UNIVERSITATEA DE VEST DIN TIMIŞOARA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
PROGRAMUL DE STUDII DE LICENŢĂ: INFORMATICĂ

LUCRARE DE LICENŢĂ

COORDONATOR:
Lect. Dr. Liviu Octavian Mafteiu-Scai

ABSOLVENT:
Crivoi Mihail

TIMIŞOARA
2024

UNIVERSITATEA DE VEST DIN TIMIŞOARA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
PROGRAMUL DE STUDII DE LICENŢĂ: INFORMATICĂ

Rezolvarea Sistemelor de Ecuații Utilizând Ant Lion Optimizer și Monkey Algorithm

COORDONATOR:
Lect. Dr. Liviu Octavian Mafteiu-Scai

ABSOLVENT:
Crivoi Mihail

TIMIŞOARA
2024

Cuprins

Introducere

Chiar dacă rezolvarea sistemelor de ecuații (SE) este o problemă foarte veche și intens discutată în matematică, aceasta continuă să genereze un interes deosebit datorită prezenței sale în aproape toate domeniile, dovadă fiind numărul mare de lucrări științifice care continuă să apară.

Lucrarea de licență, intitulată „Învățarea Automată în Rezolvarea Sistemelor de Ecuații", explorează utilizarea tehnicilor de optimizare și învățare automată pentru rezolvarea sistemelor de ecuații liniare. Scopul acestei lucrări este de a investiga algoritmi de optimizare inspirați din natură, precum Ant Lion Optimization Algorithm (ALO) și Monkey Algorithm (MA), și aplicabilitatea acestora în contextul rezolvării eficiente a sistemelor de ecuații liniare.

Din punct de vedere matematic, există două tipuri principale de metode pentru rezolvarea SE: metode directe și metode iterative. Primele constau în transformarea sistemului de ecuații într-un sistem triunghiular echivalent, care este mai ușor de rezolvat. Principalul dezavantaj al metodelor directe constă în erorile de rotunjire cauzate de reprezentarea finită a numerelor în memoria calculatorului. Un astfel de exemplu este cazul eliminării Gaussiene, unde pentru un sistem de dimensiuni medii, de multe ori soluția obținută este departe de una corectă din cauza acumulării erorilor de rotunjire. Mai mult, complexitatea nu este una favorabilă, fiind $O(n^3)$. Cele mai cunoscute metode de acest tip sunt: Cramer, eliminarea Gauss, Gauss-Jordan, factorizarea LU etc.

Metodele iterative determină soluția pe baza unui proces iterativ, pornind de la o aproximație inițială a soluției. O condiție foarte importantă pentru obținerea unei soluții este ca sistemul să fie bine condiționat. Procesul iterativ poate fi oprit după un număr prestabilit de pași sau după atingerea unei precizii suficiente a soluției. Un avantaj este că, în practică, erorile de rotunjire și trunchiere sunt nesemnificative în majoritatea cazurilor, astfel încât soluția iterativă poate avea o acuratețe mai bună decât soluția obținută prin metodele directe [1]. Câteva exemple de metode iterative sunt: Gauss-Seidel, gradientul conjugat, metodele Quasi-Newton, Broyden, Chebyshev etc. Este evident că alegerea corectă a unei metode numerice clasice nu este simplă, iar o alegere nepotrivită poate duce la soluții greșite.

Inteligența Artificială (IA) a condus la o nouă etapă în rezolvarea sistemelor de ecuații, fiind propuse de-a lungul timpului numeroase metode noi, precum Metodele Monte Carlo (MCM), Rețelele Neuronale Artificiale (ANN), Algoritmul Genetic (GA), Optimizarea Roiului de Particule (PSO), Algoritmul de Optimizare a Coloniei de Furnici (ACO), Algoritmul Competitiv Imperialist (ICA), optimizarea prin Căutare Armonică (HS), Algoritmul Memetic (MA), Optimizarea prin Brainstorming (BSO), Algoritmul de Căutare Cuckoo (CS), Procedura de Căutare Adaptativă Aleatorie și Lacomă (GRASP), Optimizarea Roiului de Licurici (GSO), Calculul Cuantic etc. [2], [3].

Hibridizarea acestor metode de IA cu metodele clasice a condus, de asemenea, la rezultate promițătoare. În general, aceste metode hibride combină metaeuristicile cu metode iterative performante, rezultând metode cu o complexitate spațială și temporală bună și soluții precise. În ceea ce privește convergența metodelor care utilizează tehnici din IA, există doar câteva lucrări publicate în care, pentru cazuri particulare ale metodelor propuse, convergența este demonstrată matematic [4], [5].

În acest context, lucrarea de licență analizează în detaliu modul în care algoritmii

ALO și MA pot fi adaptați și utilizați în scopul rezolvării sistemelor de ecuații liniare. De asemenea, se va explora posibilitatea utilizării rețelelor neuronale pentru a îmbunătăți precizia și viteza de rezolvare a acestor sisteme. Performanțele metodelor propuse vor fi comparate cu cele ale metodelor tradiționale, punându-se accent pe acuratețea și eficiența timpului de execuție.

Obiectivele lucrării includ:

- Prezentarea algoritmilor de optimizare ALO și MA.
- Realizarea unei analize comparative a metodelor propuse față de metodele tradiționale.

Capitolul 1

Stadiul actual al cercetării

1.1 Metode numerice tradiționale

Rezolvarea sistemelor de ecuații liniare reprezintă o problemă fundamentală în matematică aplicată, fiind esențială în numeroase domenii precum fizica, ingineria, științele computaționale și economia [1, 2]. De-a lungul timpului, au fost dezvoltate o serie de metode numerice tradiționale pentru soluționarea acestor sisteme, care se împart, în general, în două mari categorii:

1. **Metode directe**: Acestea presupun obținerea soluției printr-un set finit de operații aritmetice, conducând în mod ideal la o soluție exactă (în absența erorilor de rotunjire). Exemple notabile sunt eliminarea Gauss, factorizarea LU și factorizarea QR [3]. Aceste metode sunt eficiente pentru sisteme de dimensiuni mici și bine condiționate. Totuși, complexitatea lor computațională, de ordinul $O(n^3)$, le face impracticabile pentru sisteme foarte mari sau matrice dispersate. În plus, ele pot deveni instabile în prezența erorilor numerice cauzate de rotunjiri.

2. **Metode iterative**: Aceste metode pornesc de la o soluție aproximativă și o îmbunătățesc succesiv printr-un proces de rafinare iterativă. Exemple relevante includ metoda Gauss-Seidel, metoda gradientului conjugat și metoda Chebyshev [2]. Acestea sunt preferate în cazul sistemelor de ecuații liniare de dimensiuni mari și al matricelor sparse (cu multe elemente zero), datorită consumului mai redus de memorie și scalabilității mai bune.

