UNIVERSITATEA TRANSILVANIA

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

Automate, calculabilitate si complexitate

Analiză comparativă între Algoritmii Genetici (GA) și Optimizarea Roiului de Particule adaptată la cazul discret (DPSO) pentru VCP

Nume student: **CHIȚU RALUCA-OANA**

Program licență: **Informatică ID, anul III, an univ. 2023-2024**

Grupa: **10LD511**

CUPRINS

Capitolul 1. Introducere – despre problema acoperirii minimale cu vârfuri (VCP)

1.1. Descrierea problemei VCP

Problema acoperirii minime a vârfurilor este o problemă de optimizare combinatorie de bază, în care scopul este de a găsi un subset de vârfuri care să acopere toate marginile în care numărul de vârfuri din acest subset este cel mai mic. Problema de căutare a acoperirii minime a vertexurilor joacă un rol important în multe aplicații din lumea reală, cum ar fi dezmembrarea rețelelor,rețelele de senzori fără fir, recuperarea informațiilor, rețelele de transport public și bioinformatica. Cu toate acestea, găsirea unui număr minim de vârfuri este una dintre problemele NP-Hard. Prin urmare, pentru a rezolva această problemă de optimizare în timp polinomial, sunt necesare abordări alternative bazate fie pe tehnici de aproximare, fie pe tehnici metaeuristice.

Exemplificare

Fie G = (V, E), unde G este un graf neorientat, V = {1,2,3, ... ,n} este setul de vârfuri, E ⊆ V × V este setul de muchii, iar (u,(v) ∈ E este o muchie între vârfurile u și v. Problema acoperirii minime a vârfurilor constă în căutarea și construirea celui mai mic subset V' ⊆ V astfel încât ∀(u, (v) ∈ E : u ∈ V' sau v ∈ V'. Cu alte cuvinte, vertexul minim implică determinarea numărului minim de vârfuri dintr-un set astfel încât fiecare muchie din graf să aibă cel puțin un vârf în acel set.

Un exemplul de construire a acoperirii de vertexuri într-un graf este prezentat în Fig. 1. Astfel, avem :

- **1(a)** - este o acoperire de vertexuri invalidă deoarece muchia v_2-v_5 nu este acoperită de niciun vertex din set (*ambele puncte finale nu se află în set*)
    - Vârfurile alese sunt :
        - V_1 – acoperă muchia $v_1 – v_2$
        - V_3 – acoperă muchiile v_3 - v_5 și $v_3 – v_6$
        - V_4 – acoperă muchi v_4 - v_5
        - V_6 – acoperă muchia v_6 - v_3 și v_6 - v_5
- **1(b) și 1(c)** – este sunt exemple de acoperiri de vertexuri valide, care pot fi considerate ca fiind soluții posibile. În cazul acestora, toate muchiile sunt acoperite :
    - Vârfurile alese pentru **1(b)** sunt: v1, v3,v4,v5 – acestea acoperă toate muchiile (v1 – v2, v3 – v5; v3 – v6; v4 – v5; v5 – v2; v5 – v6)

- Vârfurile alese pentru **1(c)** sunt: v1, v3, v5 – acestea acoperă toate muchiile (v1 – v2, v3 – v5; v3 – v6; v5 – v2; v5 – v4; v5 – v6)

Dintre cele trei variante prezentate, soluția cea mai bună este 1(c) deorece cu doar trei vârfuri se pot acoperi toate muchiile. În varianta 1(b) sunt utilizate patru vârfuri.

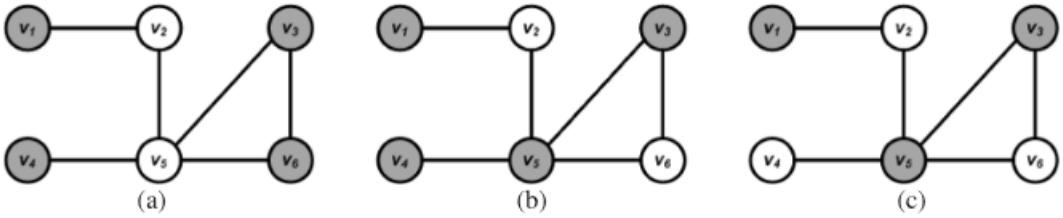


Fig 1. Exemple ale VCP

Deși au fost propuse mai multe tehnici diferite pentru rezolvarea problemei de acoperire minimă a vertexurilor, rezolvarea acestei clase importante de probleme prin utilizarea unor algoritmi de inspirație biologică a fost rareori realizată. Astfel, rezolvarea problemei de acoperire minimă a vertexurilor prin acești algoritmi rămâne încă o provocare. În ultimele două decenii, cercetările anterioare au demonstrat potențialul mare al algoritmilor de inspirație biologică pentru a rezolva în mod eficient o gamă largă de probleme în știință și inginerie. Chiar dacă utilizarea acestor algoritmi de inspirație biologică poate oferi rezultate acceptabile, calitatea soluțiilor obținute prin aceste abordări necesită în continuare îmbunătățiri.

1.2 Domenii aplicative ale VCP

Problema acoperirii cu vârfuri (Vertex Cover Problem - VCP) este o problemă NP-completă din teoria grafurilor, cu multe aplicații practice în diverse domenii. Câteva domenii în care VCP este relevantă sunt :

1. Rețele de comunicații - design și optimizare de rețele

   - **rețele de senzori**: în rețelele de senzori wireless, se poate folosi VCP pentru a determina un set minim de senzori care trebuie activi pentru a monitoriza toate punctele de interes.
   - **rețele de telecomunicații**: optimizarea infrastructurii rețelelor de telecomunicații, unde trebuie plasate stații de bază sau repetitori pentru a acoperi toate punctele de utilizatori.

