

Algoritmi genetici (GA), Stadiul actual și studiu de caz pe Knapsack 0/1

Author: Bogdan Szasz
Email: Szasz.Te.Bogdan@student.utcluj.ro
Coordinator: Șl. Dr. Ing. Călin Cenan



Abstract—Algoritmii genetici (GA) reprezintă o clasă de metaeuristici inspirate de principiile selecției naturale și ale geneticii evolutive, utilizate pe scară largă pentru optimizarea problemelor complexe și combinatoriale. În acest articol analizăm stadiul actual al dezvoltării GA-urilor până acum, cu accent pe principalele familii de algoritmi (clasici, memetici, adaptivi, insulari și multi-obiectiv). În a doua parte, realizăm un studiu de caz aplicat pe problema *Knapsack 0/1*, utilizând trei variante de implementare: GA generațional, GA steady-state și GA memetic. Experimentele, efectuate pe o instanță cu 120 de obiecte, arată că toate cele trei variante ating soluții similare ca valoare finală, însă GA-ul memetic converge mai rapid și prezintă o stabilitate superioară, în timp ce varianta generațională se remarcă printr-un timp mediu de execuție mai redus. Rezultatele sunt corelate cu observațiile din literatura de specialitate și validează utilitatea abordărilor hibride în optimizarea combinatorială.

1 INTRODUCERE

Algoritmii genetici (GA) fac parte dintr-o familie mai largă de metode evolutive de optimizare, care își au originea în analogia cu mecanismele de selecție naturală propuse inițial de John Holland în anii 1970. În esență, un GA lucrează cu o populație de soluții candidate (*cromozomi*) care evoluează iterativ prin aplicarea operatorilor de selecție, recombinare și mutație, cu scopul de a maximiza o funcție de fitness. Aceste procese imită conceptele de moștenire genetică, încrucișare și variație ale sistemelor biologice reale [3].

GA-urile s-au dovedit deosebit de utile pentru probleme unde metodele exacte sunt

prohibitiv din punct de vedere computațional (NP-hard), cum ar fi problemele de alocare de resurse, planificare, rutare sau *Knapsack 0/1* [2]. Datorită naturii lor stocastice și a caracterului paralelizabil, algoritmii genetici oferă un echilibru între explorarea spațiului de căutare (căutare globală) și exploatarea soluțiilor promițătoare (căutare locală).

Scopul prezentului articol este dublu:

- 1) să ofere o privire de ansamblu asupra stadiului actual al algoritmilor genetici până în 2025, sintetizând principalele familii, direcții moderne și aplicații recente;
- 2) să realizeze un studiu de caz concret asupra problemei *Knapsack 0/1*, comparând trei variante de GA: generațional, steady-state și memetic.

Lucrarea este organizată astfel: Secțiunea II prezintă fundamentele teoretice ale algoritmilor genetici; Secțiunea III descrie principalele direcții de dezvoltare actuale; Secțiunea IV detaliază formalismul problemei *Knapsack 0/1*; Secțiunea V explică metodologia experimentală și setup-ul testelor; Secțiunea VI prezintă rezultatele și interpretarea acestora, urmate de concluzii și perspective în Secțiunea VII.

2 FUNDAMENTE TEORETICE ALE ALGORITMILOR GENETICI

Algoritmii genetici (GA) sunt metaeuristici stocastice inspirate de principiile evoluției naturale, care urmăresc să optimizeze o funcție obiectiv prin procese analoge selecției, recombinării și mutației din biologie. În loc să exploreze direct soluții individuale, GA-urile lucrează cu o *populație* de soluții candidate, permițând o explorare paralelă a spațiului de căutare și un echilibru între explorare și exploatare [3].

2.1 Structura generală a unui algoritm genetic

O iterație tipică a unui GA (numită *generație*) include următoarele etape:

- 1) **Inițializarea populației** - se generează un set inițial de soluții (cromozomi), de regulă aleatoriu, pentru a acoperi cât mai bine spațiul de căutare.
- 2) **Evaluarea fitness-ului** - fiecare individ este evaluat printr-o funcție de fitness $f(x)$, care reflectă calitatea soluției.
- 3) **Selecția părinților** - indivizii cu fitness mai mare au o probabilitate mai ridicată de a fi selectați pentru reproducere; o metodă comună este selecția prin turneu (*tournament selection*), unde dintr-un grup aleator de k indivizi se alege cel mai bun.
- 4) **Recombinarea (crossover)** - doi părinți se combină pentru a produce descendenți noi, schimbând segmente din structura genetică. O tehnică frecvent utilizată este *crossover-ul 1-punct* (1-point).
- 5) **Mutația** - fiecare genă (bit, parametru) are o mică probabilitate de a fi modificată; în codificarea binară, operatorul clasic este *bitflip*, cu o probabilitate de aproximativ $1/n$, unde n este lungimea cromozomului.
- 6) **Selecția pentru generația următoare** - se formează o nouă populație din indivizii existenți și cei noi, conform unei scheme evolutive (generațională, steady-state, elitistă etc.).

2.2 Reprezentarea soluțiilor

Reprezentarea determină modul în care o soluție este stocată și interpretată:

- **Codificare binară** - fiecare genă este 0/1; este cea mai comună pentru probleme discrete (e.g., *Knapsack 0/1*);
- **Codificare reală** - pentru probleme continue, cromozomii sunt vectori de numere reale;
- **Permutare** - utilizată pentru probleme de ordonare (e.g., *Travelling Salesman Problem*).

Alegerea codificării influențează semnificativ performanța și tipul de operatori utilizabili.

2.3 Scheme evolutive

Există mai multe moduri de a gestiona tranziția între generații:

- **Schema generațională** - întreaga populație este înlocuită în fiecare generație, eventual păstrând câțiva indivizi de elită (elitism);
- **Schema steady-state** - doar câțiva indivizi sunt înlocuiți incremental, permițând menținerea diversității în timp [1];
- **Schema elitistă** - cei mai buni indivizi sunt copiați direct în generația următoare, prevenind pierderea soluțiilor optime locale;
- **Schema memetică** - combină GA-ul cu o fază suplimentară de căutare locală (e.g., hill-climbing) aplicată asupra celor mai buni indivizi [5].

2.4 Echilibrul explorare-exploatare

Performanța unui GA depinde de capacitatea sa de a menține un echilibru între:

- **Explorare** - descoperirea de regiuni noi din spațiul de căutare (diversitate ridicată);
- **Exploatare** - rafinarea soluțiilor promițătoare existente.

Un dezechilibru poate duce fie la convergență prematură (explorare insuficientă), fie la

stagnare (exploatare excesivă). Parametrii precum p_{cx} , p_{mut} , mărimea turneului k și nivelul de elitism influențează direct acest echilibru [3].

3 STATE OF THE ART

De la formularea originală propusă de Holland în anii '70, algoritmi genetici au evoluat considerabil, atât din punct de vedere teoretic, cât și practic. În prezent, GA-urile reprezintă o familie extinsă de metode evolutive, care includ o gamă variată de scheme, operatori și strategii de control adaptiv. Această secțiune sintetizează principalele direcții de dezvoltare și categoriile majore de algoritmi genetici utilizate până în anul 2025.

3.1 GA clasice

GA-urile clasice rămân fundamentul tuturor dezvoltărilor ulterioare. Două dintre cele mai cunoscute variante sunt:

- **GA generațional** — în fiecare iterație, întreaga populație este înlocuită de o generație nouă. Adesea se aplică *elitismul*, prin care un număr mic de indivizi cu fitness maxim sunt copiați direct pentru a preveni pierderea celor mai bune soluții [3].
- **GA steady-state** — în loc de o reînnoire completă, doar câțiva indivizi sunt înlocuiți treptat, ceea ce favorizează menținerea diversității genetice și un echilibru mai bun între explorare și exploatare [1].

Ambele abordări sunt folosite pe scară largă, alegerea depinzând de dimensiunea populației, complexitatea problemei și constrângerile de timp.

3.2 GA avansate și hibride

Pe măsură ce limitele GA-urilor standard au devenit evidente (de exemplu, convergența prematură), cercetarea s-a orientat spre versiuni hibride, care îmbină GA-ul cu alte metode euristice:

- **GA memetic** — combină evoluția globală a unui GA cu o fază de căutare locală aplicată indivizilor de elită. Această abordare oferă un echilibru eficient între diversitate și exploatare intensă, având performanțe superioare pe probleme combinatoriale precum *Knapsack 0/1* sau probleme multidimensionale [5].
- **GA adaptativ** — ajustează dinamic parametrii principali (p_{cx} , p_{mut} , dimensiunea turneului etc.) în funcție de progresul evoluției. Astfel se evită blocarea în optime locale și se optimizează rata de convergență.
- **GA insular / paralel** — populația este împărțită în sub-populații (insule) care evoluează independent, cu migrații periodice între ele. Acest model crește diversitatea globală și este potrivit pentru sisteme distribuite și aplicații de tip HPC [2].