Deși robuste și bine înțelese din punct de vedere teoretic, metodele numerice tradiționale prezintă limitări importante: sunt sensibile la condiționarea matricei sistemului, pot necesita resurse computaționale semnificative și nu sunt întotdeauna aplicabile în cazurile în care datele sunt incerte sau problema este nedefinită strict matematic.

1.2 Tehnici bazate pe inteligență artificială pentru rezolvarea sistemelor de ecuații liniare

O alternativă promițătoare la metodele tradiționale este oferită de tehnicile inspirate din inteligența artificială [4]. Aceste metode abordează problema rezolvării sistemelor de ecuații liniare dintr-o perspectivă diferită, transformând-o într-o problemă de

optimizare globală. Astfel, în loc de a determina direct valorile necunoscutelor, algoritmii caută soluții care minimizează o funcție obiectiv asociată sistemului (de exemplu, eroarea pătratică dintre partea stângă și partea dreaptă a ecuațiilor).

Metodele bazate pe inteligență artificială includ, dar nu se limitează la, algoritmi evolutivi, algoritmi genetici, algoritmi de optimizare bazate pe comportamentul natural (cum ar fi optimizarea viespii, optimizarea roiurilor de particule) sau metaeuristici precum algoritmul Monkey (MA) și algoritmul Ant Lion Optimization (ALO) [5]. Aceste tehnici prezintă avantajul flexibilității și capacității de a explora spații de căutare complexe, fiind capabile să evite capcanele minime locale în căutarea soluției optime globale.

În plus, aceste metode sunt mai puțin sensibile la condiționarea matricei și pot fi aplicate în mod eficient în situații în care sistemele sunt supradeterminate, subdeterminate sau chiar incomplete, unde metodele tradiționale întâmpină dificultăți [6]. Totuși, ele prezintă și dezavantaje, în special în ceea ce privește timpul de execuție, deoarece explorarea spațiului de soluții poate fi costisitoare din punct de vedere computațional.

În concluzie, tehnicile bazate pe inteligența artificială oferă o cale alternativă și complementară pentru rezolvarea sistemelor de ecuații liniare, cu un potențial semnificativ pentru probleme complexe și de mari dimensiuni.

1.2.1 Metoda Monte Carlo

Metoda Monte Carlo utilizează eșantionarea aleatorie pentru a aproxima soluția sistemului de ecuații liniare [7]. Ideea de bază constă în generarea unui număr mare de soluții candidate, evaluate în raport cu o funcție obiectiv care cuantifică eroarea dintre partea stângă și partea dreaptă a sistemului. Pe baza acestor evaluări, sunt selectate soluțiile care minimizează această eroare, oferind astfel o aproximare a soluției sistemului.

- **Avantaje**:
  - Este ușor de paralelizat, ceea ce permite exploatarea eficientă a resurselor computaționale moderne.
  - Se adaptează bine la probleme de dimensiuni mari, inclusiv la cele cu structură complexă sau neliniară.
  - Nu necesită condiționări speciale asupra sistemului sau a matricei, fiind robustă în fața unor probleme dificile.
- **Dezavantaje**:
  - Convergența este, în general, lentă, necesitând un număr foarte mare de iterații pentru a atinge o precizie acceptabilă.
  - Precizia soluției depinde puternic de calitatea și numărul de eșantioane generate.
  - Nu garantează întotdeauna găsirea soluției optime globale, fiind sensibilă la variabilitatea aleatorie.

1.2.2 Algoritmii genetici

Algoritmii genetici sunt tehnici inspirate de procesul evolutiv al selecției naturale, în care o populație de soluții candidate evoluează prin operații genetice precum încrucișarea (crossover), mutația și selecția celor mai bune soluții [8].

- **Avantaje**:
  - Pot explora eficient spații de căutare complexe, cu multiple minime locale.
  - Sunt capabili să identifice mai multe soluții posibile în cazul sistemelor cu soluții multiple sau subdeterminate.
  - Pot trata sisteme slab condiționate sau zgomotoase, unde metodele clasice pot eșua.
- **Dezavantaje**:
  - Timpul de convergență poate fi ridicat, mai ales pentru probleme de dimensiuni mari.
  - Complexitatea algoritmică crește semnificativ odată cu dimensiunea populației și a spațiului de soluții, necesitând resurse computaționale considerabile.
  - Performanța depinde sensibil de parametrizarea corectă a operatorilor genetici și de strategiile de selecție.

1.2.3 Optimizarea bazată pe roiuri de particule (Particle Swarm Optimization)

Algoritmul de optimizare bazat pe roiuri de particule (PSO) este inspirat de comportamentul colectiv observat în natură, cum ar fi roiurile de păsări sau bancurile de pești [9]. Fiecare particulă din roi reprezintă o soluție potențială, iar mișcarea acesteia în spațiul de căutare este determinată atât de propria experiență anterioară, cât și de cele mai bune soluții găsite de celelalte particule. Această interacțiune permite o explorare eficientă a spațiului soluțiilor.

- **Avantaje**:
  - Capacitate bună de explorare globală a spațiului de căutare;
  - Performanță stabilă și robustă în diverse tipuri de probleme de optimizare;
  - Posibilitatea de a fi îmbunătățit prin mecanisme adaptative pentru ajustarea parametrilor, crescând eficiența și viteza de convergență;
  - Implementare simplă și cost computațional moderat.
- **Dezavantaje**:
  - Poate suferi de convergență prematură către minime locale;
  - Performanța depinde puternic de parametrizarea algoritmului (de exemplu, viteza particulelor, coeficienții de învățare);
  - Nu garantează găsirea soluției globale optime.

1.2.4 Algoritmul coloniilor de furnici (Ant Colony Optimization)

Optimizarea bazată pe comportamentul coloniilor de furnici (ACO) se inspiră din modul în care furnicile reale găsesc cele mai scurte trasee între sursa de hrană și cuib, prin depozitarea și urmarea urmelor de feromoni [10]. În contextul rezolvării sistemelor de ecuații liniare, ACO utilizează acest principiu pentru a explora spațiul soluțiilor, combinând colaborarea indirectă între agenți și actualizarea adaptivă a informației feromonale.

- **Avantaje**:
  - Este potrivită pentru sisteme slab condiționate sau pentru cele care prezintă multiple soluții posibile;
  - Prezintă o bună scalabilitate și poate fi paralelizată eficient, crescând viteza de calcul în aplicații mari;
  - Are flexibilitate în adaptarea la diverse tipuri de probleme și poate integra criterii suplimentare în funcția de evaluare;
  - Metoda este robustă în prezența zgomotului și a incertitudinilor în date.
- **Dezavantaje**:
  - Timpul de convergență poate fi mai mare comparativ cu alte metode de optimizare;
  - Necesită ajustarea atentă a parametrilor, cum ar fi rata de evaporare a feromonilor și intensitatea depunerii acestora;
  - Performanța poate fi afectată de structura spațiului de căutare, necesitând uneori strategii suplimentare pentru evitarea stagnării.