2. Bioinformatică - analiza interacțiunilor proteice

- **rețele de Interacțiuni proteice**: în bioinformatică, VCP poate fi utilizată pentru a găsi un set minim de proteine care interacționează direct sau indirect cu toate proteinele dintr-un set dat.

3. Securitate și supraveghere - sisteme de supraveghere video

- **plasarea camerelor**: determinarea unui set minim de camere de supraveghere pentru a monitoriza toate zonele de interes într-o clădire sau spațiu public.

4. Proiectare de circuite - testarea și verificarea circuitelor

- **testarea circuitelor**: în proiectarea circuitelor integrate foarte complexe, VCP poate fi folosită pentru a selecta un set minim de puncte de testare care pot detecta toate defectele posibile.

5. Managementul proiectelor - alocarea resurselor

- **managementul sarcinilor**: determinarea unui set minim de resurse (de exemplu, echipe de lucru) necesare pentru a acoperi toate sarcinile unui proiect, unde fiecare echipă poate îndeplini anumite sarcini specifice.

6. Inginerie software - testarea și validarea

- **testarea software-ului**: selectarea unui set minim de cazuri de test care acoperă toate punctele de cod sau funcționalități ale unui program software.

7. Rețele de transport - plasarea senzorilor de trafic

- **monitorizarea traficului**: în infrastructura de transport, VCP poate ajuta la determinarea unui set minim de senzori de trafic pentru a monitoriza toate intersecțiile sau punctele critice dintr-o rețea rutieră

8. Domeniul economic - optimizarea costurilor

- **alocarea bugetară**: determinarea unui set minim de investiții care acoperă toate nevoile unui proiect economic, pentru a minimiza costurile și a maximiza eficiența.

9. Inteligenta Artificială și Teoria Jocurilor - planificare și strategie

   - **jocuri și simulări**: în contextul jocurilor și simulărilor, VCP poate fi utilizată pentru a găsi strategii optime de acoperire a tuturor scenariilor posibile cu un set minim de resurse.

Capitolul 2. Metodologie

2.1. Descriere implementare algoritmi

2.1.1. Algoritm Genetic (AG)

CODIFICARE

În cazul algoritmilor genetici există mai multe metode de codificare a soluției și anume:

- **Codificare de tip binar (*binary encoding*):** reprezintă cea mai comună metodă și presupune ca soluțiile (*cromozomii*) să fie reprezentați sub formă de stringuri de tip binar (*care conțin doar 1 și 0);*
- **Codificare de tip permutare (*permutation encoding*):** utilizată pentru probleme care necesită operații de ordonare (*de exemplu, problema lanificării job-urilor*);
- **Codificarea bazată pe valoare (*value encoding):*** se folosește în probleme unde soluțiile sunt prezentate sub formă de valori de tip real sau complex (cum ar fi problema rucsacului în care scopul este maximiza valoarea totală a obiectelor din rucsac fără a depăși capacitatea de greutate a acestuia;
- **Codificare de tip arbore (*tree encoding*):** utilizată pentru programarea genetică.

Pentru problema VCP, am ales **codificarea binară** întrucât decizia de a stabili dacă o soluție este bună sau nu, se rezumă la a răspunde cu DA și NU. Soluțiile în acest caz nu necesită a fi ordonate și totodată valoarea fiecărui vârf nu este de interes pentru rezolvarea problemei. De aceea, codificarea de tip permutare, cât și cea bazată pe valoare nu reprezintă o alegere bună. Totodată, nici codificarea de tip arbore nu este o alegere bună, întrucât VCP nu este o problemă direcționată către programarea genetică.

FUNCȚIA FITNESS (OBIECTIV)

În algoritmii genetici, **funcția obiectiv (sau funcția de fitness)** este o componentă esențială. Această funcție este utilizată pentru a evalua cât de "bună" este o soluție în cadrul algoritmului. Scopul este de a optimiza sau de a găsi o soluție care maximizează sau minimizează această funcție, în funcție de problema specifică pe care algoritmul genetic încearcă să o rezolve.

Pentru a putea rezolva problema acoperirii cu vârfuri (VCP) prin aplicarea algoritmilor genetici, am decis utilizarea următoarei funcții obiectiv, transformând problema într-una de minimizarea a acestei funcții:

Funcție obiectiv = număr de vărfuri selectate (*minim posibil)* + alfa (*coeficient de penalizare)* x număr muchii neacoperite

Notă: coeficientul alfa este introdus pentru a penaliza acele soluții care nu sunt bune, adică acele soluții în care selecția de vârfuri, nu acoperă toate muchiile.

SELECȚIA

Selecția este procesul prin care indivizii dintr-o populație sunt selectați pentru a fi folosiți în procesul de reproducere și generare a unei noi generații de indivizi. Scopul selecției este de a favoriza transmiterea caracteristicilor dorite și de a îmbunătăți performanța soluției în cadrul algoritmului genetic.

Există mai multe metode de selecție folosite în algoritmii genetici, iar fiecare metodă are avantaje și dezavantaje în funcție de problema specifică și de obiectivele algoritmului. Iată câteva dintre cele mai comune metode de selecție:

- **Selecția aleatoare**: în această metodă, indivizii din populația actuală sunt selectați pentru a participa la procesul de reproducere în mod aleatoriu, fără a ține cont de fitnessul lor. Fiecare individ are o șansă egală de a fi selectat.
- **Ruleta (roulette wheel selection):** Fiecare individ are o șansă proporțională de a fi selectat pentru reproducere, în funcție de fitnessul său relativ în cadrul populației. Indivizii cu fitness mai mare au o

probabilitate mai mare de a fi selectați, dar chiar și cei cu fitness scăzut au o șansă de a fi aleși.

