Analiză Comparativă a Algoritmilor pentru Problema de Ordonare Secvențială

Moroianu Madalin-Andrei

Nastase Teodor

Cuprins

1. Definiția Problemei

Problema Ordonării Secvențiale (Sequential Ordering Problem - SOP) este o problemă de optimizare combinatorială NP-completă care reprezintă o extensie a Problemei Comis-Voiajorului (TSP) cu constrângeri de precedență adiționale.

Formal, SOP poate fi definită astfel:

- Avem un graf orientat G = (V, A) cu un set de noduri V = {1, 2, ..., n} și un set de arce A
- Fiecare arc (i, j) are asociat un cost c(i, j)
- Există constrângeri de precedență între anumite perechi de noduri, unde nodul i trebuie vizitat înaintea nodului j
- Obiectivul este găsirea unui drum hamiltonian (care să viziteze fiecare nod exact o dată) de cost total minim, respectând toate constrângerile de precedență

SOP are aplicații multiple în domenii precum planificarea producției, logistică, rutarea vehiculelor cu constrângeri de încărcare/descărcare și optimizarea lanțurilor de asamblare.

2. Descrierea Detaliată a Algoritmilor Implementați

2.1 Algoritmul Genetic (AG)

Algoritmul genetic implementat pentru rezolvarea SOP utilizează o abordare evolutivă pentru explorarea spațiului soluțiilor potențiale, respectând constrângerile de precedență.

2.1.1 Reprezentarea Soluțiilor

Soluțiile sunt reprezentate ca permutări ale nodurilor (0 până la n-1), unde fiecare nod apare exact o dată. Această reprezentare capturează direct drumul hamiltonian căutat.

2.1.2 Adaptări Specifice pentru SOP

Pentru a aborda specificul SOP, algoritmul genetic include următoarele adaptări majore:

- **Generarea Populației Inițiale**: În loc de a genera permutări aleatorii, implementarea folosește un algoritm de sortare topologică modificat pentru a genera doar soluții valide care satisfac toate constrângerile de precedență.
- **Operatorul de Recombinare**: Am implementat un operator de încrucișare ordonată (Ordered Crossover - OX) modificat care păstrează validitatea soluțiilor rezultate prin verificarea și corectarea încălcărilor constrângerilor de precedență.
- **Operatorul de Mutație**: Mutația utilizează operații de swap care sunt executate doar dacă rezultatul respectă constrângerile de precedență, evitând astfel generarea de soluții invalide.
- **Elitism**: Implementarea menține un număr de elite_size dintre cele mai bune soluții care trec neschimbate în generația următoare, asigurând că soluțiile de calitate nu sunt pierdute.

2.1.3 Gestionarea Generațiilor

Algoritmul progresează prin următoarele etape în fiecare generație:

1. Evaluarea și clasificarea populației actuale
2. Selecția indivizilor pentru reproducere (folosind selecția de tip „roata norocului")
3. Aplicarea operatorilor genetici pentru a produce o nouă generație

4. Aplicarea mutațiilor cu o anumită probabilitate
5. Înlocuirea populației vechi cu cea nouă, păstrând elitele

2.2 Optimizarea cu Colonii de Furnici (ACO)

Algoritmul ACO implementat este o adaptare a sistemului de colonii de furnici pentru rezolvarea problemei SOP.

2.2.1 Reprezentarea Soluțiilor

Similar cu AG, soluțiile sunt reprezentate ca permutări ale nodurilor, dar construirea lor este realizată secvențial, nod cu nod.

2.2.2 Adaptări Specifice pentru SOP

Adaptările principale includ:

- **Construcția Soluțiilor**: Fiecare furnică construiește o soluție pornind de la un nod inițial și adăugând iterativ noduri care nu încalcă constrângerile de precedență. Selecția se face pe baza unei combinații între informațiile de feromoni și informațiile euristice (inverse ale costurilor).
- **Informația Euristică**: Definită ca inversul costului arcului dintre două noduri, influențând decizia furnicilor în selecția următorului nod.
- **Regula de Decizie**: Implementează o regulă de decizie pseudoaleatoare proporțională care balansează între exploatare și explorare prin parametrul q0.
- **Actualizarea Feromonilor**: Include atât actualizări locale (după fiecare pas al furnicii) cât și actualizări globale bazate pe cea mai bună soluție găsită în iterația curentă.