1.2.5 Abordări hibride

O direcție de cercetare în plină dezvoltare este combinarea tehnicilor bazate pe inteligență artificială cu metodele numerice clasice, rezultând algoritmi hibrizi [11]. De exemplu, un algoritm genetic poate fi utilizat pentru a genera o soluție inițială apropiată de cea optimă, care este apoi rafinată cu ajutorul metodei Newton sau a gradientului conjugat.

- **Rezultate**: Astfel de abordări hibridizate conduc, adesea, la o îmbunătățire considerabilă a vitezei de convergență, fără a compromite acuratețea soluției finale.

1.3 Provocări actuale și direcții viitoare de cercetare

În ciuda progreselor înregistrate, aplicarea metodelor bazate pe inteligență artificială în rezolvarea sistemelor de ecuații liniare ridică în continuare provocări teoretice și practice [6]. Una dintre cele mai mari rezerve exprimate de comunitatea științifică

este lipsa unui cadru teoretic riguros care să garanteze convergența și eficiența acestor metode.

Cercetările viitoare ar trebui să se concentreze pe următoarele direcții:

- **Stabilirea unui cadru teoretic formal** care să permită analiza riguroasă a performanței algoritmilor evolutivi și metaeuristici în contextul sistemelor de ecuații liniare.
- **Integrarea abordărilor hibride** care să combine avantajele robusteții metodelor numerice cu flexibilitatea algoritmilor bazate pe inteligență artificială.
- **Dezvoltarea tehnicilor de precondiționare** adaptate algoritmilor evolutivi, care să accelereze convergența și să reducă costurile computaționale.
- **Extinderea capacității de paralelizare** a algoritmilor, pentru a putea fi utilizați eficient în sisteme de calcul distribuit sau pe arhitecturi GPU.

Intersecția dintre metodele numerice tradiționale și cele inspirate de natura algoritmică a inteligenței artificiale oferă un cadru promițător pentru rezolvarea unor probleme complexe. Astfel, direcțiile viitoare de cercetare pot contribui nu doar la optimizarea performanței algoritmice, ci și la o înțelegere mai profundă a modului în care tehnicile computaționale pot modela și rezolva probleme fundamentale din știință și inginerie.

Capitolul 2

Algoritmi de Optimizare

2.1 Metoda Ant Lion Optimizer

Algoritmul *Ant Lion Optimizer (ALO)*, propus de *Seyedali Mirjalili* în lucrarea [17], este un algoritm inspirat din natură care imită mecanismul de vânătoare al leilor furnică (antlioni) din mediul natural. Algoritmul a fost inspirat de viața acestor insecte, care aparțin familiei *Myrmeleontidae*. Ciclul de viață al acestor insecte cuprinde două faze: larvară și adultă, prima fiind cea mai importantă în cadrul algoritmului ALO. De fapt, algoritmul ALO imită interacțiunea dintre furnici și antlioni în capcana acestora. Comportamentul leilor furnică și al prăzii lor din natură a fost modelat matematic, ALO fiind bazat pe acest model matematic [17].

Algoritmul ALO este prezentat mai jos sub formă de pseudocod:

Algorithm 1 Pseudocod pentru Ant Lion Optimizer (ALO)

```
Inițializează aleator populația inițială de furnici și antlioni
Calculează fitness-ul furnicilor și al antlionilor
Identifică cei mai buni antlioni și consideră-l pe cel mai bun ca elită (optimum determinat)
while nu este îndeplinit criteriul de oprire do
    for fiecare furnică do
        Selectează un antlion folosind ruleta
        Actualizează valorile minime și maxime pentru toate variabilele
        Creează o plimbare aleatoare și normalizeaz-o
        Actualizează poziția furnicii
    end for
    Calculează fitness-ul tuturor furnicilor
    Înlocuiește un antlion cu furnica corespunzătoare dacă aceasta are un fitness mai bun
    Actualizează elita dacă un antlion devine mai bun decât elita curentă
end while
return elita
```

Algoritmul ALO este capabil să găsească aproximații bune ale optimului global în problemele de optimizare. Câteva dintre motivele principale ale succesului său sunt: selecția aleatoare a antlionilor, plimbările aleatoare ale furnicilor în jurul acestora, restrângerea adaptivă a limitelor capcanelor antlionilor, reducerea adaptivă a mișcării

furnicilor, calcularea plimbărilor aleatoare pentru fiecare furnică și pentru fiecare dimensiune, precum și compararea dintre cel mai bun antlion din fiecare iterație și cel mai bun antlion obținut până atunci (elita).

Este important de menționat că plimbările aleatoare ale furnicilor și selecția antlionilor prin ruletă ajută la evitarea stagnării în minime locale – un aspect esențial pentru metaeuristicile bazate pe populație.

Algoritmul ALO a fost utilizat pentru rezolvarea unei game variate de probleme de optimizare, precum: controlul automat al generării într-un sistem cu mai multe zone [18], proiectarea optimă a structurilor scheletice [19], problema repartizării optime a sarcinii [20], planificarea flexibilă a proceselor [21], planificarea traseelor pentru vehicule aeriene fără pilot [22], integrarea planificării procesului cu programarea acestuia [23], probleme de proiectare structurală [24], selecția caracteristicilor [25], îmbunătățirea performanței rețelelor de distribuție [26] etc.

2.2 Metoda Monkey Algorithm

Monkey Algorithm (MA) este un algoritm metaeuristic de căutare propus de R. Zhao și W. Tang în anul 2007 [27]. Algoritmul simulează comportamentul maimuțelor care se deplasează peste munți în căutare de hrană, presupunând o relație de proporționalitate directă între cantitatea de hrană și înălțimea muntelui. Astfel, muntele cel mai înalt corespunde soluției problemei în cazul maximizării globale.

Există două tipuri de mișcări ale maimuței:

- Maimuța urcă până ajunge în vârful muntelui, escaladarea fiind graduală pentru a îmbunătăți valoarea funcției obiectiv. În același timp, maimuța poate face o serie de sărituri locale (căutare locală) într-o direcție aleatorie, în speranța de a găsi un munte mai înalt;
- după ce maimuța efectuează un număr fix de urcări și sărituri locale, consideră că a explorat suficient zona învecinată poziției sale inițiale și dorește să exploreze o nouă regiune din spațiul de căutare. Pentru aceasta, maimuța efectuează o săritură globală lungă (căutare globală), scopul fiind accelerarea ratei de convergență.

Pașii de mai sus se repetă un număr specificat de ori, acest număr fiind unul dintre parametrii algoritmului. Soluția problemei este vârful muntelui cel mai înalt găsit de populația de maimuțe.