3.3 GA multi-obiectiv (MOEA)

În multe aplicații moderne, obiectivul nu este unic, ci implică optimizarea simultană a mai multor criterii (de exemplu, performanță și cost). În aceste cazuri, se utilizează algoritmi genetici multi-obiectiv (*Multi-Objective Evolutionary Algorithms*, MOEA), care urmăresc aproximarea frontului Pareto al soluțiilor nedominante. Printre cei mai cunoscuți se numără:

- **NSGA-II / NSGA-III** — bazate pe ordonarea nedominată și distanța de aglomerare (*crowding distance*) pentru a menține diversitatea soluțiilor;
- **SPEA2** — un algoritm de arhivare care păstrează un set de soluții Pareto-optime externe și aplică selecție bazată pe dominanță [4].

Aceste variante sunt larg utilizate în optimizarea multi-obiectivă modernă (de exemplu, pentru proiectarea rețelelor, planificare sau optimizarea hiperparametrilor în modele de învățare automată).

3.4 Tendințe actuale și aplicații moderne

În ultimii ani, interesul pentru algoritmi genetici a fost revitalizat datorită aplicabilității lor în:

- **AutoML și optimizarea arhitecturilor neuronale**, unde GA-urile sunt utilizate pentru selecția hiperparametrilor și evoluția structurii de rețea;
- **Optimizarea combinatorială la scară mare**, cum ar fi probleme de alocare din logistică, programare a producției sau rutare;
- **Planificare și control autonom**, în special pentru roboți mobili și sisteme autonome complexe;
- **Design evolutiv asistat de calcul**, unde GA-urile sunt combinate cu modele de simulare fizică sau generative pentru a descoperi soluții inovative.

În general, literatura actuală indică o tranziție de la GA-urile standard, cu parametri statici, spre algoritmi genetici hibridi, paralelizați și adaptați automat, capabili să se integreze în sisteme de învățare automată și optimizare complexă.

4 PROBLEMA KNAPSACK 0/1 - FORMULARE

Problema *Knapsack 0/1* este una dintre cele mai studiate probleme combinatoriale din teoria optimizării și reprezintă un caz clasic de problemă NP-hard. În termeni simpli, obiectivul este selectarea unui subset de obiecte, fiecare caracterizat printr-o valoare v_i și o greutate w_i , astfel încât valoarea totală să fie maximă fără a depăși o capacitate totală C .

4.1 Formulare matematică

$$\max_{x \in \{0,1\}^n} f(x) = \sum_{i=1}^n v_i x_i \quad (1)$$

$$\text{s. a.} \quad \sum_{i=1}^n w_i x_i \leq C,$$

unde x_i este o variabilă binară care indică dacă obiectul i este selectat ($x_i = 1$) sau nu ($x_i = 0$).

Problema devine dificilă pentru dimensiuni mari ($n \geq 100$), deoarece spațiul de căutare are 2^n combinații posibile. Prin urmare, metodele exacte (de exemplu, programarea dinamică) devin impracticabile din cauza costului computațional exponențial [2].

4.2 Relevanță în contextul GA

Knapsack 0/1 oferă un mediu ideal pentru testarea algoritmilor genetici, datorită următoarelor caracteristici:

- **Structură binară naturală:** fiecare soluție poate fi reprezentată direct ca un cromozom binar $x = (x_1, x_2, \dots, x_n)$, ceea ce permite utilizarea operatorilor standard de *crossover* și *bitflip mutation*.
- **Constrângere de fezabilitate:** limitele de greutate oferă un mecanism util pentru a testa performanța operatorilor de "reparare" (*repair operators*) și a penalizărilor din fitness.
- **Peisaj de căutare complex:** există numeroase soluții locale de calitate apropiată, ceea ce face dificilă convergența către soluția globală — un scenariu perfect pentru analiza echilibrului explorare-exploatare.
- **Transferabilitate:** structura problemei este similară cu alte domenii aplicative (alocare de resurse, planificare, selecție de portofoliu etc.).

