1 Introducere

1.1 Optimizarea multiobiectivă

Optimizarea multiobiectivă are ca scop optimizarea problemelor matematice care au ca obiectiv mai multe funcții ce trebuie optimizate în același timp. Aceste optimizări sunt folosite unde nu există o soluție unică și îmbunătățirea unui parametru duce la degradarea celorlalți.

Frontul Pareto reprezintă un set optim de soluții care evidențiază cele mai bune relații de compromis între toate obiectivele. De obicei, între obiectivele problemelor cu mai multe obiective apar conflicte prin care îmbunătățirea unui obiectiv duce la degradarea altuia, deci toate obiectivele nu pot fi îndeplinite simultan. [7]

O metodă euritică este o procedură ce determină soluții bune sau aproape optime pentru problemele de optimizare. Față de alte medote de aflare a soluțiilor optime pentro aceste probleme, metodele euristice nu garantează găsirea celei mai optime soluții. Aceste metode sunt folosite pentru problemele pentru care nu există metode de calcul pentru aflarea unei soluții exacte. În acest caz, metodele euristice reprezintă singura abordare pentru a găsi o soluție aproximativă. Chiar dacă metoda de găsire a unei soluții optime pentru o problemă există, aceasta poate fi foarte costisitoare din punct de vedere al timpului de calcul astfel, o metoda euristică poate fi folosită în loc care oferă o soluție mai puțin optimă, dar cu un timp de calcul mult mai mic. [2]

O problemă de optimizare multiobiectivă poate fi reprezentată matematic astfel:

$$\min_{x \in X}(f_1(x), f_2(x), \ldots, f_k(x)) \tag{1}$$

unde $k \geq 2$ reprezintă numărul de obiective și setul X este compus din vectorii de decizie și este definit prin funcții de constrângeri.

1.2 Algoritmi de optimizare multiobiectivi

1.2.1 NSGA-II

Algoritmul genetic cu sortare nedominată (NSGA-II) atenuează trei dificultăți întâlnite la algoritmii care folosesc sortare nedominată: complexitatea de calcul O(MN^3), abordarea neelitistă și nevoia de a specifica un parametru comun. Acest algoritm reduce complexitatea de calcul a algoritmilor nedominați la O(MN^2). [3]

1.2.2 NSGA-III

Algoritmul genetic evolutiv cu mai multe obiective (NSGA-III) adapteaza algoritmul original (NSGA-II) rezolvarea problemelor cu mai multe obiective. Acesta folosește o evoluție bazată pe principiul punctelor de referință care scoate în evidență membrii nedominați și care să fie apropiați de punctele de referință oferite. [6]

1.2.3 GDE3

Pentru optimizare globală a problemelor continue, evoluția diferențială (DE) s-a dovedit a fi unul dintre cei mai importanți algoritmi. Mutația este cel mai important operator al algoritmului și îi oefră capacitatea de a explora și de a exploata. Algoritmul de evoluție diferențială generalizată (GDE) se folosește de două compartimente de mutații care au atât strategii de explorare cât și de exploatare în ele pentru a îmbunătății performanța algoritmului DE. [8] O versiune avansată (GDE3) a algoritmului GDE a fost dezvoltată pentru optimizarea globală cu un număr arbitrar de obiective și constrângeri. Pentru probleme cu un singur obiectiv și fără constrângeri, se va folosi algoritmul original DE. În cazul problemelor cu mai multe obiective, acesta îmbunătățește performanța algoritmului GDE printr-o soluție distribuită mai bună. [4]

1.2.4 PAES

Strategia Pareto arhivată de evoluție

1.2.5 PESA2

1.2.6 SPEA2

1.2.7 IBEA

1.2.8 SMPSO

1.2.9 OMOPSO

Algoritmul multiobiectiv de optimizare cu particle de tip roi (OMOPSO) oferă performanțe competitive față de cei mai performanți algoritmi precum NSGA-II, SPEA2 și poate aproxima frontul Pareto unde ceilalți algoritmi similari de tip PSO nu reușesc. Această performanță a fost obținută prin utilizarea dominanței Pareto, a factorului de aglomerare pentru filtrarea liderilor, a operatorilor de mutație și a dominanței epsilon pentru a avea un set fix de soluții finale produse de algoritm. [5]