Algoritmul MA este descris mai jos prin pseudocod:

Algorithm 2 Pseudocod pentru Monkey Algorithm (MA)

1: Inițializează populația de maimuțe și distribuie-le aleator în spațiul de căutare
2: Măsoară înălțimea (valoarea obiectiv) în poziția fiecărei maimuțe
3: Efectuează sărituri locale de un număr fix de ori
4: Dacă există un vârf mai înalt, maimuța face sărituri locale din acea poziție
5: Dacă limita săriturilor locale este epuizată fără a găsi un vârf mai bun, se efectuează o săritură globală
6: Dacă criteriile de oprire nu sunt îndeplinite, se repetă procesul începând de la pasul 2

În ceea ce privește săritura globală, maimuța efectuează această mișcare de-a lungul unei drepte care trece prin centrul populației de maimuțe și poziția sa curentă, în oricare dintre cele două direcții.

Algoritmul MA poate rezolva o multitudine de probleme complexe de optimizare, caracterizate prin non-liniaritate, nondiferențiabilitate și dimensiuni mari, cum ar fi: planificarea traseelor roboților și rutarea vehiculelor [28], planificarea extinderii rețelelor de transmisie [29], rezolvarea problemei rucsacului 0-1 [30], integrarea energiei regenerabile [31], analiza clusterelor [32], selecția caracteristicilor [33], împărțirea componentelor ECE [34], programarea sarcinilor în cloud [35], împachetarea cercurilor cu variabile binare [36], rezolvarea problemei amplasării facilităților fără capacitate [37], amplasarea optimă a senzorilor pentru monitorizarea sănătății [38], identificarea proteinelor esențiale [39], rezolvarea problemei acoperirii seturilor [40], detectarea botnet-ului Neris [41], planificarea sistemelor de distribuție a energiei electrice [42], proiectarea medicamentelor [43], programarea flow-shop [44] etc.

Capitolul 3

Metodele Propuse pentru Rezolvarea Sistemelor de Ecuatii

Găsirea unei soluții pentru un sistem de ecuații este, de fapt, o problemă de optimizare multi-obiectiv (MOP), care poate fi definită matematic astfel:

$$\min F(X) = \begin{pmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{pmatrix} \tag{3.1}$$

unde $x \in \Omega$, $\Omega \subset \mathbb{R}^n$ este spațiul deciziilor sau spațiul variabilelor, iar $x = (x_1, x_2, \ldots, x_n) \in \Omega$ este o soluție candidat. Funcția $F(X) : \Omega \to \mathbb{R}^m$ constă din m funcții obiectiv reale (în cazul nostru, m este numărul ecuațiilor), iar $\mathbb{R}^m$ este spațiul obiectiv.

Ideea de bază a algoritmilor propuși pentru găsirea soluțiilor sistemelor de ecuații este generarea de soluții aleatorii și îmbunătățirea lor cu metodele propuse până când condițiile de oprire sunt satisfăcute.

În general, există trei condiții de terminare: se găsește o soluție exactă (valoarea funcției fitness este zero, situație foarte rară), se găsește o soluție aproximativă (se atinge o anumită precizie), sau se atinge un număr presetat de iterații.

Găsirea soluției sistemului de ecuații $f(X)$, unde vectorul $X = \{x_1, x_2, \ldots, x_n\}$ reprezintă necunoscutele, presupune găsirea unei soluții astfel încât fiecare ecuație din sistem să fie egală cu zero:

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0 \\ f_2(x_1, x_2, \ldots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) = 0 \end{cases} \tag{3.2}$$

O soluție este o atribuire a valorilor variabilelor $x_1, x_2, \ldots, x_n$ astfel încât fiecare ecuație să fie satisfăcută. Mulțimea soluțiilor este mulțimea tuturor soluțiilor posibile pentru sistemul de ecuații și pot exista trei situații: unică, infinit multe sau fără soluție.

În abordarea noastră, metodele propuse sunt privite ca o problemă de optimizare (găsirea soluției prin minimizarea unei funcții obiectiv date). Această funcție este numită funcție fitness și trebuie să măsoare cantitativ cât de bună este o soluție dată în rezolvarea problemei. Funcția fitness spune algoritmului cât de bună este o soluție

particulară. Este evident că trebuie să se satisfacă condiția $F(X) \geq 0$ prin definiție. Astfel, pentru ca x^* să fie minimul global al lui $F(X)$, adică o soluție exactă a sistemului de ecuații, trebuie să avem:

$$f_1(x^*) = f_2(x^*) = \ldots = f_n(x^*) = 0$$

Așa cum se poate vedea în secțiunea Rezultate Experimentale, metodele propuse pot găsi soluții chiar și în cazurile când numărul necunoscutelor este mai mic sau mai mare decât numărul ecuațiilor (sisteme supradeterminate și subdeterminate), sau în cazurile în care unele metode matematice clasice eșuează.

3.1 Metoda Propusă Ant Lion Optimizer pentru Rezolvarea Sistemelor de Ecuatii

Metoda propusă se bazează pe algoritmul ALO și este concepută pentru rezolvarea sistemelor liniare de ecuații.

Ca input, algoritmul propus primește:

- A, b: matricea și vectorul, părți ale ecuației matriceale $Ax = b$;
- numărul de furnici (`num_ants`) corespunzător soluțiilor candidate;
- numărul de antlioni (`num_antlions`) corespunzător soluțiilor ghid;
- `dimension`: dimensiunea vectorului soluție x;
- limitele pentru fiecare dimensiune (`initial_bounds`);
- numărul maxim de iterații (`max_iter`).

Ca output, algoritmul oferă:

- `elite_antlion`: cea mai bună soluție găsită;
- numărul de iterații;
- fitness-ul final/cel mai bun obținut.

Algoritmul Ant Lion Optimizer este descris prin pseudocod:

Algorithm 3 Pseudocod Algoritm Propus Ant Lion Optimizer (ALO)

Inițializează aleator pozițiile furnicilor și furniilor în intervalul limitelor
for fiecare furnică și furnie **do**
    Calculează fitness-ul: fitness(x) = MSE(Ax, b)
end for
Alege furnia cu cel mai bun fitness ca `elite_antlion`
while criteriile de oprire nu sunt îndeplinite (număr maxim de iterații sau fitness suficient de bun) **do**
    Calculează factorul de intensitate I (controlează explorarea)
    for fiecare furnică **do**
        Selectează o furnie ghid folosind selecția ruletă (probabilitate $\propto \frac{1}{\text{fitness}}$)
        for fiecare dimensiune j **do**
            Calculează limitele minime și maxime pentru mișcarea aleatorie a furnicii, pe baza pozițiilor `elite_antlion` și furniei selectate
            Generează două drumuri aleatorii unidimensionale: unul ghidat de furnia selectată și unul ghidat de `elite_antlion`
            Normalizează rezultatele drumurilor aleatorii pentru a se încadra în intervalul $[\min, \max]$
            Actualizează poziția furnicii în dimensiunea j
        end for
        Calculează fitness-ul noii poziții a furnicii
    end for
    for fiecare furnică **do**
        if fitness-ul furnicii este mai bun decât cel al furniei corespunzătoare **then**
            Înlocuiește furnia cu furnica respectivă
        end if
    end for
    if există o furnie cu fitness mai bun decât `elite_antlion` **then**
        Actualizează `elite_antlion` și fitness-ul acesteia
    end if
    if `elite_antlion` nu s-a îmbunătățit de mai multe iterații **then**
        Reinițializează pozițiile furniilor slabe cu poziții aleatorii în limite
    end if
end while
return `elite_antlion`

