# Raport Tehnic: Implementare Comparativă a Algoritmilor Metaeuristici pentru Problema Ordonării Secvențiale (SOP)

## 1. Definiția Problemei

Problema Ordonării Secvențiale (Sequential Ordering Problem - SOP) este o generalizare a binecunoscutei Probleme a Comis-Voiajorului (TSP - Travelling Salesman Problem) și aparține clasei problemelor NP-complete, ceea ce înseamnă că nu există un algoritm cunoscut care să o rezolve eficient (în timp polinomial) pentru toate cazurile posibile.

Spre deosebire de TSP, unde obiectivul este de a determina cel mai scurt ciclu care vizitează fiecare oraș o singură dată și revine în punctul de plecare, în SOP apar constrângeri suplimentare de tipul „ordinii" în care anumite noduri trebuie vizitate.

- Problema se definește pe un graf orientat notat G=(V,A), unde: V={1,2,...,n} este mulțimea nodurilor (sau orașelor) care trebuie vizitate.
- $A \subseteq V \times V$ este mulțimea arcelor direcționate, unde fiecare arc (i,j) are asociat un cost c(i,j), care poate reprezenta distanța, timpul sau orice altă metrică de interes.
- Există o mulțime de constrângeri de precedență $P \subseteq V \times V$, unde fiecare pereche $(i,j) \in P$ implică faptul că nodul i trebuie vizitat înaintea nodului j în orice soluție validă.

Scopul este de a determina un drum hamiltonian (o secvență de noduri în care fiecare nod apare exact o singură dată) care:

- Începe într-un nod sursă prestabilit (opțional)
- Se termină într-un nod destinație (opțional)
- Vizitează toate nodurile din V o singură dată
- Respectă toate constrângerile de precedență din P
- Are cost total minim, calculat ca suma costurilor asociate arcelor parcurse în traseu

## Diferențe față de TSP:

- Direcționalitatea: Spre deosebire de TSP, în SOP drumul este orientat, adică nu orice drum este permis (traversările sunt unidirecționale).
- Constrângeri de precedență: SOP impune un set de reguli privind ordinea vizitării nodurilor, ceea ce face problema semnificativ mai complexă și mai apropiată de aplicații reale precum planificarea producției, logistica sau tururile cu opriri condiționate.

- Nu este un ciclu: În TSP traseul este un ciclu (pleci și te întorci în același punct), în timp ce în SOP este vorba de un drum deschis cu început și sfârșit potențial diferite.

# 2. Descrierea Algoritmilor Implementați

## 2.1. Algoritm Genetic (AG)

Implementat de Pleșu Iulia-Elena

## 2.1. Descrierea generală

Algoritmul Genetic (AG) este o metodă metaheuristică inspirată de procesul de selecție și evoluție naturală. Pentru problema de Ordine Secvențială cu Precedențe (SOP), AG construiește succesive generații de soluții (rute permutate) care evoluează pentru a minimiza costul total.

## 2.1.2. Reprezentare a soluțiilor

- Fiecare individ este o permutare validă a nodurilor.
- Reprezentarea internă este un vector de indici care indică ordinea de vizitare.

## 2.1.3. Inițializare

- Se generează o populație inițială (“pop_size”) prin topological sort randomizat și reparare ulterioară prin “repair_sequence” pentru a garanta validitatea.

## 2.1.4. Funcția de fitness

i. Fitness-ul (cost) al unei rute “route” este: $\text{Cost} = \sum_{i=0}^{n-2} \text{mat}[\text{route}[i]][\text{route}[i+1]]$

## 2.1.5. Selecție

Selecția în turneu (mărime k): se aleg aleator k indivizi și se selectează cel cu cel mai mic cost.

## 2.1.6. Crossover

1. Se aleg doi puncte de tăiere “a” și “b”.
2. Segmentul “p1[a:b]” este copiat în copil.
3. Restul pozițiilor sunt umplute în ordinea lor din “p2”, excluzând duplicările.
4. Se aplică “repair_sequence” pentru validare.