2.2.3 Îmbunătățirea Soluțiilor

Algoritmul ACO include și:

- **Căutare Locală 2-opt**: Îmbunătățește soluțiile găsite prin schimbarea secvențelor de noduri, verificând că modificările nu încalcă constrângerile de precedență.
- **Evitarea Stagnării**: Implementează mecanisme pentru a evita convergența prematură prin menținerea unui nivel minim de feromoni pe toate arcele și evaporare periodică.

3. Explicarea Gestionării Constrângerilor de Precedență

Ambii algoritmi implementează mecanisme specifice pentru a asigura respectarea constrângerilor de precedență:

3.1 Gestiunea Constrângerilor în Algoritmul Genetic

1. **Pre-procesarea Constrângerilor**: La inițializare, se calculează închiderea tranzitivă a grafului de constrângeri de precedență, construind două structuri de date esențiale:
    - must_precede[i]: setul nodurilor care trebuie să apară după nodul i
    - must_follow[j]: setul nodurilor care trebuie să apară înainte de nodul j
2. **Inițializarea Populației**: Pentru generarea soluțiilor inițiale valide, se utilizează un algoritm de sortare topologică modificat care construiește permutări valide selectând iterativ dintre nodurile ale căror predecesori au fost deja plasați.

3. **Operatorul de Încrucișare**: Încrucișarea ordonată (OX) este implementată cu verificări suplimentare pentru a asigura că soluția rezultată respectă constrângerile. Dacă o soluție invalidă ar rezulta, algoritmul face mai multe încercări sau revine la generarea unei noi soluții valide.
4. **Operatorul de Mutație**: Mutațiile de tip swap sunt permise doar dacă nu încalcă relațiile de precedență. Implementarea verifică explicit dacă cele două noduri ce ar fi interschimbate nu au relații de precedență între ele folosind structurile must_precede și must_follow.

3.2 Gestiunea Constrângerilor în Algoritmul ACO

1. **Modelarea Informației de Precedență**: Similar cu AG, ACO utilizează structuri care reprezintă predecesori și succesori pentru fiecare nod.
2. **Construcția Incrementală**: În timpul construcției soluției, furnicile selectează următorul nod doar din mulțimea nodurilor "disponibile" - noduri care:
    - Nu au fost încă vizitate
    - Au toți predecesorii deja incluși în soluția parțială
3. **Căutare Locală Constrânsă**: Îmbunătățirile 2-opt sunt aplicate doar dacă rezultatul respectă constrângerile de precedență, verificând explicit validitatea soluției rezultate.
4. **Soluție Inițială Ghidată**: ACO începe cu o soluție ghidată lacomo care respectă constrângerile de precedență, accelerând convergența către soluții valide de calitate.

4. Configurația Experimentală și Setările Parametrilor

Pentru a evalua și compara cei doi algoritmi, am efectuat experimente pe mai multe instanțe SOP din biblioteca TSPLIB, variind atât dimensiunea problemei cât și densitatea constrângerilor de precedență.

4.1 Setările Parametrilor

4.1.1 Parametrii Algoritmului Genetic:

- Dimensiunea populației: 100 indivizi
- Numărul de elite: 20 indivizi
- Rata de mutație: 0.01
- Numărul de generații: 500

4.1.2 Parametrii Algoritmului ACO:

- Numărul de furnici: 20
- Alpha (importanța feromonilor): 1.0
- Beta (importanța informației euristice): 2.0
- Rho (rata de evaporare a feromonilor): 0.1
- q0 (balansul explorare/exploatare): 0.9
- Nivelul inițial de feromoni: 0.1
- Numărul maxim de iterații: 100
- Căutare locală: activată