- **Turnir (tournament selection):** Indivizii sunt grupați aleatoriu în turnee și cel mai bun individ din fiecare turneu este selectat pentru reproducere. Mărimea turneului poate varia și poate fi o variabilă ajustabilă a algoritmului.
- **Rank (rank selection):** Indivizii sunt sortați în funcție de fitnessul lor și li se atribuie un rang. Probabilitatea de a fi selectat este invers proporțională cu rangul lor, ceea ce înseamnă că indivizii mai buni au o probabilitate mai mare de a fi aleși.
- **Metoda de selecție cu prag (threshold selection**) este o tehnică simplă și intuitivă utilizată în algoritmii genetici pentru a selecta indivizii pentru reproducere. În această metodă, se stabilește un prag de fitness, iar toți indivizii cu un fitness mai mare sau egal cu acest prag sunt selectați pentru a fi incluși în populația de reproducere.

În vederea rezolvării problemei VCP, am ales ca și metode de selecție: metoda selecției aleatoare și cea a ruletei. Motivația alegerii lor constă în două motive principale și anume:

- fiind la începutul studiului algoritmilor genetici, cele două metode oferă o implementare mai simplă și ușor de înțeles, ambele fiind prezentate în cadrul cursului;
- am dorit să evaluez diferența între o metodă care ține cont de funcția de fitness (metoda ruletă) și o metodă care nu ține cont de această funcție (metoda selecției aleatoare).

ÎNCRUCIȘARE

Încrucișarea ca și termen general reprezintă, în reproducerea umană, combinarea informației genetice dintre doi indivizi (*părinții)*, rezultând un individ nou (*copilul)*.

În sfera algoritmilor genetici, acest fenomen este reprezentat printr-un operator genetic și este o modalitate de a genera soluții noi fie printr-o distribuție aleatoare a unei probabilități sau a unui model care poate fi analizat static, dar nu poate fi prezis cu acuratețe.

Există mai multe metode de încrucișare care pot fi aplicate în funcție de codificarea aleasă pentru soluția problemei ce se dorește a fi rezolvată. Astfel, avem:

- **Încrucișarea într-un singur punct (*Single point crossover*):** se selectează în mod arbitrar un singur punct de încrucișare, iar materialul

genetic este schimbat astfel: se ia de la primul părinte toată informația de la început până la punctul de încrucișare, iar de la al doilea părinte toată informația care este după punctul de încrucișare. Acest tip de încrucișare se poate folosi pentru: binary encoding, permutation encoding, value encoding.

- **Încrucișarea în două puncte (*Two point crossover*):** se selectează în mod arbitrar două puncte de încrucișare și astfel cromozomul este împărțit în trei segmente: un segment înainte de primul punct de încrucișare; un segment între cele două puncte de intersecție și un segment după al doilea punct de intersecție. Astfel se ia de la primul părinte toată informația de la început până la primul punct de încrucișare, de la al doilea părinte informația dintre cele două puncte de încrucișare, iar ceea ce este după al doilea punct de încrucișare se ia din nou de la primul părinte. Este folosită pentru: binary encoding și value encoding.
- **Încrucișarea uniformă (*Uniform crossover*):** informația genetică este aleasă în mod aleatoriu de la unul dintre cei doi părinți cu o anumită probabilitate, rezultând un amestec uniform de informație. Se utilizează pentru binary encoding și value encoding.
- **Încrucișarea aritmetică (*Arithmetic crossover*):** presupune efectuarea unor operații aritmetice între doi părinți, rezultând un nou individ (*soluție).* Acest tip de încrucișare se poate folosi pentru: binary encoding și value encoding.
- **Încrucișare arborelului (*Tree crossover*):** se selectează câte un punct de încrucișare pentru fiecare părinte, apoi aceștia sunt împărțiți în acest punct. Partea de sub aceste puncte de încrucișare, este schimbată între părinți pentru a crea arbori descendenți.

Algerea metodei de încrucișare are la bază în primul rând tipul de codificare a problemei, în cazul de față fiind codificare binară. Deși, cu excepția încrucișării arborelui, toate metodele prezentate anterior sunt aplicabile, în general când avem de a face cu binary coding, metoda încrucisării în două puncte este preferată deoarece oferă un echilibru foarte bun între explorare și exploatare, permițând schimbul de material genetic în mai multe poziții, menținând în același timp o anumită integritate structurală în cromozomii descendenți.Cu alte cuvinte, simplist spus, informația genetică este combinată în așa fel încât individul rezultat conține informație genetică destul de apropiată de părinți. În cazul metodei uniform crossover, individul rezultat poate să difere foarte mult de părinți, în timp ce arithmetic crossover are nevoie de adaptări. Totodată, în cazul metodelor single point și aritmetic crossover, există riscul ca individul rezultat să fie, de fapt, unul dintre părinți.

Pe aceste considerente, am ales ca metodă de încrucișare **two point crossover**.

MUTAȚIA

În termeni simpli, mutația poate fi definită ca o mică modificare aleatorie a cromozomului, pentru a obține o nouă soluție. Ea este utilizată pentru a menține și a introduce diversitate în populația genetică.