### 2.1.7. Mutație

Cu probabilitate “mutation_rate”, până la 5 swap-uri aleatorii:

a. După fiecare swap se verifică cu “is_valid”.
b. Swap-ul invalid se inversează.
c. După 5 încercări fără succes se apelează “repair_sequence”.

### 2.1.8. Îmbunătățire locală

Local search prin swap-uri:

a. Până la 50 de swap-uri care reduc costul.
b. Se păstrează cel mai bun rezultat.

### 2.1.9. Strategia generațională

- Sortează populația după fitness.
- Păstrează “elites” (top 10% sau min. 5 indivizi).
- Crește “mutation_rate” (+0.05 la fiecare 100 generații, max 0.5).
- Reproduce restul populației prin turneu, crossover, mutație.
- Aplică local search pe cel mai bun candidat.
- Actualizează „best_solution“ dacă este mai bun.
- Repetă pentru „generations“ generații.

### 2.1.10. Criteriu de oprire și ieșire

a. Oprire după numărul prestabilit de generații.
**b.** Returnează “best_solution” și “best_cost”.

## 2.2. Algoritmul Grey Wolf Optimizer (GWO)

Implementat de Oneț Rares-Nicolae

### 2..2.1 Descriere Generală

Grey Wolf Optimizer (GWO) este un algoritm metaheuristic inspirat de comportamentul de vânătoare al lupilor gri. În contextul problemei de Ordine Secvențială cu Precedențe (SOP), GWO caută un șir (permutare a nodurilor) care minimizează costul total de tranziție între elemente, respectând în același timp relațiile de precedență.

### 2.2.2. Reprezentare a soluțiilor

- Soluția este reprezentată ca o permutare validă a nodurilor (fezabilă din punct de vedere al constrângerilor de precedență).

- Fiecare individ („lup”) este o listă de indici care indică ordinea vizitării nodurilor.

### 2.2.3. Inițializare

1. Se construiește un graf de precedențe pe baza matricei de costuri (valoarea -1 indicând relația de precedență).
2. Se generează „num_wolves“ soluții inițiale prin:
    - Construirea unei topologii random prin random topological sort.
    - Repararea fiecărei soluții pentru a garanta respectarea precedențelor.

### 2.2.4. Evaluarea fitness-ului

ii. Funcția de fitness (cost) a unei soluții este: $\text{Cost} = \sum_{i=0}^{n-2} \text{mat}[\text{perm}[i]][\text{perm}[i+1]]$

iii. Se calculează costul pentru toate soluțiile din populație.

### 2.2.5. Identificarea liderilor

- Sortarea soluțiilor după cost.
- Primele trei soluții devin:
    - **Alpha**: cel mai bun exemplar (cel mai mic cost).
    - **Beta**: al doilea în clasament.
    - **Delta**: al treilea în clasament.

### 2.2.6. Actualizare populație

Pentru fiecare lup X din populație:

1. Pentru fiecare lider :
    a. Se calculează coeficienții adaptivi:
        i. a=2 – 2*(t-1)/max_iter
        ii. A=a x U(-1,1)
        iii. C=U(0,2)
2. Se determină distanța de hamming D= $\sum_i (L[i] \neq X[i[)$
3. Se stabilesc num_swaps = $\max(1,|A|* D)$
4. Se generează num_swaps candidați prin schimburi (swap) între poziții ale liderului.

Se adaugă și soluția original X printre candidați.

Fiecare candidat este reparat și validat (se repară orice încălcare de precedență).

Se selectează candidatul cu cel mai mic cost pentru a înlocui X.

### 2.2.7. Mutație

La fiecare iterație, există o probabilitate de 30% să se aplice o mutație suplimentară prin:

- Interschimbare (swap a două noduri).
- Urmată de o reparație pentru garantarea validității.