Analiză operațională a metodei propuse

Algoritmul propus adaptează mecanismul ALO pentru a căuta vectorul x care minimizează eroarea pătratică medie între produsul Ax și vectorul rezultat b. Prin analogie cu comportamentul natural al furnicilor și furniilor, fiecare soluție candidate (furnică) este atrasă de una sau mai multe soluții promițătoare (furnii), ghidând astfel procesul de optimizare.

Selectarea furniilor ghid se realizează cu ajutorul unei selecții de tip ruletă, favorizând soluțiile cu fitness mai bun. Prin drumuri aleatorii constrânse între limite calculate din pozițiile celor mai buni antlioni și ale furniilor selectate, se echilibrează explorarea și exploatarea spațiului de căutare.

Actualizarea periodică a furniilor slabe (prin reinițializare) contribuie la evitarea stagnării algoritmului în minime locale.

Avantaje și limitări

Metoda propusă prezintă mai multe avantaje:

- Poate fi aplicată sistemelor mari sau slab condiționate, unde metodele directe sau iterative clasice pot eșua;
- Permite obținerea de soluții robuste chiar și în prezența incertitudinilor sau a perturbărilor;

Totuși, există și unele limitări:

- Timpul de execuție este semnificativ mai mare decât în cazul metodelor tradiționale;
- Performanța depinde puternic de alegerea parametrilor (dimensiunea populației, limitele inițiale, etc.);
- Nu garantează întotdeauna obținerea soluției exacte, ci o aproximație suficient de bună.

3.2 Metoda Propusă Monkey Algorithm pentru Rezolvarea Sistemelor de Ecuatii

Metoda propusă se bazează pe algoritmul MA și este concepută pentru rezolvarea sistemelor liniare de ecuații.

Ca input, algoritmul propus primește:

- A, b: matricea și vectorul din ecuația matriceală $Ax = b$;
- `num_monkeys`: numărul de maimuțe (soluții candidate);
- `dimension`: dimensiunea vectorului soluție x;
- `initial_bounds`: limitele valorilor posibile pentru x;
- `max_iterations`: numărul maxim de iterații.

Ca output, algoritmul oferă:

- `best_monkey`: cea mai bună soluție găsită;
- numărul de iterații;
- fitness-ul final.

Algoritmul Monkey este descris prin pseudocod:

Algorithm 4 Pseudocod Monkey Algorithm (MA)

Inițializează populația de maimuțe cu valori aleatorii în intervalul [`initial_bounds`[0], `initial_bounds`[1]]
Calculează fitness-ul fiecărei maimuțe: `fitness(x) = MSE(Ax, b)`
Găsește maimuța cu cel mai bun fitness și o salvează ca `best_monkey`
while criteriile de oprire nu sunt îndeplinite **do**
    Ajustează parametrii de învățare și explorare în funcție de numărul iterațiilor
    for fiecare maimuță, în afară de `best_monkey` **do**
        Aplică Climbing (SPSA) pentru îmbunătățire locală
        Aplică Watch Jump pentru a explora o regiune apropiată (dacă fitness-ul se îmbunătățește)
        Aplică din nou Climbing
        Aplică Somersault pentru explorarea în jurul centrului de greutate al populației
    end for
    Recalculează fitness-ul pentru toate maimuțele
    if o soluție mai bună este găsită **then**
        Actualizează `best_monkey`
    end if
    if nu există îmbunătățire după multe iterații **then**
        Reînnoiește unele maimuțe cu performanțe slabe ca variații ale `best_monkey`
    end if
end while
return `best_monkey`

Analiză operațională a metodei Monkey

Algoritmul *Monkey* reproduce comportamentul exploratoriu al unei maimuțe care se deplasează într-un spațiu tridimensional în căutarea unui punct optim. În contextul rezolvării sistemelor de ecuații liniare, fiecare maimuță reprezintă o posibilă soluție pentru vectorul x, iar procesul de căutare are loc într-un spațiu definit de limitele impuse și de dimensiunea sistemului.

Algoritmul este alcătuit dintr-o succesiune de faze, fiecare cu un rol specific:

- **Climbing**: o strategie de tip gradient-free, similară metodei SPSA (Simultaneous Perturbation Stochastic Approximation), care permite rafinarea unei soluții într-un mod eficient din punct de vedere computațional;
- **Watch Jump**: o mișcare stocastică de mică amplitudine care favorizează explorarea locală, utilă pentru evitarea stagnării într-un minim local;
- **Somersault**: un mecanism de relocare a maimuței în jurul unei poziții centrale (de regulă, media populației), menit să stimuleze diversitatea și să prevină convergența prematură.

Parametrii algoritmului, precum intensitatea săriturilor sau amplitudinea deplasărilor, sunt actualizați pe parcursul rulării pentru a echilibra eficient fazele de explorare și exploatare. În etapele finale, accentul se deplasează spre rafinarea soluțiilor existente.

Avantaje și limitări

Metoda propusă prezintă mai multe avantaje:

- Poate fi aplicată sistemelor mari sau slab condiționate, unde metodele directe sau iterative clasice pot eșua;
- Permite obținerea de soluții robuste chiar și în prezența incertitudinilor sau a perturbărilor;
- Este ușor de paralelizat, ceea ce o face potrivită pentru execuție pe GPU sau în sisteme distribuite;
- Are o capacitate crescută de explorare globală a spațiului soluțiilor, evitând blocajele în minime locale;
- Nu necesită informații derivate precum gradientul funcției obiectiv.

Totuși, există și unele limitări:

- Timpul de execuție este, în general, mai mare decât în cazul metodelor tradiționale;
- Performanța depinde puternic de alegerea și calibrarea parametrilor (dimensiunea populației, numărul de iterații, limitele inițiale etc.);
- Nu garantează întotdeauna obținerea soluției exacte, ci o aproximație suficient de bună;
- Convergența poate fi lentă dacă algoritmul nu este adaptat specificului problemei;
- Poate necesita multiple rulari pentru stabilitate statistică.