1.2.10 CMAES

1.2.11 MOEAD

1.3 Framework-ul MOEA

Framework-ul MOEA este o bibliotecă Java gratuită și cu sursa deschisă pentru dezvoltarea si experimentarea algoritmilor de optimizare cu un singur sau mai multe obiective. Acest framework, poate lucra cu algoritmi genetici, evoluție diferențială, optimizare tip roi de particule, programare genetică, evoluție gramatică și altele. Acesta include un număr de algoritmi precum NSGA-II, NSGA-III, ε-MOEA, GDE3, PAES, PESA2, SPEA2, IBEA, SMS-EMOA, SMPSO, OMOPSO, CMA-ES și MOEA/D. De asemenea, acesta oferă uneltele necesare pentru a dezvolta rapid, executa și testa statistic algoritmii de optimizare. [9]

1.4 Obiectivul si conținutul lucrării

În această lucrare am propus un algoritm multiobiectiv de optimizare pornind de la un algoritm genetic dezvoltat pentru un singur obiectiv. Voi prezenta pe scurt framework-ul MOEA cu sursă deschisă folosit pentru dezvoltare, testarea și optimizarea algoritmului, cei mai performanți algoritmi genetici ce au contribuit la dezvoltarea și îmbunătățirea performanței acestui

algoritm, variantele cu un singur obiectiv și variantele multiobiective dezvoltate. De asemenea, voi prezenta și o problemă pentru testarea algoritmului din domeniul ingineriei electrice unde voi compara performanțele acestui algoritm cu cele ale algoritmilor de top.

1.5 Dezvoltarea unui algoritm multiobiectiv

Pornind de la un algoritm cu singur obiectiv de tip ACOR, vom dezvolta un algoritm multiobiectiv și performant ce concurează cu algoritmii evolutivi multiobiectivi de ultimă generație.

2 Algoritmi de implementare propusi

2.1 NSGA-II și NSGA-III

2.1.1 NSGA-II

Principalele probleme ale algoritmului genetic de sortare nedominată (NSGA) sunt: complexitate de calcul ridicată pentru sortarea nedominată, lipsa elitismului care s-a dovedit foarte importantă în performanțele unui algoritm genetic și necesitatea unui parametru comun σ_{share}. [3]

Algoritmul NSGA îmbunătățit (NSGA-II) propune soluții pentru probleme descrise și s-a dovedit a fi mai bun decât alți doi algoritmi similari PAES și SPEA.

Pentru a identifica soluțiile din primul front nedominat dintr-o populație N, fiecare soluție poate fi comparată cu fiecare soluție din populație pentru a afla dacă este dominată. Pentru asta avem nevoie de O(MN) comarații pentru fiecare soluție, unde M este numărul de obiective. [3]

2.1.2 NSGA-III

În cele mai multe cazuri, problemele de optimizare multiobiective sunt definite ca avand cel puțin 4 obiective, iar cele cu mai puțin de 4 obiective sunt tratate ca facând parte din altă clasă de algoritmi deoarece frontul Pareto generat poate fi vizualizat cu ușurință. De asemenea, numărul de obiective maxim des întâlnit este curpins intre 10 si 15. Dificutățile întâmpinate de

algoritmii evolutivi multiobiectivi ce folosesc principiul de dominare sunt: o proporție mare din populație își pierde dominanța, complexitatea de calcul crește semnificativ cu numărul de obiective, recombinarea ineficientă a soluțiilor, colectarea de metrice crește costul complexității de calcul și datorită necesității unei populații mai mari, vizualizarea este mult mai greoaie. [6]

Algoritmul propus (NSGA-III) diferă de algoritmul original (NSGA-II) prin modificări semnificative ale operatorului de selecție și menținerea diversității populației este ajutată prin aplicarea adaptivă a unui număr de puncte de referință. [6]

2.2 Algoritmi de tip ACOR

Algoritmul euristic de optimizare pe baza coloniei de furnici funcționează după o paradigmă numită sistem de furnici este propus ca o nouă abordare pentru optimizarea combinatorică stocastică. Principalele caracteristici ale acestui model sunt: raspunsul pozitiv contribuie la descoperirea rapidă a soluțiilor bune, calculul distribuit evită o convergență prematură și folosirea unui euristic constructiv lacom ajută la găsirea unor soluții acceptabile la începutul procesului de căutare.[1]

2.2.1 SOACOR

2.2.2 MOACOR

3 Implementare

Implementarea algoritmului a fost facută cu ajutorul framework-ului MOEA. Acesta pune la dispoziția utilizatorului o multitudine de unelte pentru a dezvolta, masura și compara algoritmi multiobiectivi cu ajutorul limbajului de programare Java. De asemenea, conține și un număr mare de implementări ale celor mai populari algorimti și probleme multiobiective.