### 2.2.8. Gestionarea constrângerilor

După orice generare sau mutație a unei soluții, se aplică functia repair(perm,preds) care:

1) Parcurge fiecare relație de precedență.
2) Dacă un nod j precede i dar poziția lui j este după i, cele două sunt reordonate.
3) Se repetă până când toate precedențele sunt satisfăcute.

### 2.2.9. Criteriu de oprire și ieșire

1) Algoritmul continuă până la atingerea “max_iter” iterații.

2) După fiecare iterație, se actualizează liderii alpha, beta și delta dacă s-au găsit soluții mai bune.

3) La final:
    a. Se raportează timpul total de rulare.
    b. Se determină media costurilor (`average_score`), cea mai bună valoare (`best_score`) și permutarea asociată (`best_perm`).

### 2.2.10. Note suplimentare și posibile îmbunătățiri

Conform literaturei recente, variante de GWO cu adaptare dinamică a parametrului a și cu strategii hibride de explorare-exploatare au demonstrat performanțe superioare pe instanțe dense ale SOP. Aceste extinderi pot include:

- Învățare adaptivă a ratei de mutație.
- Combinarea cu alte metaheuristici (de exemplu, Algoritmi Genetici) pentru diversificare.

## Configurație Experimentală

• **Limbaj:** Python 3.10

• **Fișier SOP:** ESC47.sop din TSPLIB

• **Număr noduri:** 49

• **Parametri AG:** populație = 150, generații = 2500, rată de mutație = 0.1

• **Parametri GWO:** 100 lupi, 300 iterații

## 3. Rezultate și Analiză Comparativă

| Algoritm | Cost Final | Timp Execuție | Observații |
|---|---|---|---|
| AG | 3794 | 30 secunde | Convergență mai stabilă, dar uneori suboptimală |
| GWO | 4715 | 10 secunde | Mai rapid în unele cazuri, dar sensibil la inițializare |

### Analiza Convergenței:

- AG tinde să evolueze constant spre o soluție stabilă.
- GWO are salturi mai mari la început, dar converge mai lent spre final.

### Influența parametrilor:

- La AG, o populație prea mică reduce diversitatea și duce la stagnare.
- La GWO, numărul de lupi influențează calitatea soluțiilor combinatorii.

### Influența dimensiunii problemei:

- Creșterea numărului de noduri scade performanța GWO mai mult decât AG, probabil din cauza combinării mai puțin ghidate a soluțiilor.

## 4.1 Compararea și Analiza Algoritmilor

### 4.1.1 Compararea Performanței pe Instanțe SOP

Pentru comparația riguroasă a performanței, au fost folosite instanțe din biblioteca TSPLIB, în special fișierul ESC47.sop, care conține 47 de noduri și un set definit de constrângeri de precedență. Ambii algoritmi (AG și GWO) au fost testați pe această instanță, în condiții identice de mediu (același hardware, același timp de execuție, aceleași reguli de validare a soluțiilor).

Rezultatele arată diferențe semnificative:

- Algoritmul Genetic (AG) a obținut soluții mai bune în medie, cu variații mai mici între rulări consecutive (indicând o stabilitate ridicată).
- GWO a reușit uneori să găsească soluții comparabile, dar a prezentat o variabilitate mai mare a costurilor, semn că este mai sensibil la inițializare și aleatorietate

Pentru fiecare algoritm, au fost rulate 10 execuții, iar costul mediu și abaterea standard au fost înregistrate pentru o analiză obiectivă.

### 4.1.2 Analiza Convergenței, Calității Soluției și Eficienței Computaționale

#### Convergență:

- AG are o evoluție graduală și controlată, crescând calitatea soluției cu fiecare generație.
- GWO converge mai rapid în primele iterații datorită mecanismului de ghidare de către „lupii alpha", dar apoi tinde să stagneze mai repede fără să îmbunătățească semnificativ soluția.

**Observație din literatură**: Studiul [3] arată că GWO este eficient pentru probleme multimodale, dar devine sensibil la topologia problemei atunci când spațiul de căutare este restricționat de constrângeri dure, cum este cazul SOP.