În general, indivizii (*soluțiile*) rezultate în urma procesului de încrucișare sunt supuse procesului de mutație înainte de a fi introduse în populație. Aceasta presupune modificarea aleatorie a unor gene (*parametri*) ale urmașilor pe baza a ceea ce se numește probabilitate de mutație. Aceasta este specificată de către utilizator, însă dacă această are o valoare prea mare atunci transformarea devine în fapt o căutare aleatorie primitivă. De aceea ea trebuie să fie redusă (*scăzută)*. Scopul este de a introduce un nou material genetic în populație care poate duce la soluții mai bune.

Există mai multe tipuri de mutații dintre care unele din cele mai comune sunt :

- **Mutație de biți**: acest tip de mutație se aplică atunci când indivizii sunt reprezentați ca șiruri binare. Acesta răstoarnă aleatoriu biții individuali cu o anumită probabilitate (*în funcție de această probabilitate, dată de rata mutației, 0 este schimbat cu 1 și invers, 1 este schimbat cu 0*).
- **Mutație gaussiană**: se utilizează atunci când indivizii sunt reprezentați ca vectori cu valori reale. Mutația gaussiană adaugă fiecărei gene o valoare aleatorie extrasă dintr-o distribuție gaussiană cu o medie și o abatere standard specificate.

- **Mutație uniformă**: similară cu mutația de biți, dar aplicabilă indivizilor reprezentați ca vectori de valori discrete. Schimbă aleatoriu valoarea unei gene cu o altă valoare într-un interval predefinit.
- **Mutație de schimb**: potrivită pentru reprezentările bazate pe permutări. Mutația de schimbare selectează două poziții în cromozom și schimbă elementele din aceste poziții.
- **Mutația de inversiune**: acest operator de mutație selectează în mod aleatoriu un subset de gene și inversează ordinea acestora în cadrul cromozomului.
- **Mutația de alunecare**: utilizată în reprezentările cu valori reale, mutația creep adaugă o mică valoare aleatorie la fiecare genă într-un anumit interval.
- **Mutație de limită**: limitează mutația pentru a rămâne în limitele predefinite pentru fiecare genă din cromozom.
- **Mutație neuniformă**: similară mutației gaussiene, dar cu un parametru de scară neconstantă. Începe cu o rată de mutație mare și o scade în timp.

Întrucât codificarea aleasă este de tip binary encoding, este de la sine înțeles că metoda (operatorul) de mutație utilizat este implicit metoda mutației de biți (*bit inversion).*

Notă: deși aplicabilă în cazul codificării binare, mutația uniformă presupune ca indivizii să fie prezentați ca vectori de valori discrete.

2.1.2. Algoritm Optimizarea discretă a roiului de particule (DPSO)

VARIANTĂ DPSO

Optimizarea discretă a roiului de particule (DPSO) este unul dintre cei mai recent dezvoltați algoritmi de optimizare meta-heuristică bazată pe populație în cadrul inteligenței roiului, care poate fi utilizat în orice problemă de optimizare discretă.

Acest algoritm are la bază, în fapt, algoritmul PSO (*Particle Swarm Optimization*). PSO, pe scurt, presupune ca un grup de indivizi (*particule*) se interconecteză, comunică unii cu ceilalți, astfel încât să aibă loc alinierea lor și deplasarea către un obiectiv comun (*exemplu: hrană*). Pe parcursul procesului de iterație PSO, fiecare particulă își actualizează locația atât pe baza cunoștințelor sau a experienței anterioare, precum și pe baza cunoștiințelor primite de la o anumită particulă vecină. Asta înseamnă că, analizând o anumită particulă, își menține poziția și învață din experiență întâlnită atunci când navighează ca un asemeni unui stol.

Modelul de navigație al particulelor este important, deoarece comunicarea este vitală în timpul procesului de navigație. Feedback-ul este primit atât de la nivelul local, cât și de la cel global, în timpul procesului de căutare globală.

Algoritmul DPSO are mai multe variații pentru probleme care presupun utilizarea de variabile discrete (numere care nu sunt de tip real, de exemplu) printre care:

- **DPSO de tip binar (*binary DPSO*)** - poziția fiecărei particule este actualizată pe baza unei funcții sigmoide aplicată vitezei de mișcare a acesteia
- **DPSO de tip discret (*discrete DPSO*)** – particulele se mișcă într-un spațiu discret
- **DPSO de tip integer (*integer DPSO*)** – pozițiile particulelor și viteza lor sunt de tip integer
- **DPSO de tip permutare (*permutation-based DPSO*)** - particulele reprezintă permutări, iar actualizările implică operatori de permutare, cum ar fi permutări, inversări sau inserții.

Pentru problema acoperirii cu vârfuri am ales varianta de tip binar: **binary DPSO**

FUNCȚIA FITNESS (OBIECTIV)

Pentru a putea rezolva problema acoperirii cu vârfuri (VCP) prin aplicarea algoritmului binary DPSO, am decis utilizarea următoarei funcții obiectiv:

Funcție obiectiv = număr de vărfuri selectate (*minim posibil)* + alfa (*coeficient de penalizare)* x număr muchii neacoperite

Notă: coeficientul alfa este introdus pentru a penaliza acele soluții care nu sunt bune, adică acele soluții în care selecția de vârfuri, nu acoperă toate muchiile.

PAȘII ALGORITMULUI

Pașii pentru implementarea algoritmului de Optimizare prin Roi de Particule Discretă (DPSO) pentru problema Acoperirii de Vârfuri:

1. Definire reprezentare soluție: în cazul acesta, soluția va fi un subset de vârfuri
2. Inițializare: un set de particule aleatorii în spațiul de căutare a soluțiilor. Fiecare particulă va avea o poziție (soluție) și o viteză inițială.

