*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⊆V×VA \subseteq V \times VA⊆V×V este mulțimea arcelor direcționate, unde fiecare arc (i,j)(i, j)(i,j) are asociat un cost c(i,j)c(i, j)c(i,j), care poate reprezenta distanța, timpul sau orice altă metrică de interes.
* Există o mulțime de constrângeri de precedență P⊆V×VP \subseteq V \times VP⊆V×V, unde fiecare pereche (i,j)∈P(i, j) \in P(i,j)∈P implică faptul că nodul iii trebuie vizitat înaintea nodului jjj 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 PPP,
* Și 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

**Reprezentare:** Soluțiile sunt permutări ale nodurilor, verificate și reparate astfel încât să respecte constrângerile de precedență.

**Operatori genetici:**

* **Selecție:** Turneu (Tournament Selection) cu k = 3.
* **Recombinare:** Order Crossover (OX), urmat de o fază de reparare pentru a asigura respectarea constrângerilor.
* **Mutație:** Swap aleator între două poziții, urmat de reparare dacă este necesar.

**Fitness:** Costul total al traseului generat, calculat din matricea de distanțe.

**Gestionarea constrângerilor:** Toate permutările sunt validate sau reparate pentru a respecta precedențele.

**2.2. Algoritmul Grey Wolf Optimizer (GWO)**  
Implementat de Oneț Rares-Nicolae

**Reprezentare:** Soluțiile sunt permutări valide ale nodurilor.

**Inițializare:** Generare de soluții valide prin încercări repetate și verificarea respectării constrângerilor.

**Actualizare populație:** Cele mai bune trei soluții sunt “alfa”, “beta” și “delta”. Soluțiile noi sunt construite combinând informațiile din aceste trei.

**Mutare:** Cu o probabilitate de 30%, se aplică o mutație prin interschimbarea a două noduri urmată de validare.

**Fitness:** Totalul costurilor de-a lungul traseului propus.

**Gestionarea constrângerilor:** Reparații și verificări după fiecare generare sau mutație.

**3. Configurație Experimentală**

* Limbaj: Python 3.10
* Fișier SOP: ESC47.sop din TSPLIB
* Număr noduri: 49
* Parametri AG: populație = 50, generații = 200, rată de mutație = 0.1
* Parametri GWO: 35 lupi, 200 iteratii

**4. Rezultate și Analiză Comparativă**

| **Algoritm** | **Cost Final** | **Timp Execuție** | **Observații** |
| --- | --- | --- | --- |
| AG | 5410 | 43 secunde | Convergență mai stabilă, dar uneori suboptimală |
| GWO | 4668 | 22 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. Acest lucru este datorat presiunii selecției și efectului operatorilor genetici (recombinare + mutație).
* GWO converge **mai rapid în primele iteratii** datorită mecanismului de ghidare de către „lupii alpha”, dar apoi tinde să **stagneze mai repede** fără să îmbunătățească semnificativ soluția.

**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.

**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 |

**5. 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 |

**6. 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.

**7. Concluzii și Recomandări**

* Pentru instanțe SOP de dimensiuni medii, algoritmul genetic oferă o mai bună robustete și calitate generală.
* GWO poate fi o alegere bună pentru explorare rapidă în fazele inițiale sau ca parte dintr-un algoritm hibrid.
* Recomandăm utilizarea AG în aplicații unde acuratețea soluției este prioritară și GWO când timpul de execuție este critic.

**8. 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.

**Atașamente:**

* Cod sursă AG: Genetic\_alg.py
* Cod sursă GWO: Greywolf.py
* Fișier date: ESC47.sop