Capitolul 4

Rezultate Experimentale

Pentru a realiza o comparație riguroasă a performanței algoritmilor propuși – Ant Lion Optimizer (ALO) și Monkey Algorithm (MA) – au fost utilizate patru metode clasice de rezolvare a sistemelor de ecuații liniare. Acestea includ metoda Gradientului Conjugat (CG), metoda Eliminării Gaussiene cu pivotare totală (GE), metoda iterativă Gauss-Seidel (GS) și metoda cu aproximații succesive propusă de Broyden (B). Fiecare dintre aceste metode a fost implementată în limbajul Python, urmând aceeași structură de cod și condiții de testare aplicate algoritmilor metaheuristici, pentru a asigura o evaluare corectă și echitabilă.

Metodele clasice au fost alese pentru diversitatea lor conceptuală: CG este o metodă iterativă special concepută pentru sisteme simetrice și definite pozitiv, GE este o metodă directă ce oferă rezultate exacte în condiții ideale, GS este o metodă iterativă eficientă pentru matrici diagonale dominante, iar metoda lui Broyden este o tehnică quasi-Newton adaptabilă și flexibilă, adesea utilizată pentru probleme neliniare, dar aplicabilă și în rezolvarea sistemelor liniare.

În cadrul experimentelor numerice, s-au folosit două tipuri de seturi de date pentru a evalua robustețea și acuratețea fiecărei metode: (1) Matrici generate aleator, având valori proprii controlate, pentru a simula diverse condiții numerice, inclusiv cazuri bine și prost condiționate, și (2) Matrici reale provenite din colecția Harwell–Boeing Sparse Matrix Collection – o bază de date recunoscută internațional ce conține matrici sparse, extrase din aplicații reale din domenii precum ingineria structurală, transportul de căldură, rețele electrice sau probleme de optimizare. Această colecție este disponibilă public prin Matrix Market, la adresa: `https://math.nist.gov/MatrixMarket/collections/hb.html`.

Scopul utilizării acestor două tipuri de matrici este de a testa nu doar performanța algoritmilor în condiții ideale, ci și capacitatea lor de generalizare în fața provocărilor practice. Astfel, analiza comparativă devine mai relevantă pentru aplicații reale, nu doar pentru cazuri teoretice.

4.1 Primul set de experimente

Scopul principal al primului set de experimente a fost acela de a verifica dacă metodele propuse reușesc să identifice soluția corectă a sistemelor de ecuații, soluție care să fie identică sau cel puțin foarte apropiată de cea obținută prin metodele tradiționale consacrate, precum Gradient Conjugat, Eliminare Gauss, Gauss-Seidel și metoda Broyden. Prin această comparație am dorit să evaluăm atât acuratețea, cât și robustețea

algoritmilor noștri în rezolvarea sistemelor liniare.

În mod special, am fost interesați să observăm comportamentul metodelor propuse în situații mai complexe, în care metodele clasice prezintă dificultăți sau chiar eșuează. Aceste situații includ sistemele supradeterminate, unde numărul ecuațiilor este mai mare decât numărul necunoscutelor, și sistemele subdeterminate, în care numărul ecuațiilor este insuficient pentru a determina o soluție unică. Pentru acest motiv, am inclus în analiza noastră sistemele numerotate 9 și 10, conform Tabelului 1, care reprezintă aceste cazuri speciale și care pun la încercare capacitatea algoritmilor de a găsi soluții fezabile.

Un alt obiectiv important al acestui set de experimente a fost acela de a verifica stabilitatea numerică și viteza de convergență a metodelor noastre în raport cu metodele tradiționale. Am urmărit, de asemenea, să determinăm dacă abordările propuse pot oferi o soluție mai rapidă sau mai precisă, în special în cazul sistemelor mari și/sau cu condiționare deficitară.

În Tabelul 4.1 sunt prezentate o parte dintre rezultatele obținute, ilustrând diferențele și similitudinile între soluțiile calculatate de metodele clasice și cele generate de metodele noastre. Datele experimentale confirmă, în mare parte, că metodele propuse sunt capabile să furnizeze soluții corecte și precise, chiar și în cazul sistemelor pentru care metodele tradiționale au dificultăți.

Tabela 4.1: Rezultate experimentale – soluții și acuratețea acestora

Nr.	Sistem ecuații	Accur. MA	Accur. ALO	CG	GE	GS	B
1	bcspwr01 *	✓ 0.01	✓ 0.1	✓	✓	✓	✓
2	can_24 *	✓ 0.001	✓ 0.1	✓	✓	✓	✓
3	jgl011 *	✓ 0.0001	✓ 0.001	✓	X	X	✓
4	jgl009 *	✓ 0.0001	✓ 0.001	✓	X	X	✓
5	will57 *	✓ 0.0001	✓ 0.01	✓	X	✓	✓
6	ibm32 *	✓ 0.1	✓ 0.01	✓	✓	✓	✓
7	$x + 2y = -5$ $3x - y = 13$	✓ 0.00001	✓ 0.01	✓	✓	X	✓
8	$x - y + z = 1$ $x + y + z = 3$ $-x + y + z = -3$	✓ 0.00001	✓ 0.01	X	✓	X	✓
9	$5x + 4y - z = 8$ $x - y + z = 4$ **Sistem subdeterminat**	✓ 0.00001 *Infinitate soluții*	✓ 0.01 *Infinitate soluții*	X	X	X	X
10	$x + y = 2$ $3x - y = 2$ $-x + y = 0$ **Sistem supradeterminat**	✓ 0.00001	✓ 0.01	X	X	X	✓

Legendă:

- **MA** – Algoritmul Monkey propus
- **ALO** – Algoritmul Ant Lion Optimization propus
- **CG** – Gradient Conjugat
- **GE** – Eliminare Gauss
- **GS** – Gauss-Seidel
- **B** – Broyden

✓ Soluție găsită

× Soluție negăsită

∗ Matrici din:
`https://math.nist.gov/MatrixMarket/collections/hb.html`

Concluzii

Ca principale concluzii ale acestui prim set de experimente, metodele propuse, Algoritmul Monkey (MA) și Algoritmul Ant Lion Optimization (ALO), au demonstrat o capacitate constantă de a găsi soluții pentru toate sistemele testate, inclusiv pentru cele pentru care metodele tradiționale au eșuat. În mod particular, MA a evidențiat o acuratețe superioară, cu valori ale erorii relative foarte mici, de ordinul 10^{-5} până la 10^{-4} pentru majoritatea sistemelor, în timp ce ALO a obținut soluții cu acuratețe ușor mai redusă, însă încă foarte satisfăcătoare, de ordinul 10^{-2} până la 10^{-1}.

În comparație, metodele tradiționale precum Eliminarea Gauss (GE) și Gauss-Seidel (GS) au întâmpinat dificultăți în cazul unor sisteme, cum ar fi jgl011, jgl009 și will57, unde nu au reușit să găsească soluții, în timp ce Gradient Conjugat (CG) și metoda Broyden (B) au avut performanțe mai bune, rezolvând majoritatea sistemelor, dar fără a putea trata sistemele supradeterminate sau subdeterminate.