Framework-ul MOEA pune la dispoziție o interfață generică numită Algorithm pentru a defini propietățile comune tuturor algoritmilor și anume: problema pe care algoritmul o optimizează, populația curentă ce conține cele mai bune soluții la momentul invocarii acesteia, o metodă pentru a efectua un

pas logic al algoritmului, o metodă pentru evaluarea unei soluții, numărul de evaluări efectuate și o metodă de a rula algoritmul cu o condiție de finalizare, care poate fi activată pe baza unor criterii sau a numărului de evaluări.

Pentru a ușura dezvoltarea unui nou algoritm, clasa AbstractAlgorithm oferă o implementare comună pentru majoritatea algoritmilor, astfel pentru dezvoltarea unui noi algoritm este suficientă implementarea metodelor ce precizează cum se face o iterație a algoritmului și cea pentru obținerea arhivei epsilon ce provine din extinderea clasei EpsilonBoxEvolutionaryAlgorithm.

Acest framework permite configurarea unui algoritm prin utilizarea claselor Properties și TypeProperties. Acestea permit pasarea de parametrii către un algoritm sau o problemă și în absența precizării acestora se pot folosi valori predefinite.

3.1 Prima versiune

Algoritmul multiobiectiv definește următorii parametrii cu valori predefinite:

- localitateaProcesuluiDeCautare 0.1
- vitezaDeConvergenta 0.85
- numarFurnici 100
- selectieTournamet false

O iterație a algoritmului presupune calcularea ponderilor și a probabilităților de selecție, construcția și evaluarea soluțiilor, calcularea furnicii dominante și reducerea populației la numărul de furnici definit la creare algoritmului prin metoda de sortare nedominată.

Un element din vectorulu de ponderi ce are numărul de elemente egal cu mărimea populației, este definit astfel:

$$\epsilon^{\frac{-1*i^2}{2*constanta^2*marimea\ \ populatiei^2}} * \frac{1}{constanta} \tag{2}$$

unde:

- $constanta = localitatea\ procesului\ de\ cautare * marimea\ populatiei * \sqrt{2\pi}$

Probabilitățile de selcție reprezintă un vector format din ponderile calculate antecedent unde fiecare element este împărțit la suma ponderilor. Construcția soțiilor se face în funcție de parametrul algorimului "selectieTournamet". În funcție de acesta, se va obține un indice ce va fi folosit în calcularea deviațiilor standard și pentru a construi o nouă soluție. Deviațiile standard se obțin cu ajutorul următoarei formule:

$$deviatie\ standard_i = \frac{viteza\ de\ convergenta * (\sum_{i=0}^{n} variabila\ solutiei_i - variabila\ solutiei_i nd}{marimea\ populatiei - 1} \tag{3}$$

unde:

- n = mărimea populației
- indice = obținut prin selecție tournament sau prin extragerea unui număr aleator din vectorul probabilităților cumulate de selcție

Construcția unei noi soluții se face prin alegerea unui individ din populație cu ajutorul indicelui calculat antecedent și ajustarea variabilelor acestuia cu vectorul deviațiilor standard.

O încercare de a îmbunătății acestă versiune a fost obținută prin modificarea formulei de calcul a ponderilor utilizate în deviațiile standard:

$$\epsilon^{\frac{-1*i^2}{2*localitatea\ \ procesului\ \ de\ \ cautare^2*marimea\ \ populatiei^2}} * \frac{1}{constanta} \tag{4}$$

3.2 A doua versiune

În abordarea inițială a implementării algorimului cu ajutorul frameworkului MOEA, s-a folosit clasa de baza "AbstractEvolutionaryAlgorithm". Pentru a îmbunătății performanțele algorimului, această clasa a fost înlocuită cu "OMOPSO" pentru a beneficia de caracteristicile acestui algorim în special funcția de mutație ce aparține algorimilor PSO. Clasa "OMOPSO" și algorimul nostru se folosesc de această metodă deoarece extind la randul lor clasa "AbstractPSOAlgorithm".

O iterație a acestui algoritm presupune construcția de soluții noi, aplicarea operatorului de mutație, evaluarea particulelor, actualizarea celor mai bune soluții locale, adăugarea particulelor la lideri și evaluarea acestora.

