul de evaluare, selecție și modificare este repetat (de 10000 de ori) pentru a obține mai multe generații de indivizi.

Evaluarea Pentru fiecare generație, calitatea indivizilor este măsurată pe baza unei funcții *fitness*, pentru a-i putea selecta pe cei mai bine adaptați pentru supraviețuire. Funcția fitness (f) trebuie aleasă invers proporțional cu rezultatul fiecărui individ din cadrul populației, deci un cromozom care are o lungime a circuitului mai mică va avea o valoare mai mare în cadrul funcției fitness.

Am ales funcția f(x)= 1/pathLen, unde pathLen este lungimea circuitului în cadrul cromozomului x.

Selecția Am folosit selecția bazată pe Roata norocului, unde probabilitatea unui individ de a fi selectat este proporțională cu fitness-ul acestuia. Pentru a implementa această metodă de selecție, sunt calculate probabilitățile indivizilor de a fi aleși dupa formula p(i)=f(i)/T, unde T este suma totala a fitness-urilor, și prin adunarea probabilităților q(i+1)=q(i)+p(i) se observă o șansă mai mare ca un număr generat aleator să fie în intervalul (q(i),q(i+1]), pentru un cromozom cu fitness-ul mai mare, deci o șansă mai mare de selecție a acelui individ pentru supraviețuire.

Elitismul Roata norocului are totuși o șansă mică să nu aleagă indivizii mai adaptați, de aceea utilizarea elitismului este foarte utilă. Acesta asigură păstrarea cromozomilor cu cele mai mari valori ale fitness-ului în noua generație și previne aplicarea operatorilor genetici asupra lor (am ales păstrarea primilor 2 cei mai adaptați cromozomi).

Mutația Mutația modifică o genă(sau mai multe) aleasă aleator a unui cromozom dintre cei selectați(cu o probabilitare de 0.7%). Această probabilitate trebuie să fie scăzută, altfel algoritmul ar deveni un simplu algoritm de căutare aleatoare. Scopul mutației este de a introduce diversitate în populația curentă, pentru a evita oprirea într-un punct de minim local. Am folosit 2 tipuri de mutație: mutația TWORS, care inversează 2 gene alese pe poziții aleatoare și mutația RMS, care alege o secvență de gene S, delimitată de două poziții i și j alese aleator și inversează întreaga secvență.

Crossover Este folosit pentru a combina informația genetică dintre 2 cromozomi(cu o probabilitate de 30%), iar rezultatul încrucișării este adăugat în populația din noua generație și astfel valoarea medie a fitness-ului crește (în general) deoarece doar cei mai adaptați indivizi sunt selectați pentru supraviețuire iar majoritatea celor care nu produc rezultate bune sunt eliminați (există o probabilitate mică de selecție a acestora, fiind utili pentru menținerea diversității).

3 Experimente și descrierea lor

Pentru instanțele din tabelul următor, pentru 30 de rulări independente, au fost obținute următoarele date de ieșire (Best, Average, Worst, Deviația standard):

Instanță	Nr.orașe	Algoritm	Best	Avg	Worst	DS	Valoare
br17	17	SA	39	39	39	0	39
br17	17	Genetic	39	39	39	0	39
ftv35	36	SA	1559	1638,27	1742	39,8373	1473
ftv35	36	Genetic	1659	1798,83	2022	94,1779	1473
p43	43	SA	5627	5653,43	5677	13,4766	5620
p43	43	Genetic	5620	5627	5638	4,61158	5620
ft55	56	SA	1776	1964,8	2136	80,3751	1608
ft55	56	Genetic	2002	2232,23	2232,23	116,998	1608
ft53	53	SA	7510	8302,97	9214	365,589	6905
ft53	53	Genetic	8191	9126,7	10027	472,859	6905
ft70	70	SA	40800	41668,7	42819	608,209	38673
ft70	70	Genetic	42268	43769,6	45918	753,457	38673
kro124	100	SA	43250	46833,6	51735	1975,97	36230
kro124	100	Genetic	43584	46552,1	50163	1735,12	36230
rbg323	323	SA	1517	1561,1	1602	24,5226	1326
rbg323	323	Genetic	2546	2702,77	2882	72,0258	1326
rbg403	403	SA	2568	2605,03	2647	19,914	2465
rbg403	403	Genetic	4071	4071	4280	53,4064	2465
rbg443	443	SA	2830	2895,27	2895,27	27,591	2720
rbg443	443	Genetic	4594	4759,57	4944	79,3669	2720

4 Comparații. Concluzii

Precizie. Din punctul de vedere al preciziei valorilor obținute (comparativ cu cele mai bune valori cunoscute), se observă faptul că, în general, valoarea de tip "Best" pentru metoda Simulated Annealing este mai apropiată de lungimea circuitului minim decât de cea de tip "Best" pentru algoritmul genetic implementat. De asemenea, în general, media valorilor obținute pentru algoritmul genetic este mai mare sau egală (sau, cel puțin, foarte apropiată) decât cea specifică celeilalte metode meta-euristice.
Eficiență. Din punctul de vedere al complexității timpului, algoritmul Simulated Annealing este mai eficient decât algoritmul genetic.

Bibliografie

[1] Instanțe:
http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/atsp/

[2] Pagina cursului:
https://profs.info.uaic.ro/~eugennc/teaching/ga/res/gaSlidesOld.pdf
https://profs.info.uaic.ro/~eugennc/teaching/ga/

[3] Simulated Annealing
https://www.hindawi.com/journals/cin/2016/1712630/
https://en.wikipedia.org/wiki/Simulated_annealing
https://www.researchgate.net/publication/233584468_An_Effective_Simulated_Annealing_Algorithm_for_Solving_the_Traveling_Salesman_Problem

[4] Algoritmi genetici
https://www.hindawi.com/journals/cin/2017/7430125/ https://ro.wikipedia.org/wiki/Problema_comis-voiajorului http://comopt.ifi.uni-heidelberg.de/software/ https://www.geeksforgeeks.org/traveling-salesman-problem-using-genetic-algorithm/
https://arxiv.org/ftp/arxiv/papers/1203/1203.3099.pdf

[5] Genetic Algorithms vs. Simulated Annealing https://scholar.smu.edu/cgi/viewcontent.cgi?article=1000&context=engineering_compsci_research
http://www.cplusplus.com/reference/vector/vector/assign/