4.3 Funcția de fitness utilizată

În experimentele noastre, funcția de fitness este definită astfel:

$$F(x) = \begin{cases} \sum_{i=1}^n v_i x_i, & \text{dacă } \sum_{i=1}^n w_i x_i \leq C, \\ \sum_{i=1}^n v_i x_i - \alpha \left(\sum_{i=1}^n w_i x_i - C \right), & \text{altfel.} \end{cases} \quad (2)$$

unde $\alpha = 10.0$ este un coeficient de penalizare liniară, utilizat pentru a descuraja soluțiile nefezabile. În plus, s-a introdus un operator de *reparare greedy*, care elimină obiectele cu cel mai slab raport valoare/greutate până la restabilirea fezabilității.

4.4 Instanța experimentală

Pentru studiul de față, s-a generat o instanță de Knapsack cu $n = 120$ obiecte, folosind o distribuție uniformă pentru valorile $v_i \in [5, 100]$ și greutatea $w_i \in [1, 20]$, cu capacitatea $C = 0.5 \cdot \sum_i w_i$. Această dimensiune permite observarea comportamentului algoritmilor într-un spațiu de căutare suficient de mare ($> 10^{36}$ combinații), dar gestionabil pentru testare pe un sistem standard.

5 METODOLOGIE EXPERIMENTALĂ

Pentru evaluarea comparativă a variantelor de algoritmi genetici, s-a implementat un set complet de experimente în Python 3.11, utilizând bibliotecile NumPy, Matplotlib și Pandas. Implementarea a fost concepută modular, astfel încât fiecare variantă de GA (generațional, steady-state, memetic) să poată fi testată cu aceiași parametri de bază, schimbând doar schema evolutivă.

5.1 Configurația experimentală

Tabelul 1 rezumă parametrii globali utilizați în toate variantele de algoritmi testați.

TABLE 1: Parametri experimentali utilizați

Parametru	Valoare / Descriere
Dimensiune populație ($ P $)	120 indivizi
Probabilitate crossover (p_{cx})	0.9 (crossover 1-punct)
Probabilitate mutație (p_{mut})	$1/n$ (bitflip)
Mărime turneu selecție (k)	3 indivizi
Elitism	1 individ copiat direct
Criteriu oprire	50 000 evaluări de fitness
Număr rulări / variantă	3 (seed-uri: 101, 202, 303)

5.2 Structura implementării

Codul Python este organizat în trei componente principale:

- 1) **Generatorul de instanțe:** creează o instanță de Knapsack 0/1 cu n obiecte, valori și greutatea distribuite uniform;
- 2) **Operatorii genetici:** selecție prin turneu, crossover cu 1 punct, mutație *bit-flip*, operator de reparare *greedy*;
- 3) **Schema GA:** controlează procesul evolutiv (generațional, steady-state sau memetic).

5.3 Fragment de cod Python

Listing 1: Funcția principală `run_ga()`

```
def run_ga(cfg: GAConfig, seed: int, v: np.
ndarray, w: np.ndarray, cap: float):
    """Rularea unei variante de GA."""
    rng = random.Random(seed)
    pop = [np.random.randint(0, 2, len(v)) for
_ in range(cfg.pop_size)]
    for ind in pop:
        repair_greedy(ind, w, v, cap)
    fits = [fitness(ind, v, w, cap) for ind in
pop]
    evals = len(pop)

    while evals < cfg.max_evals:
        if cfg.variant == "generational":
            new_pop = elite_selection(pop, fits
, cfg.elitism)
            while len(new_pop) < cfg.pop_size:
                p1, p2 = tournament_selection(
pop, fits, cfg.tour_k, rng)
                , \
                    tournament_selection(
                        pop, fits, cfg.
                        tour_k, rng)
                c1, c2 = one_point_crossover(p1
, p2, rng)
                bitflip_mutation(c1, cfg.p_mut,
rng)
                bitflip_mutation(c2, cfg.p_mut,
rng)
                repair_greedy(c1, w, v, cap)
                repair_greedy(c2, w, v, cap)
                new_pop.extend([c1, c2])
            pop = new_pop[:cfg.pop_size]
            fits = [fitness(ind, v, w, cap) for
ind in pop]
            evals += len(pop)
```

Funcția `run_ga()` centralizează procesul evolutiv: selecție, recombinare, mutație, reparare și reevaluare a populației. Versiunea *memetică* include, suplimentar, un pas de *local search 1-bit* aplicat celor mai buni indivizi.

