Analiză comparativă a algoritmilor genetici - Stadiul actual al tehnologiei

Autor: Bogdan Szasz
Email: Szasz.Te.Bogdan@student.utcluj.ro
Coordonator: Șl. Dr. Ing. Călin Cenan

5 noiembrie 2025

Obiective

- **Context:** ce sunt algoritmii genetici (GA) și de ce contează azi (2025).
- **Stadiul artei:** tipuri de GA și direcții recente.
- **Studii sursă:** articole cheie.
- **Studiu de caz:** Knapsack 0/1 - comparăm câteva variante GA.
- Concluzii rapide + direcții viitoare.

Algoritmi genetici - prezentare generală

Idee de bază: căutare euristică inspirată de evoluția naturală [3]. Avem o *populație* de soluții (cromozomi) pe care le selectăm, recombinăm și mutăm, cu obiectivul de a maximiza o funcție de fitness.

- **Reprezentare**: cromozom binar/real/permutare. În cazul nostru, *binar* (0/1).
- **Fitness**: calitatea soluției; În cazul nostru: valoare totală în sac, aplicând o penalizare în cazul în care se depășește capacitatea [3].
- **Selecție**: turneu (*tournament*) - simplu, control al presiunii prin parametrul k.
- **Recombinare**: 1-point crossover - schimb de segmente între părinți.
- **Mutație**: bitflip (inversare de bit cu probabilitate mică $\approx 1/n$).
- **Scheme**: *generațional* (înlocuiești toată populația), *steady-state* (înlocuiri incrementale), *memetic* (GA + căutare locală).

Hint: diferențele de schemă și operatori duc la dinamici de explorare/exploatare diferite [3].

Stadiul artei - taxonomie scurtă

- **GA clasice:**
  - *Generațional*: populație nouă per generație + elitism menține vârfurile[3].
  - *Steady-state*: copii puțini, înlocuiri punctuale → diversitate mai mare în timp[1].
  - *Elitist*: copiezi cei mai buni direct; reduce riscul de „uitare" a soluțiilor bune.
- **GA avansate:**
  - *Memetice*: GA + căutare locală (exploatare agresivă)[5].
  - *Adaptative*: p_{cx}, p_{mut} se ajustează dinamic după stare.
  - *Insulare/paralele*: mai multe sub-populații cu migrații (diversitate globală) [2].
- **Multi-obiectiv (MOEA)**:
  - *NSGA-II/III*, *SPEA2*: fronturi Pareto, crowding, menținere diversitate[4].
- **Nișare/speciație**:
  - *Fitness sharing, crowding*: împiedică dominarea de un singur „tip" de soluție.
- **Domenii curente**:
  - AutoML/tuning hiperparametri, combinatorială mare, planificare, design neural.

Studiu de caz: Knapsack 0/1 — formulare formală

Problemă: avem n itemi, fiecare cu valoare v_i și greutate w_i. Căutăm subsetul care maximizează valoarea totală sub o capacitate C.

$$\max_{x \in \{0,1\}^n} \quad \sum_{i=1}^{n} v_i x_i \quad \text{s. a.} \quad \sum_{i=1}^{n} w_i x_i \leq C$$

Relevanță: problemă **NP-hard** (cel puțin *nondeterministic polynomial time*) clasică; ideală pentru a compara diferite scheme GA (explorare/exploatare) [3].

- **Abordare:** maximizăm valoarea totală, sub constrângerea de greutate.
- **Instanță:** $n = 120$ itemi, capacitate $= 0.5 \cdot \sum w_i$.
- **Reprezentare:** vector binar; fitness cu penalizare + reparare greedy (fezabilizare).

Variante GA testate

1. GA generațional (elitism 1).
2. GA steady-state (înlocuire incrementală).
3. GA memetic (local search pe elite).

Parametri: $|P|$=120, p_{cx}=0.9, p_{mut}=$1/n$, tournament k=3, max. 50k evaluări/run.

GA generațional (cu elitism)

```
# Populatie initiala P
while evals < max_evals:
    E = elite(P, elitism)            # selectam cei mai buni indivizi (elitism)
    Pnew = E                         # noua populatie cu elitele
    while |Pnew| < |P|:              # umplem pana la dim initiala
        p1, p2 = select_tournament(P, k)        # alegem parinti prin tournament
        c1, c2 = crossover_1p(p1, p2) with prob p_cx
        mutate_bitflip(c1, p_mut); mutate_bitflip(c2, p_mut)
        repair_feasible(c1); repair_feasible(c2)
        add c1, c2 to Pnew           # adaugam copiii in noua populatie
    P = Pnew                         # inlocuim vechea populatie cu cea noua

```

Pro: simplu, stabil cu elitism. Contra: risc de convergență prematură fără diversitate [3].

GA steady-state (înlocuire incrementală)

```
# Populatie initiala P
while evals < max_evals:
    p1, p2 = select_tournament(P, k)          # alegem parinti prin turneu
    c1, c2 = crossover_1p(p1, p2) with prob p_cx
    mutate_bitflip(c1, p_mut); mutate_bitflip(c2, p_mut)
    repair_feasible(c1); repair_feasible(c2)
    f1, f2 = fitness(c1), fitness(c2)         # evaluam copiii
    replace_worst_if_better(P, c1, f1)        # inlocuim cel mai slab individ daca
                                                copilul e mai bun
    replace_worst_if_better(P, c2, f2)        # idem pentru al doilea copil

```

Pro: mai multă diversitate în timp. Contra: progres mai lent per unitate de timp [1].