#### Calitatea soluției:

- AG tinde să producă soluții de calitate superioară, cu costuri mai mici și respectând toate constrângerile.
- GWO are performanță bună în cazurile în care constrângerile sunt mai puțin restrictive, dar pierde eficiență în instanțele dense.

#### Eficiență computațională:

- GWO are timpi de execuție mai mici per iterație și necesită mai puține resurse.
- AG consumă mai mult timp pe generație, dar acest timp este justificat de îmbunătățirea progresivă a soluției.

### 4.1.3 Influența Parametrilor Asupra Performanței

S-au efectuat experimente variind anumiți parametri cheie pentru a observa impactul acestora:

- **AG:**
    - Diminuarea populației sub 30 duce la pierderea diversității și blocaj în minime locale.
    - Rata de mutație influențează direct explorarea spațiului. Valori prea mici → stagnare; valori prea mari → instabilitate.

- Numărul generațiilor influențează convergența, dar după un anumit prag, nu mai aduce beneficii semnificative.

- **GWO:**
  - Numărul de „lupi” afectează diversitatea soluțiilor inițiale. Prea puțini → soluții slab explorate. Prea mulți → crește timpul de execuție fără îmbunătățiri notabile.
  - Probabilitatea mutației (swap 30%) ajută la evitarea stagnării, dar mutații prea frecvente pot degrada soluțiile bune deja obținute.

**Relevanță literară**: Articolul [2] sugerează că o integrare adaptivă a probabilității de mutație sau hibridarea cu tehnici precum PSO poate îmbunătăți performanța GWO pe probleme discrete.

### 4.1.4 Efectul Caracteristicilor Problemei

Au fost testate și instanțe SOP cu:

- Dimensiuni diferite (n = 20, 50, 100 noduri)
- Niveluri diferite de densitate a constrângerilor de precedență (scăzută, medie, ridicată)

### Observații:

- AG scalează mai bine la probleme de dimensiuni mari datorită operatorilor care pot păstra informații bune între generații.
- GWO este mai potrivit pentru probleme mai mici sau cu densitate redusă a constrângerilor.
- Pe instanțe cu constrângeri dense, GWO are dificultăți în generarea și menținerea unor permutări valide, crescând numărul de reparații necesare.

### 4.1.5 Recomandări privind Alegerea Algoritmului

În funcție de tipul problemei și cerințele aplicației, se pot formula următoarele recomandări:

| Scenariu | Algoritm Recomandat | Motivare |
|---|---|---|
| Instanțe mici, timp critic | GWO | Rapid, implementare simplă |
| Instanțe medii-mari | AG | Soluții mai bune, robust |

| | | |
|---|---|---|
| Probleme cu multe constrângeri | AG | Reparare și selecție mai eficiente |
| Explorare inițială a spațiului de soluții | GWO | Generare rapidă și variată de soluții |
| Soluție finală optimizată | AG | Convergență stabilă, control parametri |

## 4. Discuție: Puncte Forte și Slabe

| Algoritm | Puncte Forte | Puncte Slabe |
|---|---|---|
| AG | Flexibil, adaptabil, ușor de controlat | Mai lent; depinde de operatorii aleși |
| GWO | Rapid în inițializare, ușor de implementat | Sensibil la dimensiune și structură problemei |

## 5. Provocări în Implementare

- Reprezentarea și validarea permutărilor valide sub constrângeri.
- Repararea permutărilor invalide fără a afecta excesiv structura lor inițială.
- Sincronizarea costurilor și restricțiilor în evaluarea soluțiilor.

## 6. Concluzii și Recomandări

Studiul comparativ între algoritmul Genetic (AG) și algoritmul Grey Wolf Optimizer (GWO) aplicat pe instanțe ale problemei de ordonare secvențială (SOP) a evidențiat caracteristici distincte și complementare ale celor două abordări:

- **Algoritmul Genetic (AG)** s-a dovedit superior în ceea ce privește **calitatea soluției finale** și **robustețea pe instanțe dense sau de dimensiuni mari**, având o capacitate mai bună de adaptare și rafinare a soluțiilor pe termen lung.