Construcția soluțiilor se face cu ajutorul deviațiilor standard calculate pentru fiecare variabilă a fiecărei particule în funcție de parametrul algoritmului "tipulCalcululuiDeviatiilor":

V1

$$deviatiiStandard = \frac{vitezaDeConvergenta * (\sum_{i=0}^{n} lbp_i - particula_j)}{marimeaPopulatiei - 1} \quad (5)$$

unde:

- n: marimeaPopulatiei
- lbp: cea mai bună particulă locală
- j: poziția curentă pentru care se calculează deviațiile

V2

$$deviatiiStandard = vitezaDeConvergenta * (particula_j - lider) \quad (6)$$

- lider: ales aleator din populația de lideri
- j: poziția curentă pentru care se calculează deviațiile

V3

$$deviatiiStandard = \frac{vitezaDeConvergenta * (\sum_{i=0}^{numarulDeLideri} particula_j - lider_i)}{numarulDeLideri} \quad (7)$$

- lider: ales aleator din populația de lideri
- j: poziția curentă pentru care se calculează deviațiile

V4

$$deviatiiStandard = \frac{vitezaDeConvergenta * (\sum_{i=0}^{marimeaPopulatiei} lbp_i - lbp_j)}{marimeaPopulatiei - 1} \tag{8}$$

- lider: ales aleator din populația de lideri
- j: poziția curentă pentru care se calculează deviațiile

V5

$$deviatiiStandard = \frac{vitezaDeConvergenta * (\sum_{i=0}^{marimeaPopulatiei} lbp_j - lbp_i)}{marimeaPopulatiei - 1} \tag{9}$$

- lider: ales aleator din populația de lideri
- j: poziția curentă pentru care se calculează deviațiile

V6

$$deviatiiStandard = \frac{vitezaDeConvergenta * (\sum_{i=0}^{numarulDeLideri} lbp_j - lider_i)}{numarulDeLideri} \tag{10}$$

- lider: ales aleator din populația de lideri
- j: poziția curentă pentru care se calculează deviațiile

La fiecare particulă se va aduna un număr aleator înmulțit cu deviația standard calculată pe aceeași poziție și valoarea unei soluții inițiale care va fi un individ din populația curentă, cea mai bună particulă locală sau un lider aleator în funcție de valoarea parametrului "tipulCalcululuiDeviatiilor".

Operatorul de mutație aplicat noilor soluții, va adăuga la fiecare variabilă diferența dintre valoarea maximă sau minimă în funcție de o variabilă aleatoare și valoarea curentă a acesteia.

Actualizarea celor mai bune soluții locale se face cu ajutorul funcției "updateLocalBest" ce provine din clasa moștenită "AbstractPSOAlgorithm". Acesta compară vectorul de particule cu vectorul de lideri locali și un individ din populație îi va lua locul unui lider doar dacă comparatorul de dominanță pe baza obiectivelor Pareto va alege individul din populație. Ultimul pas al iterației este marcat de adăugarea populației curente la vectorul global de lideri.

4 Funcții benchmark

Congresul IEEE al calculului evolutiv este unul dintre cele mai mari și importante conferințe din cadrul calculului evolutiv.

4.1 Funcții C.E.C.

Pentru a compara performanțele algoritmilor, în cadrul conferințelor C.E.C. au fost prezentate probleme benchmark precum: DTLZ1, DTLZ2, DTLZ3, DTLZ4, Fonseca, Kursawe și Schaffer2.

4.2 LoenyMO

5 Rezultate

5.1 Din funcțiile C.E.C.

5.2 Din LoneyMO

6 Concluzii

References

[1] Marci Dirugi, Vittorio Maniezzo, and Alberto Colorni. "Ant System: Optimization By a Colony of Cooperating Agents". In: *IEEE TRANSACTIONS ON SYSTEMS* 26.1 (1996), pp. 29–41.

[2] H.A.Eiselt and C.-L.Sandblom. "Integer Programming and Network Models". In: *Springer* (2000).

[3] Kalyanmoy Deb et al. "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II". In: *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION* 6.2 (2002), pp. 182–197.

[4] Saku Kukkonen and Jouni Lampinen. "GDE3: The third Evolution Step of Generalized Differential Evolution". In: *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION* 18.4 (2005), pp. 577–601.

[5] Saku Kukkonen and Jouni Lampinen. "Improving PSO-Based Multiobjective Optimization Using Crowding, Mutation and ϵ-Dominance". In: *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION* 18.4 (2005), pp. 577–601.

[6] Kalyanmoy Deb and Himanshu Jain. "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints". In: *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION* 18.4 (2014), pp. 577–601.

[7] Ehsan Goodarzi, Mina Ziaei, and Edward Zia Hosseinipour. "Introduction to Optimization Analysis in Hydrosystem Engineering". In: *Springer* 25 (2014).

[8] Hossein Sharifi Noghabi, Habib Rajabi Mashhadi, and Kambiz Shojaee G. "Generalized Differential Evolution". In: *7th International Conference on Computer and Knowledge Engineering* (2017).

[9] David Hadka. *MOEA Framework*. URL: `http://moeaframework.org/`. (accessed: 01.06.2023).