3. Evaluare: fiecare particulă se va evalua folosind funcția obiectiv care măsoară cât de bine acoperă vârfurile grafului.
4. Actualizare poziții și viteze: pe baza formulelor

2.2. Procesul de calibrare

2.2.1 Algoritmi genetici

Procesul de calibrare în cazul rezolvării problemei algoritmilor genetici acoperirii cu vârfuri (VCP) presupune atât analiza parametrilor utilizați, cât și modul de testare al algoritmului implementat. Astfel, avem :

a. parametri – descriere și influență :

- mărimea populației : reprezintă numărul de soluții candidate (indivizi) dintr-o generație. Acest parametru inflențează :
  - diversitatea soluțiilor: o populație mai mare asigură o diversitate genetică mai mare, ceea ce poate ajuta la explorarea mai eficientă a spațiului soluțiilor și la evitarea optimizării locale premature.
  - costul computațional: o populație mai mare necesită mai mult timp și resurse pentru evaluarea și manipularea indivizilor în fiecare generație.
  - convergența : o populație mică poate duce la convergență rapidă (minime locale)
- numărul de generații : numărul de generații reprezintă numărul de iterări pe care algoritmul le parcurge înainte de a se opri. Influențează :
  - calitatea soluției: mai multe generații permit algoritmului să evolueze soluțiile pe o perioadă mai lungă de timp, ceea ce poate duce la soluții mai bune.
  - costul computațional: creșterea numărului de generații crește timpul total de execuție a algoritmului.
  - evitarea stagnării: dacă algoritmul stagnează și nu mai găsește soluții mai bune, creșterea numărului de generații nu va îmbunătăți rezultatul.
- rata de mutație : rata de mutație este probabilitatea cu care se modifică un individ (soluție) la fiecare generație. Influențează :
  - diversitatea genetică: o rată de mutație mai mare crește diversitatea soluțiilor, ceea ce ajută la explorarea spațiului soluțiilor și la evitarea optimizării locale premature.

- stabilitatea soluțiilor: o rată de mutație prea mare poate introduce prea multă variabilitate, destabilizând soluțiile bune și împiedicând convergența (stabilizarea algoritmului)
- exploatare vs. explorare: o rată de mutație adecvată trebuie să mențină un echilibru între explorarea de noi soluții și exploatarea soluțiilor existente.

➢ rata de încrucișare : rata de încrucișare este probabilitatea cu care două soluții (indivizi) se combină pentru a produce descendenți. Influențează :

- recombinarea genelor: o rată de încrucișare mai mare favorizează recombinarea soluțiilor existente, ceea ce poate produce indivizi mai buni prin combinarea caracteristicilor favorabile din părinți diferiți.
- diversitatea genetică: rata de încrucișare afectează diversitatea populației, dar nu în aceeași măsură ca rata de mutație. Ea contribuie la diversitate prin combinarea diferitelor gene.
- stabilitatea și convergența: o rată de încrucișare prea mare poate împiedica convergența dacă soluțiile bune sunt distruse frecvent. În schimb, o rată prea mică poate limita capacitatea algoritmului de a găsi soluții mai bune prin recombinare.

➢ coeficientul de penalizare : este o valoare adăugată la funcția de fitness pentru a penaliza soluțiile care nu respectă constrângerile problemei. În cazul VCP, soluțiile sunt penalizate dacă nu acoperă toate muchiile din graf. Rolul său este:

- descurajarea soluțiilor incorecte: coeficientul de penalizare ajută la descurajarea soluțiilor care nu respectă complet constrângerile problemei. De exemplu, în VCP, dacă o soluție nu acoperă toate muchiile grafului, funcția de fitness va include o penalizare proporțională cu numărul de muchii neacoperite.
- ghidarea evoluției: penalizările pot ghida algoritmul genetic să evolueze spre soluții valide prin favorizarea soluțiilor care respectă constrângerile problemei.
- evitarea convergenței premature: coeficientul de penalizare ajută la menținerea diversității în populație prin penalizarea soluțiilor suboptimale (minime locale), prevenind astfel convergența prematură la soluții nevalide sau suboptime.

Un coeficient de penalizare mare va penaliza sever soluțiile nevalide, forțând algoritmul să se concentreze pe soluții care respectă constrângerile. Pe de altă parte, însă, un coeficient de penalizare mic poate permite soluțiilor nevalide să persiste în populație, ceea ce poate fi util în fazele timpurii ale evoluției pentru a explora un spațiu de soluții mai larg. De aceea, este important să se efectueze experimente pentru a calibra corect acest parametru.

b. valori parametri ca set de date :

Date testare POPULATION_SIZE

Nr. crt.	POPULATION_SIZE
1	20
2	50
3	100
4	150
5	200
6	250
7	300
8	350
9	400
10	500

Date testare MAX_GENERATIONS

Nr. crt.	MAX_GENERATIONS
1	50
2	100
3	200
4	300
5	400
6	500
7	600
8	700
9	800
10	1000

Date testare MUTATION_RATE

Nr. crt.	MUTATION_RATE
1	0.01
2	0.05
3	0.1
4	0.15
5	0.2
6	0.25
7	0.3
8	0.35
9	0.4
10	0.5

Date testare CROSSOVER_RATE

Nr. crt.	CROSSOVER_RATE
1	0.5
2	0.6
3	0.65
4	0.7
5	0.75
6	0.8
7	0.85
8	0.9
9	0.95
10	1