Un punct esențial al experimentelor îl reprezintă capacitatea metodelor MA și ALO de a rezolva cu succes sisteme liniare supradeterminate (exemplul 10) și subdeterminate (exemplul 9), cazuri în care toate metodele tradiționale au eșuat, cu excepția metodei Broyden care a găsit soluția pentru sistemul supradeterminat. Algoritmii propuși nu doar că au găsit soluții, dar au și recunoscut natura acestor soluții, în cazul sistemului subdeterminat indicând existența unei infinități de soluții posibile.

Aceste rezultate confirmă faptul că MA și ALO oferă alternative viabile și robuste pentru rezolvarea sistemelor liniare, în special în situațiile problematice unde metodele clasice sunt limitate.

4.2 Al doilea set de experimente

În acest al doilea set de experimente, obiectivul principal a fost evaluarea fezabilității și eficienței metodelor propuse în comparație cu metodele clasice, prin analiza timpului de execuție necesar pentru rezolvarea sistemelor de ecuații testate. Astfel, am urmărit să determinăm dacă avantajele de acuratețe ale algoritmilor Monkey (MA) și Ant Lion Optimization (ALO) vin însoțite și de performanțe temporale competitive sau acceptabile față de tehnicile convenționale precum Gradient Conjugat (CG), Eliminarea Gauss (GE), Gauss-Seidel (GS) și metoda Broyden (B).

Rezultatele obținute pentru diferite sisteme liniare sunt sintetizate în Tabelul 4.2, unde sunt prezentate timpii de rulare măsurați în condiții similare de execuție, permițând o comparație directă între metode.

Analiza acestor date ne va ajuta să înțelegem mai bine compromisurile între precizie și timp în aplicarea metodelor propuse, precum și potențialele lor aplicații practice în rezolvarea sistemelor mari și complexe.

Tabela 4.2: Rezultate experimentale – timp de execuție (secunde)

Nr.	Sistem ecuații	MA (medie)	ALO (medie)	CG	GE	GS	B
1	`bcspwr01 *`	76	99	<1	<1	<1	<1
2	`can_24 *`	64	60	<1	<1	<1	<1
3	`jgl011 *`	75	33	<1	X	X	<1
4	`jgl009 *`	49	18	<1	X	X	<1
5	`will57 *`	48	454	<1	X	<1	<1
6	`ibm32 *`	70	145	<1	<1	<1	<1

Concluzii

Din perspectiva timpului de execuție, rezultatele prezentate în Tabelul 4.2 indică faptul că metodele metaeuristice propuse, Algoritmul Monkey (MA) și Algoritmul Ant Lion Optimization (ALO), au un cost temporal mult mai ridicat comparativ cu metodele tradiționale precum Gradient Conjugat (CG), Eliminarea Gauss (GE), Gauss-Seidel (GS) și metoda Broyden (B). Astfel, pentru toate sistemele testate, timpii medii de rulare ai metaeuristicilor sunt cuprinsi între câteva zeci și câteva sute de secunde, în timp ce metodele clasice oferă rezultate în intervalul de sub o secundă, cu excepția unor cazuri în care metoda GE sau GS nu au reușit să găsească soluția.

De asemenea, nu se poate evidenția o superioritate clară între cele două metode metaeuristice, MA și ALO, deoarece performanța acestora depinde puternic de funcțiile de randomizare utilizate și de strategiile specifice de explorare și exploatare a spațiului soluțiilor. De exemplu, în cazul sistemului `will57 *`, ALO are un timp de execuție mult mai mare față de MA, iar pentru alte sisteme situația este inversă.

Aceste observații indică faptul că, deși metaeuristicile asigură o acuratețe bună și pot aborda sisteme problematice, costul computațional este un factor limitativ important. Prin urmare, pentru aplicații unde timpul de calcul este critic, metodele tradiționale rămân recomandate, iar utilizarea metaeuristicilor trebuie atent evaluată în funcție de cerințele specifice ale problemei.

Capitolul 5

Concluzii și direcții viitoare

În această lucrare, două metaeuristici inspirate din natură – algoritmul Ant Lion Optimizer (ALO) și algoritmul Monkey Algorithm (MA) – au fost utilizate pentru rezolvarea sistemelor de ecuații liniare. Problema a fost transformată într-o problemă de optimizare, mai exact într-o problemă de minimizare din perspectiva unei funcții obiectiv.

S-a demonstrat experimental că ambii algoritmi pot fi folosiți, cu adaptările necesare, pentru a rezolva sisteme de ecuații liniare, având capacitatea de a găsi întotdeauna o soluție suficient de bună în comparație cu metodele matematice clasice. Mai mult, algoritmii propuși sunt preciși și au reușit să găsească soluția unui sistem de ecuații chiar și în cazurile în care metodele tradiționale eșuează.

Rezultatele experimentale au arătat că ambii algoritmi au performanțe comparabile, însă algoritmul Monkey Algorithm a prezentat o rată de convergență superioară în majoritatea testelor efectuate. Acest lucru sugerează o mai bună eficiență în explorarea spațiului soluțiilor și o capacitate crescută de a evita blocarea în minime locale.

Direcții viitoare de cercetare includ:

- *Paralelizarea algoritmilor ALO și MA*, pentru a reduce timpii de execuție care, în forma lor secvențială, pot deveni prohibitiv de mari în cazul sistemelor de dimensiuni mari;
- *Extinderea capabilităților algoritmilor* pentru a putea identifica *toate soluțiile posibile* ale unui sistem de ecuații într-un anumit interval sau spațiu de căutare, ceea ce ar fi util în analiza sistemelor neliniare sau subdeterminate.

Ca o concluzie generală, se poate afirma că utilizarea metaeuristicilor pentru rezolvarea sistemelor de ecuații liniare reprezintă o alternativă viabilă și promițătoare la metodele clasice, mai ales în contexte unde acestea din urmă întâmpină dificultăți.

Bibliografie

[1] L. N. Trefethen, D. Bau III, *Numerical Linear Algebra*, SIAM, 1997.

[2] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM, 2003.

[3] G. H. Golub, C. F. Van Loan, *Matrix Computations*, 4th ed., Johns Hopkins University Press, 2013.

[4] Z. Zhang, M. Z. Bazant, *Machine learning for solving linear systems and inverse problems*, Proceedings of the National Academy of Sciences, vol. 116, no. 51, pp. 25350-25355, 2019.

[5] S. Mirjalili, *Nature-Inspired Optimization Algorithms*, Academic Press, 2019.

[6] C. Lucas, M. Clerc, *A review of evolutionary algorithms for solving systems of linear equations*, Applied Soft Computing, vol. 14, pp. 575-584, 2014.

[7] R. E. Caflisch, *Monte Carlo and quasi-Monte Carlo methods*, Acta Numerica, vol. 7, pp. 1-49, 1998.