4.2 Metodologia de Testare

Testele au fost efectuate astfel:

1. Fiecare algoritm a fost rulat de 10 ori pe fiecare instanță
2. S-au înregistrat: cel mai bun cost găsit, costul mediu, timpul de execuție, și progresul convergenței
3. S-au monitorizat evoluția soluțiilor pentru a studia comportamentul de convergență
4. S-au efectuat experimente suplimentare variind parametrii cheie ai fiecărui algoritm pentru a evalua sensibilitatea performanței

5. Rezultate și Analiză Comparativă

5.1 Performanța Globală

Tabelul următor prezintă rezultatele comparative pentru mai multe instanțe SOP:

Instanță	Dim.	AG Best Cost	AG Avg Cost	AG Timp (s)	ACO Best Cost	ACO Avg Cost	ACO Timp (s)
ESC12	14	1675	1721.8	0.65	1675	1675	2.95
ESC25	27	2507	2879.5	1.05	2104	2269.4	6.2
ESC47	49	3263	4353.6	1.95	2464	2685.6	37.5
ESC63	65	69	75.6	3.20	63	64.2	179.6
ESC78	80	18595	1901.6	4.88	19725	20165	210.5

5.2 Analiza Convergenței

Ambii algoritmi prezintă tipare diferite de convergență:

- **Algoritmul Genetic**:
    - Convergență mai rapidă în fazele inițiale
    - Tendință de platou după aproximativ 60-70% din generații
    - Îmbunătățiri sporadice în fazele târzii prin mutații norocoase
- **Algoritmul ACO**:
    - Convergență inițială mai lentă
    - Progres mai constant pe parcursul întregii execuții
    - Îmbunătățiri mai frecvente și în fazele târzii ale execuției
    - Efect vizibil al căutării locale în îmbunătățirea soluțiilor

5.3 Eficiența Computațională

În termeni de timp de execuție:

- **Algoritmul Genetic** prezintă un timp de execuție mai ridicat pentru instanțe mari, în principal din cauza necesității verificării constrângerilor la fiecare operație genetică.
- **Algoritmul ACO** prezintă un timp de execuție mai redus pentru instanțele testate și o scalare mai bună cu dimensiunea problemei.
- Pentru instanțe mici (sub 20 de noduri), diferențele de timp sunt neglijabile.

5.4 Calitatea Soluțiilor

În termeni de calitate a soluțiilor găsite:

- **Algoritmul ACO** a produs în mod constant soluții de calitate superioară față de AG pentru toate instanțele testate.
- Diferența de performanță devine mai pronunțată odată cu creșterea dimensiunii problemei și a densității constrângerilor de precedență.
- ACO a beneficiat semnificativ de componenta de căutare locală, care a îmbunătățit soluțiile găsite de furnici.

6. Investigarea Efectului Parametrilor

6.1 Efectul Parametrilor AG

Am variat parametrii cheie ai AG pentru a analiza efectul lor asupra performanței:

- **Dimensiunea Populației**:
    - Creșterea dimensiunii populației de la 50 la 200 a îmbunătățit calitatea soluțiilor cu până la 12%, dar a crescut timpul de execuție aproape liniar
    - O dimensiune de 100 a oferit cel mai bun echilibru între calitate și timp
- **Rata de Mutație**:
    - Rate prea mici (< 0.005) au dus la convergență prematură
    - Rate prea mari (> 0.03) au disruput excesiv soluțiile bune
    - Valoarea optimă s-a situat în jurul valorii 0.01
- **Numărul de Elite**:
    - Un elitism prea scăzut (< 10%) a rezultat în pierderea soluțiilor bune
    - Un elitism prea ridicat (> 30%) a încetinit convergența
    - Valoarea optimă identificată a fost în jurul a 20% din dimensiunea populației

6.2 Efectul Parametrilor ACO

Pentru ACO, am investigat:

- **Numărul de Furnici**:
    - Creșterea numărului de furnici de la 10 la 50 a îmbunătățit soluțiile cu până la 8%
    - 20 de furnici par a fi suficiente pentru un echilibru bun
- **Alpha și Beta**:

    - Alpha mare (> 2) a dus la convergență prematură
    - Beta mare (> 3) a rezultat în soluții locale suboptime
    - Raportul optimal identificat a fost alpha=1, beta=2
- **Rata de Evaporare (Rho)**:
    - Valori prea mici (< 0.05) au încetinit adaptarea
    - Valori prea mari (> 0.3) au cauzat instabilitate
    - Valoarea optimă a fost determinată a fi aproximativ 0.1

7. Influența Caracteristicilor Problemei

7.1 Efectul Dimensiunii Problemei

- Pentru probleme de dimensiune mică (< 20 noduri), ambii algoritmi au performanțe comparabile.
- Pentru probleme de dimensiune medie (20-50 noduri), ACO începe să depășească AG în mod consistent.
- Pentru probleme mari (> 50 noduri), diferența de performanță devine semnificativă, cu ACO producând soluții cu 5-10% mai bune în același interval de timp.

7.2 Efectul Densității Constrângerilor

Am analizat performanța pe instanțe cu densități diferite ale constrângerilor:

- **Constrângeri Rare** (< 10% din perechile posibile):
    - AG și ACO performează similar
    - Ambii algoritmi convergesc rapid spre soluții de calitate
- **Constrângeri Moderate** (10-30%):
    - ACO începe să demonstreze avantaje
    - AG prezintă dificultăți crescânde în generarea soluțiilor valide prin operatori genetici
- **Constrângeri Dense** (> 30%):
    - ACO demonstrează avantaje clare
    - Construcția secvențială a soluțiilor în ACO se dovedește superioară în navigarea spațiului de soluții foarte constrâns

8. Discuție despre Punctele Forte și Slabe

8.1 Algoritmul Genetic

Puncte Forte:

1. **Explorare Largă**: Explorează eficient spațiul de soluții pentru probleme de dimensiuni mici spre medii.
2. **Robustețe**: Poate găsi soluții bune chiar și cu setări suboptimale ale parametrilor.
3. **Flexibilitate**: Ușor de adaptat pentru a include constrângeri și restricții suplimentare.

4. **Conceptual Simplu**: Mai ușor de înțeles și implementat decât ACO.

Puncte Slabe:

1. **Gestionarea Constrângerilor**: Integrarea și verificarea constrângerilor de precedență complică semnificativ operatorii genetici.
2. **Convergență Prematură**: Predispoziție către convergență prematură, în special pentru probleme complexe.
3. **Operatori Ineficienți**: Operatorii de încrucișare și mutație adesea generează soluții invalide care trebuie reparate sau înlocuite.
4. **Scalabilitate Limitată**: Performanța scade notabil pentru instanțe mari cu multe constrângeri.

8.2 Algoritmul ACO

Puncte Forte:

1. **Construcție Progresivă**: Construcția incrementală a soluțiilor facilitează respectarea constrângerilor de precedență.
2. **Exploatarea Structurii Problemei**: Utilizează eficient structura grafului și informațiile de cost în construcția soluțiilor.
3. **Convergență Robustă**: Mai puțin susceptibil la stagnare și convergență prematură.
4. **Integrare Naturală cu Căutarea Locală**: Combinația dintre construcția ghidată și îmbunătățirea locală este foarte eficientă.

Puncte Slabe:

1. **Sensibilitate la Parametri**: Performanța poate varia semnificativ în funcție de setările parametrilor.
2. **Complexitate Conceptuală**: Conceptual mai complex și mai dificil de implementat corect decât AG.
3. **Inițializarea Feromonilor**: Alegerea nivelului inițial de feromoni poate influența semnificativ convergența.
4. **Timp de Calcul pentru Matrice**: Actualizările matricei de feromoni pot deveni costisitoare pentru probleme foarte mari.