Date testare ALPHA

Nr. crt.	ALPHA
1	10
2	50
3	100
4	200
5	300
6	400
7	500
8	600
9	700
10	750

2.2.2 DPSO

a. parametri – descriere și influență :

- numărul de particule: reprezintă numărul de particule din populația (sau „stolul”) care sunt evaluate în fiecare iterație a algoritmului. O valoare mai mare poate ajuta la explorarea unui spațiu de căutare mai mare, dar poate face și algoritmul să fie mai costisitor din punct de vedere computațional.
- numărul maxim de iterații: cu cât este mai mare, cu atât algoritmul are mai mult timp pentru a căuta soluții mai bune, dar creșterea acestui număr poate crește și timpul de calcul.
- parametrul w: este factorul de inerție, care controlează proporția cu care viteza particulelor din stol sunt afectate de propria lor viteză anterioară. O valoare mai mare a lui w încurajează explorarea, în timp ce o valoare mai mică încurajează exploatarea. Acesta este un echilibru delicat între explorare și exploatare.
- parametrii c_1 și c_2 : reprezintă coeficienții de învățare cognitivă și socială, care controlează influența vecinilor și cea a celei mai bune poziții găsite până în prezent de o particulă asupra vitezei acelei particule. Valori mai mari pentru acești coeficienți încurajează o influență mai mare din partea celor mai bune poziții locale sau globale găsite până în prezent.
- parametrul alpha: este un parametru suplimentar care poate fi folosit pentru a controla o anumită caracteristică specifică a algoritmului, cum ar fi viteza sau dimensiunea pașilor. Poate fi folosit pentru a ajusta comportamentul algoritmului în funcție de specificul problemei care este optimizată

b. valori parametri ca set de date :

Date testare swarmSize

Nr. crt.	swarmSize
1	20
2	50
3	100
4	150
5	200
6	250
7	300
8	350
9	400
10	500

Date testare maxIterations

Nr. crt.	maxIterations
1	50
2	100
3	200
4	300
5	400
6	500
7	600
8	700
9	800
10	1000

Date testare w

Nr. crt.	w
1	0.5
2	0.7
3	0.4
4	0.6
5	0.3
6	0.55
7	0.8
8	0.45
9	0.65
10	0.75

Date testare c1

Nr. crt.	c1
1	1.05
2	-1.5
3	1.5
4	-1.3
5	1.8
6	-1.2
7	1.2
8	1.7
9	1.9
10	2

Date testare c2

Nr. crt.	c2
1	-1.9
2	1.5
3	0.5
4	1.4
5	-1.4
6	1.2
7	1.1
8	0.45
9	1.9
10	1.3

Date testare alpha

Nr. crt.	alpha
1	10
2	8
3	12
4	15
5	5
6	9
7	11
8	7
9	13
10	14

2.3. Descrierea setului de date de test – ambii algoritmi

În ceea ce privește stabilirea setului de date pentru testare, acesta poate fi construit fie în mod automatizat, fie manual.

Metoda manuală care poate conduce la un set de date, este cea în care controlul conectării muchiilor nu mai este automatizat. În acest sens, se poate folosi un editor pentru grafuri ((https://csacademy.com/app/graph_editor/) sau literatura de specialitate(https://research.ijcaonline.org/volume82/number4/pxc3892126.pdf).

Însă, pentru analiza celor doi algoritmi (AG și DPSO) setul de date a fost generat automat fiind selectat pe baza unei metode de generare a munchilor, în funcție de numărul de vârfuri.

```
void generateCompleteGraph() {
    for (int u = 1; u <= V; ++u) {
        for (int v = u + 1; v <= V; ++v) {
            addEdge(u, v);
        }
    }
```

V – număr de vârfuri

Astfel, s-au luat în considerare, din punct de vedere al numărului de vârfuri următoarele date :

Nr. crt.	Număr vârfuri
1	10
2	20
3	30
4	40
5	50
6	60
7	70
8	80
9	90
10	100

Totodată, acest set de date a fost aplicat pentru ambii algoritmi.

2.4. Instanțe de testare – combinare date

Pentru cei doi algoritmi, AG și DPSO, s-au luat în calcul 10 testări, fiecare cu date diferite.

Algerea datelor s-a făcut pe justificarea de a combina valori mari cu valori mici, astfel să existe un oarecare echilibru între date și de a observa comportamul algoritmilor când există o variație mare-mic între datele variabilelor constante.