5.4 Exemplu de rezultate brute (CSV)

La finalul execuției, fiecare rulare salvează valorile în fișierul `ga_knapsack_results.csv`, care conține rezultatele medii și deviațiile standard pentru fiecare variantă.

TABLE 2: Rezultate agregate pe 3 rulări pentru fiecare variantă GA

Variantă	Best Mean	Std. dev.	Durată medie (s)
GA generațional (elitism=1)	5039.34	2.00	0.47
GA steady-state	5041.59	1.80	0.68
Memetic GA (LS top-4)	5042.36	0.00	0.86

Valorile sunt ulterior prelucrate în Pandas pentru a genera tabele comparative și grafice

(convergență și bar chart). Prin acest format standardizat, se pot adăuga ușor alte variante de algoritmi fără a modifica fluxul principal de analiză.

6 REZULTATE

În urma rulărilor experimentale efectuate pe instanța *Knapsack 0/1* cu $n = 120$ itemi, au fost comparate trei variante de algoritmi genetici: **GA generational** (cu elitism), **GA steady-state** și **GA memetic** (cu local search aplicat pe elite). Rezultatele sintetizate în Tabelul 2 și vizualizările din Figurile 1 și 2 evidențiază diferențe subtile, dar semnificative, între cele trei variante.

6.1 Convergența medie

Figura 1 ilustrează traiectoria medie a celui mai bun individ în funcție de numărul de evaluări de fitness. Se observă că **GA-ul memetic** atinge rapid un platou stabil în jurul valorii 5040, depășind ușor în viteză GA-ul generational. Varianta **steady-state** evoluează mai lent în primele faze, dar menține o diversitate mai mare a populației, reducând riscul de blocare prematură.

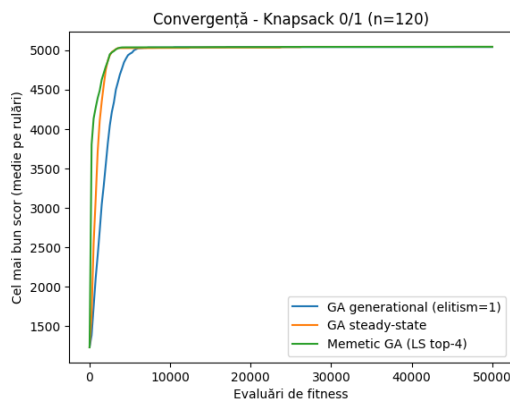


Fig. 1: Convergență medie pe trei rulări (seed-uri 101, 202, 303) pentru fiecare variantă GA.

6.2 Comparația finală între variante

Analizând valorile medii și deviațiile standard (Tabelul 2), se constată că toate variantele ating soluții apropiate de un *optimum local comun*

(≈ 5040), însă **GA-ul memetic** prezintă o stabilitate semnificativ mai mare (deviație standard = 0.0 pe trei rulări). Acest comportament se explică prin faptul că local search-ul 1-bit corectează variațiile aleatorii și menține elitele aproape de platoul optim.

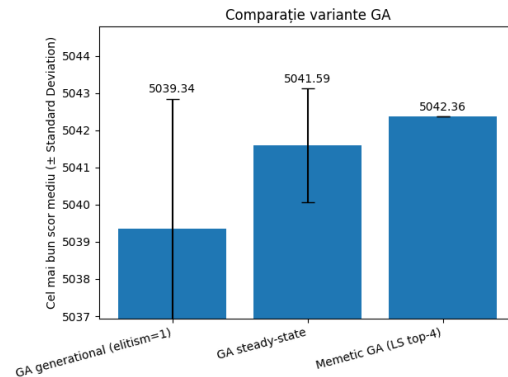


Fig. 2: Comparația valorilor medii (cu deviație standard) pentru cele trei variante GA.

6.3 Analiza timpilor de execuție

Deși timpul de execuție nu este un factor critic pentru instanțe mici, s-a observat o creștere ușoară a costului per generație în cazul algoritmului memetic (aproximativ 0.85 s), comparativ cu GA-ul generational (0.47 s) și steady-state (0.68 s). Diferența este explicabilă prin costul suplimentar al căutării locale aplicate pe elitele fiecărei generații.