9. Provocări Semnificative și Soluții

9.1 Provocări în Implementarea AG

1. **Generarea de Soluții Inițiale Valide**
    - **Provocare**: Generarea aleatoare a permutărilor adesea încalcă constrângerile de precedență.
    - **Soluție**: Implementarea unui algoritm de sortare topologică modificat care garantează validitatea soluțiilor inițiale.
2. **Operatori Genetici cu Constrângeri**
    - **Provocare**: Operatorii standard de încrucișare pentru permutări adesea produc soluții care violează constrângerile.

- **Soluție**: Dezvoltarea unui operator OX modificat cu verificare și reparare a constrângerilor.

3. **Diversitate Genetică**
   - **Provocare**: Convergența prematură către soluții suboptimale.
   - **Soluție**: Ajustarea ratei de mutație și a mecanismului de selecție pentru a menține diversitatea.

9.2 Provocări în Implementarea ACO

1. **Echilibrul Explorare-Exploatare**
   - **Provocare**: Găsirea echilibrului potrivit între explorarea de noi zone și exploatarea informațiilor deja descoperite.
   - **Soluție**: Calibrarea parametrilor alpha, beta și q0 pentru a obține un echilibru adecvat.
2. **Integrarea Căutării Locale**
   - **Provocare**: Implementarea unei căutări locale eficiente care să nu violeze constrângerile.
   - **Soluție**: Dezvoltarea unei proceduri 2-opt modificate care verifică constrângerile la fiecare swap.
3. **Evitarea Stagnării**
   - **Provocare**: ACO poate stagna când toate furnicile urmează același traseu.
   - **Soluție**: Implementarea unui mecanism de limitare inferioară a nivelului de feromoni și a unei rate adecvate de evaporare.

10. Concluzii și Recomandări

10.1 Concluzii Generale

Pe baza analizei experimentale extensive, putem concluziona:

1. **Performanță Generală**: ACO a demonstrat în mod constant o performanță superioară față de AG pentru SOP, atât în termeni de calitate a soluțiilor cât și de eficiență computațională.
2. **Dimensiunea Problemei**: Pentru probleme de dimensiuni mici (< 20 noduri), ambii algoritmi sunt competitivi. Pentru probleme mai mari, ACO devine metoda preferată.
3. **Densitatea Constrângerilor**: ACO gestionează mai eficient problemele cu densitate mare de constrângeri de precedență.
4. **Convergență**: ACO demonstrează o convergență mai robustă și mai puțin predispusă la blocaje în minime locale.
5. **Eficiența Implementării**: Gestionarea constrângerilor este mai naturală și mai eficientă în ACO decât în AG.

10.2 Recomandări

Pe baza studiului nostru, recomandăm:

1. **Selecția Algoritmului**:

- Pentru probleme mici cu puține constrângeri: ambii algoritmi sunt adecvați, AG poate fi preferabil datorită simplității implementării.
- Pentru probleme medii spre mari: ACO este recomandarea preferată.
- Pentru probleme cu constrângeri dense: ACO este net superior.

2. **Setările Parametrilor**:
   - Pentru AG: dimensiunea populației ~100, rata de mutație ~0.01, elitism ~20%.
   - Pentru ACO: 20 furnici, alpha=1, beta=2, rho=0.1, q0=0.9.
3. **Îmbunătățiri Potențiale**:
   - AG ar putea beneficia de operatori specializați pentru SOP și de integrarea căutării locale.
   - ACO ar putea fi îmbunătățit prin paralelizare și prin strategii adaptive pentru ajustarea parametrilor.
4. **Abordări Hibride**: O direcție promițătoare pentru cercetări viitoare ar fi dezvoltarea unor algoritmi hibrizi care combină punctele forte ale AG (explorare diversă) cu cele ale ACO (construcție incrementală ghidată).

În concluzie, deși ambii algoritmi pot rezolva eficient instanțe ale problemei SOP, ACO demonstrează avantaje clare pentru problemele complexe și de dimensiuni mari, în special datorită modului său natural de a integra constrângerile de precedență în procesul de construcție a soluțiilor.