GA memetic (GA + căutare locală 1-bit)

```
# Populatie initiala P
while evals < max_evals:
    E = elite(P, elitism)                  # elitele din gen. curenta
    Pnew = E + offspring_via_GA(P)         # completam restul pop. ca in GA std.
    evaluate(Pnew)
    Top = select_best(Pnew, m)             # cei mai buni m indivizi
    for x in Top:                          # pentru fiecare individ de top
        x_improved = local_search_1bit(x) # aplicam cautare locala (flip 1 bit)
        replace_if_better(Pnew, x, x_improved)
    P = Pnew

```

Pro: exploatare puternică, convergență rapidă. Contra: cost ușor mai mare per generație [5].

Setup experimental

- Implementare Python3 (NumPy, Matplotlib, Pandas).
- n=120 itemi, instanță fixă generată cu seed determinist.
- 3 rulări / variantă (seeds 101, 202, 303).
- **Fitness:** valoare totală cu penalizare + reparare greedy.
- **Criteriu oprire:** 50 000 evaluări de fitness.
- **Output:** convergență + scor mediu $\pm$ deviație standard.

Rezultate agregate (CSV)

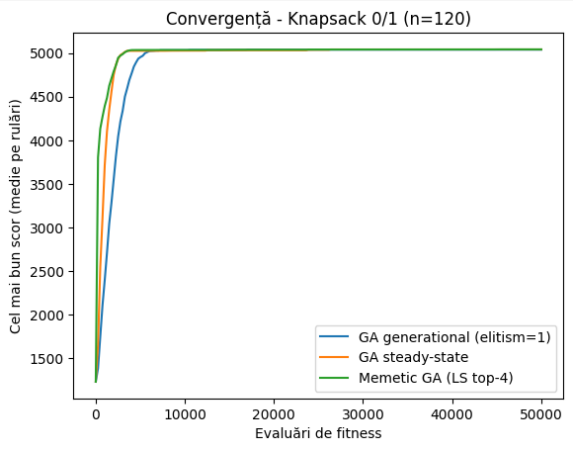
Variantă	Best Mean	Std	Dur. medie (s)
GA generațional	5039.34	2.00	0.47
GA steady-state	5041.59	1.80	0.68
Memetic GA (LS top-4)	5042.36	0.00	0.86

Interpretare:

- *Nivel absolut:* toate variantele ating ≈ 5040, deci similar ca soluție finală pe această instanță.
- *Stabilitate:* deviația standard cea mai mică la **GA-ul memetic** (0.00 pe 3 rulări) $\rightsquigarrow$ comportament foarte consistent.
- *Viteză:* **GA-ul generațional** este cel mai rapid pe rularile noastre; **GA-ul memetic** e ușor mai lent datorită căutării locale.
- *Concluzie:* dacă ai buget fix de evaluări, **GA-ul memetic** tinde să ajungă mai repede pe platou și să fie mai stabil; **GA-ul steady-state** e între cele două.

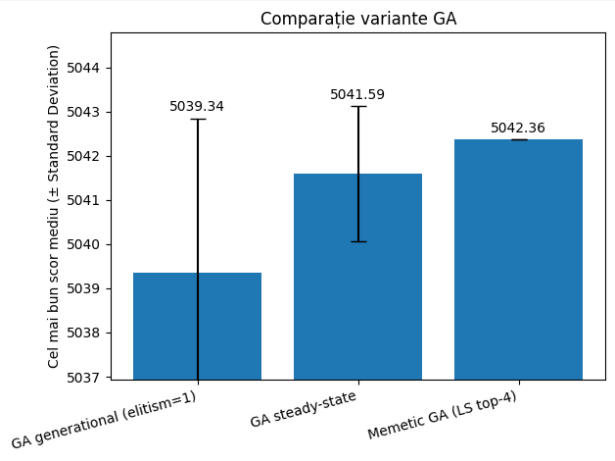
Valorile provin din `data/ga_knapsack_results.csv`. Diferențele mici (0.1%) sunt evidențiate pe grafic prin *zoom* pe axa Y.

Rezultate: convergență medie



- Medie pe 3 rulări/variantă.
- Memetic GA converge ușor mai rapid.
- Toate ating scoruri similare după 10000 evaluări (≈ 5040).

Rezultate: comparație finală



- **GA generațional:** 5039.34 ± 2.0
- **Steady-state:** 5041.59 ± 1.8
- **Memetic GA:** 5042.36 ± 0.0
- Timpuri relative: memetic ≈ 0.85 s, steady ≈ 0.68 s.

Concluzii & lecții

- Toate variantele ating soluții aproape identice ≈ 5040 (optimum local comun).
- GA memetic converge mai rapid și mai stabil (deviație 0.0).
- Steady-state menține diversitatea mai mult, dar e mai lent.
- Timpurile de execuție cresc ușor cu complexitatea (0.47 s $\rightarrow$ 0.86 s).
- Direcții viitoare: codificare reală, GA insulare, adaptivitate hibridă.

Vă mulțumesc pentru atenție!

Bibliografie I

[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.