2.4.1 Algoritmi genetici

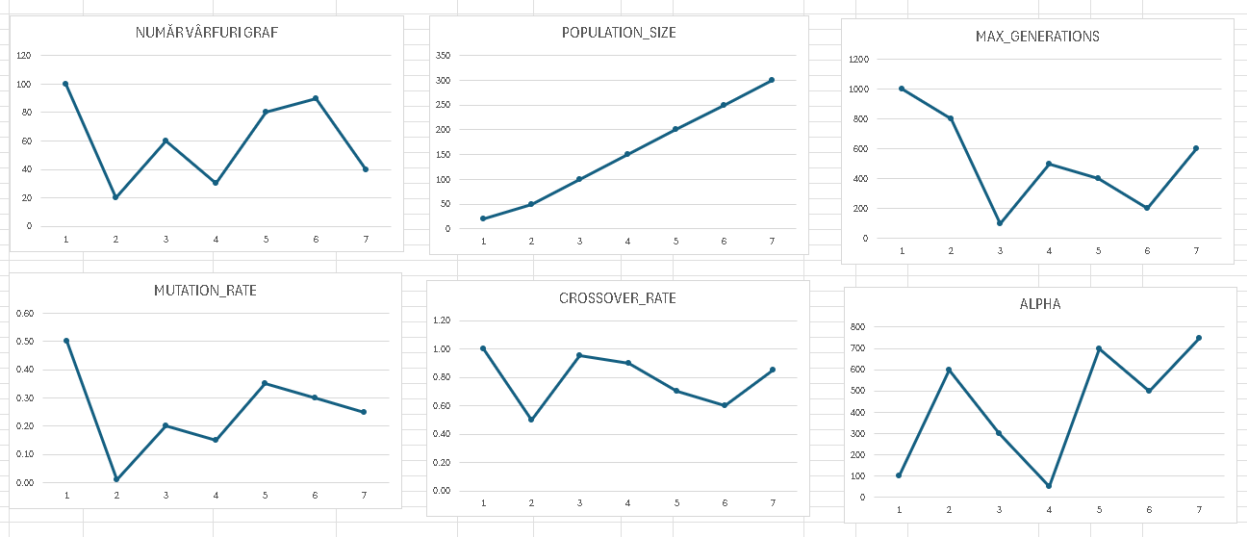
NUMĂRUL TESTĂRII	NUMĂR VÂRFURI GRAF	POPULATION_ SIZE	MAX_ GENERATIONS	MUTATION_ RATE	CROSSOVER_ RATE	ALPHA
1	100	20	1000	0.50	1.00	100
2	20	50	800	0.01	0.50	600
3	60	100	100	0.20	0.95	300
4	30	150	500	0.15	0.90	50
5	80	200	400	0.35	0.70	700
6	90	250	200	0.30	0.60	500
7	40	300	600	0.25	0.85	750

2.4.2 DPSO

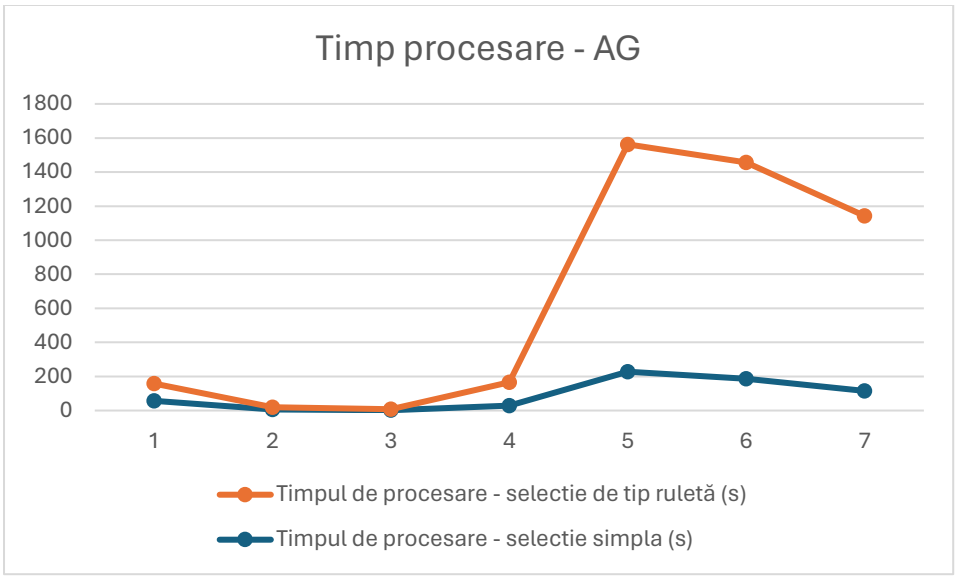
NUMĂRUL TESTĂRII	NUMĂR VÂRFURI GRAF	swarmSize	maxIterations	w	c1	c2	alpha
1	100	20	1000	0.5	1.05	-1.9	100
2	20	50	800	0.6	-1.5	1.5	600
3	60	100	100	0.8	1.5	0.5	300
4	30	150	500	0.75	-1.3	1.4	50
5	80	200	400	0.7	1.8	-1.4	700
6	90	250	200	0.45	-1.2	1.2	500
7	40	300	600	0.55	1.2	1.1	750