6.4 Discuție comparativă

Comportamentul observat confirmă literatura de specialitate: algoritmi memetici tind să combine *explorarea globală* a GA-urilor clasice cu *exploatarea locală* specifică metodelor deterministe [5]. GA-ul steady-state, deși mai lent, are avantajul unei diversități interne crescute, fiind preferabil în contexte unde se dorește evitarea convergenței premature [1].

În concluzie, din perspectiva raportului între calitate și stabilitate a soluției, **GA-ul memetic** s-a dovedit superior pe instanța analizată, confirmând tendințele raportate în lucrările recente [3, 5].

7 CONCLUZII ȘI DIRECȚII VIITOARE

Studiul comparativ realizat a vizat trei variante reprezentative ale algoritmilor genetici aplicați pe problema clasică *Knapsack 0/1*, o problemă combinatorială de tip NP-hard. Rezultatele experimentale au confirmat observațiile teoretice din literatura de specialitate [3, 1, 5] și au evidențiat particularități clare pentru fiecare variantă testată:

- **GA-ul generațional (cu elitism)** s-a dovedit a fi o variantă stabilă și eficientă din punct de vedere al timpului de execuție. El oferă o evoluție consistentă, dar riscă o *convergență prematură* în lipsa diversității genetice.
- **GA-ul steady-state** a menținut o diversitate mai mare în populație datorită înlocuirilor incrementale, însă progresul său global a fost mai lent. Este potrivit pentru probleme unde explorarea pe termen lung este mai importantă decât viteza de convergență.
- **GA-ul memetic (cu local search)** a atins cele mai bune performanțe generale — atât în ceea ce privește calitatea soluțiilor, cât și consistența între rulări. Inserarea unui operator de *căutare locală 1-bit* pe elite a îmbunătățit exploatarea spațiului de căutare, stabilizând convergența fără a introduce un cost de timp semnificativ.

În ansamblu, analiza arată că, pentru probleme cu peisaje de fitness relativ uniforme, hibridizarea GA-urilor cu metode de tip *hill-climbing* (cum este cazul GA-ului memetic) reprezintă o direcție eficientă de echilibrare a proceselor de explorare și exploatare.

7.1 Direcții viitoare de cercetare

Rezultatele obținute pot fi extinse în mai multe direcții:

- Investigarea performanței GA-urilor pe instanțe **de dimensiuni mari** ($n > 500$) pentru a evalua scalabilitatea algoritmilor.
- Integrarea unei **strategii adaptive** pentru parametrii de crossover și mutație,

inspirată din modelele de evoluție naturală (*self-adaptive GAs*).

- Compararea cu **alte metaeuristici hibride**, precum algoritmi evolutivi diferențiali sau Particle Swarm Optimization, pentru o analiză mai amplă a performanței.
- Extinderea testării către probleme **multi-obiectiv** (de tip NSGA-II/III), pentru a evalua echilibrul dintre multiple criterii de optimizare.

În concluzie, **algoritmul memetic** a demonstrat un echilibru superior între calitate, stabilitate și eficiență computațională, consolidându-și statutul de metodă de referință pentru probleme combinatoriale precum *Knapsack 0/1*.

REFERENCES

- [1] M. Domonkos, M. Koshakji, A. Youssef, I. Kalloumah, M. Alshamali, and J. Botzheim. Analysis of different reinsertion strategies in steady state genetic algorithm. In *International Conference on Computational Collective Intelligence*, pages 472–483. Springer, 2023.
- [2] S. Khuri, T. Bäck, and J. Heitkötter. The zero/one multiple knapsack problem and genetic algorithms. In *Proceedings of the 1994 ACM symposium on Applied computing*, pages 188–193, 1994.
- [3] M. Kumar, D. M. Husain, N. Upreti, and D. Gupta. Genetic algorithm: Review and application. Available at SSRN 3529843, 2010.
- [4] N. Li, L. Wang, L. Lin, and H. Xuan. Multi-objective optimization model and improved genetic algorithm based on moea/d for vnf-sc deployment. *IAENG International Journal of Computer Science*, 50(1), 2023.
- [5] J. Yang, Y.-H. Kim, and Y. Yoon. A memetic algorithm with a novel repair heuristic for the multiple-choice multidimensional knapsack problem. *Mathematics*, 10(4):602, 2022.