[8] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[9] J. Kennedy, R. Eberhart, *Particle swarm optimization*, Proceedings of ICNN'95 - International Conference on Neural Networks, vol. 4, pp. 1942-1948, IEEE, 1995.

[10] M. Dorigo, T. Stützle, *Ant Colony Optimization*, MIT Press, 2006.

[11] X. S. Yang, *Engineering Optimization: An Introduction with Metaheuristic Applications*, John Wiley & Sons, 2010.

[12] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, SIAM, 1995.

[13] L. Mafteiu-Scai, E. Mafteiu, and R. Mafteiu-Scai, "Solving Equations Systems Using Artificial Intelligence–a Survey," *International Journal of New Computer Architectures and Their Applications*, vol. 8, no. 3, pp. 102–119, 2018.

[14] K. Rajwar, K. Deep, and S. Das, "An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges," *Artificial Intelligence Review*, vol. 56, pp. 13187–13257, 2023.

[15] W. J. Gutjahr, "Convergence Analysis of Metaheuristics," in *Matheuristics*, V. Maniezzo, T. Stützle, and S. Voß, Eds., Annals of Information Systems, vol. 10, Springer, 2009.

[16] V. Vassiliadis, "Stochastic Convergence Analysis of Metaheuristic Optimisation Techniques," *Studies in Fuzziness and Soft Computing*, Springer, 2013.

[17] S. Mirjalili, "The Ant Lion Optimizer," *Advances in Engineering Software*, vol. 83, pp. 80–98, 2015.

[18] M. Raju, L. C. Saikia, and N. Sinha, "Automatic generation control of a multi-area system using ant lion optimizer algorithm based PID plus second order derivative controller," *International Journal of Electrical Power & Energy Systems*, vol. 80, pp. 52–63, 2016.

[19] S. Talatahari, "Optimum design of skeletal structures using ant lion optimizer," *Iran University of Science & Technology*, vol. 6, no. 1, pp. 13–25, 2016.

[20] Nischal, M. Mahendru, and S. Mehta, "Optimal load dispatch using ant lion optimization," *Int J Eng Res Appl*, vol. 5, no. 8, pp. 10–19, 2015.

[21] M. Petrović et al., "The ant lion optimization algorithm for flexible process planning," *JPE*, vol. 18, no. 2, pp. 65–68, 2015.

[22] P. Yao and H. Wang, "Dynamic Adaptive Ant Lion Optimizer applied to route planning for unmanned aerial vehicle," *Soft Computing*, 2016.

[23] M. Petrović et al., "The Ant Lion Optimization Algorithm for Integrated Process Planning and Scheduling," *Applied Mechanics & Materials*, vol. 834, 2016.

[24] B. S. Yıldız, "A comparative investigation of eight recent population-based optimisation algorithms for mechanical and structural design problems," *International Journal of Vehicle Design*, vol. 73, no. 1–3, pp. 208–218, 2017.

[25] H. M. Zawbaa, E. Emary, and C. Grosan, "Feature selection via chaotic antlion optimization," *PLoS One*, vol. 11, no. 3, e0150652, 2016.

[26] I. Soesanti and R. Syahputra, "Multiobjective Ant Lion Optimization for Performance Improvement of Modern Distribution Network," *IEEE Access*, vol. 10, pp. 12753–12773, 2022.

[27] R. Zhao and W. Tang, "Monkey algorithm for global numerical optimization," *Journal of Uncertain Systems*, vol. 2, no. 3, pp. 165–176, 2008.

[28] R. V. Devi, S. S. Sathya, and N. Kumar, "Monkey algorithm for robot path planning and vehicle routing problems," *2017 International Conference on Information Communication and Embedded Systems (ICICES)*, IEEE, 2017.

[29] J. Wang et al., "Discrete monkey algorithm and its application in transmission network expansion planning," *IEEE PES General Meeting*, 2010.

[30] Y. Zhou, X. Chen, and G. Zhou, "An improved monkey algorithm for a 0-1 knapsack problem," *Applied Soft Computing*, vol. 38, pp. 817–830, 2016.

[31] C. M. Ituarte-Villarreal, N. Lopez, and J. F. Espiritu, "Using the monkey algorithm for hybrid power systems optimization," *Procedia Computer Science*, vol. 12, pp. 344–349, 2012.

[32] X. Chen, Y. Zhou, and Q. Luo, "A hybrid monkey search algorithm for clustering analysis," *The Scientific World Journal*, vol. 2014, Article ID 938239.

[33] A. I. Hafez et al., "Hybrid monkey algorithm with krill herd algorithm optimization for feature selection," *2015 11th International Computer Engineering Conference (ICENCO)*, IEEE, 2015.

[34] E. Kuliev and V. Kureichik, "Monkey search algorithm for ECE components partitioning," *Journal of Physics: Conference Series*, vol. 1015, no. 4, 2018.

[35] P. Gupta and P. Tewari, "Monkey search algorithm for task scheduling in cloud IaaS," *2017 Fourth International Conference on Image Information Processing (ICIIP)*, IEEE, 2017.

[36] R. Torres-Escobar et al., "Monkey algorithm for packing circles with binary variables," *Intelligent Computing & Optimization*, vol. 1, Springer, 2019.

[37] S. Atta, P. R. S. Mahapatra, and A. Mukhopadhyay, "Solving uncapacitated facility location problem using monkey algorithm," *Intelligent Engineering Informatics: Proceedings of the 6th International Conference on FICTA*, Springer, 2018.

[38] T. H. Yi et al., "Optimal sensor placement for health monitoring of high-rise structure based on collaborative-climb monkey algorithm," *Structural Engineering and Mechanics*, vol. 54, no. 2, pp. 305–317, 2015.

[39] A. K. Payra and A. Ghosh, "Identifying essential proteins using modified-monkey algorithm," *Computational Biology and Chemistry*, vol. 88, 107324, 2020.

[40] B. Crawford et al., "A binary monkey search algorithm variation for solving the set covering problem," *Natural Computing*, vol. 19, no. 4, pp. 825–841, 2020.

[41] I. A. Saleh et al., "Using monkey optimization algorithm to detect Neris botnet," *Journal of Physics: Conference Series*, 2020.

[42] F. G. Duque et al., "Modified monkey search technique applied for planning of electrical energy distribution systems," *Frontier Applications of Nature Inspired Computation*, pp. 240–265, 2020.

[43] R. V. Devi et al., "Multi-objective monkey algorithm for drug design," *International Journal of Intelligent Systems and Applications*, vol. 11, no. 3, pp. 31, 2019.

[44] M. K. Marichelvam, Ö. Tosun, and M. Geetha, "Hybrid monkey search algorithm for flow shop scheduling problem under makespan and total flow time," *Applied Soft Computing*, vol. 55, pp. 82–92, 2017.