Capitolul 3. Prezentarea rezultatelor

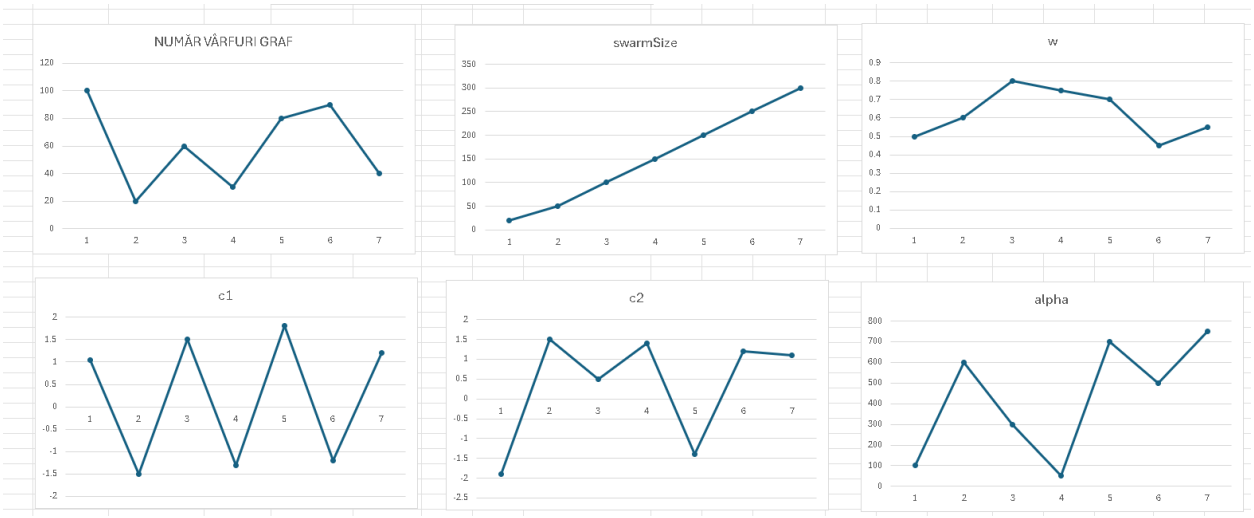
3.1. AG



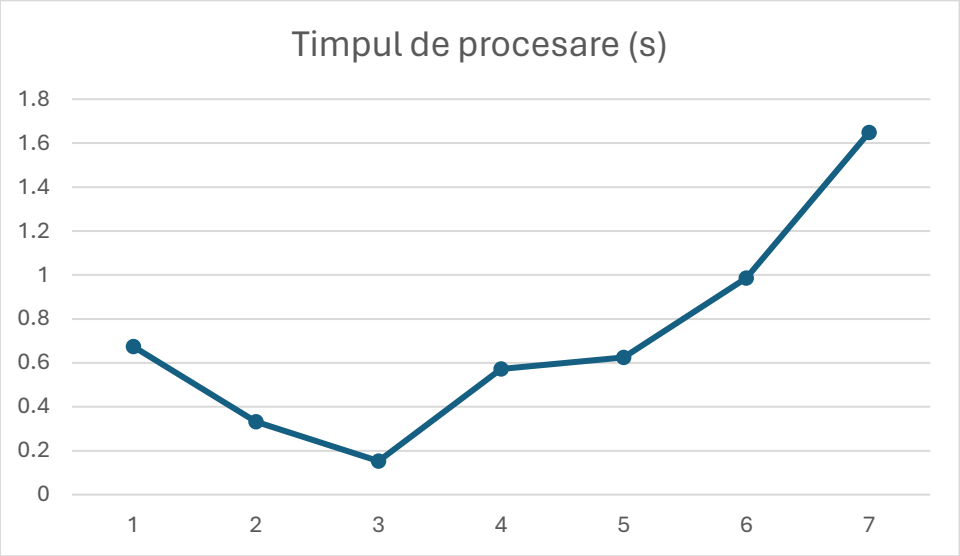
NUMĂRUL TESTĂRII	Timpul de procesare - selectie simpla (s)	Timpul de procesare - selectie de tip ruletă (s)
1	57.59	100.17
2	5.68	13.65
3	1.84	6.00
4	28.34	137.45
5	227.98	1334.50
6	185.98	1271.20
7	114.46	1027.64


Timp procesare - AG
1800
1600
1400
1200
1000
800
600
400
200
0
1
2
3
4
5
6
7
Timpul de procesare - selectie de tip ruletă (s)
Timpul de procesare - selectie simpla (s)

3.2. DPSO


NUMĂR VÂRFURI GRAF
swarmSize
w
c1
c2
alpha

NUMĂRUL TESTĂRII	Timpul de procesare (s)
1	0.674
2	0.332
3	0.153
4	0.572
5	0.624
6	0.986
7	1.648



Capitolul 4. Concluzii – interpretare și pași viitori

Interpretare

- în cazul algoritmului genetic, există o diferență mare între timpul de procesare rezultat în urma folosirii celor două metode pentru selecție. Astfel, selecția de tip ruletă are un timp mult mai mare, însă, totodată este și o metodă mai complexă și mai stabilă – cu rezultate mai reale și mai apropiate de adevăr.
- Ambii algoritmi depind de complexitate grafului : timpul de procesare scade in cazul AG și crește în cazul DPSO o dată cu scăderea numărului de vârfuri
- Mărimea populației nu influențează într-un mod vizibil rezultatele în nici unul din cei doi algoritmi, timpul de procesare scăzând chiar daca populația crește
- Rata de mutație în cazul algoritmului AG influențează rezultatele, fiind direct proporțională cu timpul de execuție
- Numărul de generații, rata de încrucișare și alpha, în cazul algoritmului AG au o influență inversă asupra rezultatului (când acestea cresc, timpul se reduce – testarea 6-7)
- Influență parametru w în cazul DPSO : când acesta crește, timpul de execuție scade, deci există o relație invers proporțională
- Influență parametru c_1 în cazul DPSO : este la ca și în cazul parametrului w
- Influență parametru c_2 în cazul DPSO : este la ca și în cazul parametrului w
- Influență parametru alpha în cazul DPSO : direct proporțională

Pași viitori

O analiză mai în detaliu se poate face urmărind mai multe testări și schimbând datele inițiale pe care acestea se bazează.

Capitolul 5. Webografie

https://research.ijcaonline.org/volume82/number4/pxc3892126.pdf

https://www.ijser.org/researchpaper/Optimal-Algorithm-for-Solving-Vertex-Cover-Problem-in-Polynomial-Time.pdf

https://www.researchgate.net/publication/343099050_Performance_Evaluation_of_Vertex_Cover_and_Set_Cover_Problem_using_Optimal_Algorithm

https://www.sciencedirect.com/science/article/pii/S2772662223000152?ref=pdf_download&fr=RR-2&rr=889b9c323c8029c4

https://www.researchgate.net/publication/282375697_Vertex_Cover_Problem-Revised_Approximation_Algorithm