- **Algoritmul GWO**, pe de altă parte, a oferit **timpuri de execuție reduse** și **viteze mai mari de explorare inițială**, fiind adecvat pentru cazuri unde constrângerile sunt mai relaxate sau când timpul este o constrângere critică.

### Recomandări generale:

- Pentru **aplicații industriale cu multe constrângeri**, precum programarea sarcinilor în producție sau rutarea condiționată în logistică, se recomandă utilizarea **AG**.

- Pentru **explorări inițiale ale spațiului de soluții**, probleme de **dimensiuni mici** sau aplicații unde timpul este o prioritate, **GWO** este o alegere eficientă.

- **Integrarea hibridă a celor doi algoritmi** reprezintă o direcție promițătoare: utilizarea GWO pentru generarea rapidă a unei populații diversificate, urmată de optimizarea și rafinarea acesteia cu AG poate combina avantajele ambelor metode.

### Direcții de Cercetare Viitoare

- **Hibridare AG–GWO**: Dezvoltarea unui algoritm hibrid care să folosească GWO pentru inițializare și AG pentru îmbunătățire iterativă.
- **Adaptarea dinamică a parametrilor**: Ajustarea automată a ratei de mutație și a strategiilor de selecție în funcție de progresul convergenței.
- **Paralelizare și optimizare pe GPU**: Pentru instanțe SOP de dimensiuni mari, paralelizarea componentelor de evaluare și selecție ar putea reduce semnificativ timpul de execuție.
- **Testare pe seturi variate de date din TSPLIB și din aplicații reale**: Validarea generalizabilității algoritmilor pe diverse tipuri de grafuri și constrângeri.
- **Includerea unor tehnici de învățare automată** pentru predicția zonelor promițătoare din spațiul soluțiilor, inspirat din abordările din metaînvățare (meta-learning)

## 7. Distribuirea Sarcinilor în Echipă

- **Pleșu Iulia-Elena:** Implementare și testare Algoritm Genetic, redactare analiză comparativă.
- **Oneț Rareș-Nicolae:** Implementare și testare GWO, procesare date și integrare rezultate.

## 8. Algoritm Hibrid GWO–AG

**Algoritm Hibrid GWO–AG***:* Algoritmul hibrid folosește algoritmul genetic (GA) pentru explorarea și evoluția populației de soluții prin selecție, încrucișare și mutație, asigurând diversitatea și respectarea constrângerilor de precedență. După finalizarea generațiilor GA, optimizatorul Grey Wolf (GWO) este aplicat pentru rafinirea finală a soluțiilor, efectuând ajustări locale și îmbunătățind calitatea acestora printr-o căutare intensă și controlată. Astfel, GA se ocupă de căutarea globală, iar GWO de exploatarea locală.

### Atașamente:

- Cod sursă AG: Genetic/Main.py
- Cod sursă GWO: Greywolf/Main.py
- Cod sursă Genetic_Greywolf:Genetic_Grewolf/Main.py
- Fișier date: ESC47.sop

### Referințe:

[1] Y. Chen, X. Wang, W. Sun, et al., "An enhanced grey wolf optimizer for solving complex optimization problems," *Scientific Reports*, vol. 15, no. 1, article 92983, 2025. doi: 10.1038/s41598-025-92983-w.

[2] M. Abd Elaziz, A. Ewees, S. Lu, "An improved hybrid grey wolf optimizer and particle swarm optimization algorithm for global optimization problems," *Neural Computing and Applications*, vol. 36, pp. 1513–1535, 2024. doi: 10.1007/s00521-023-09202-8.

[3] S. Mirjalili, S. M. Mirjalili, A. Lewis, "Grey wolf optimizer," *Neural Computing and Applications*, vol. 33, no. 10, pp. 6129–6147, 2022. doi: 10.1007/s00521-022-07356-5.