Iată un raport complet și detaliat, rescris de la zero, care include toate aspectele codului și răspunde întrebărilor suplimentare despre codificarea Gray și comparația Ruletă/Turnir.

**Raport Tehnic Complet: Analiza Improved Binary GA Runner.py**

**1. Sumar Executiv (Obiectivul General) 🎯**

Acest script Python nu implementează un algoritm genetic (GA) binar simplu. El reprezintă un **cadru de cercetare (Grid Search) complex**, conceput pentru a evalua riguros performanța unei arhitecturi de **Algoritm Genetic Hibrid și Adaptiv**.

Scopul său este să găsească valoarea minimă a unor funcții matematice notorii de dificile (funcții benchmark), folosind o combinație de patru tehnici avansate:

1. **Algoritm Hibrid (Memetic):** Combină evoluția globală (GA) cu rafinarea locală (Hill Climbing).
2. **Codificare Gray:** O reprezentare binară specială care elimină "faliile" din spațiul de căutare.
3. **Mutație Non-Uniformă:** O rată de mutație care se adaptează, scăzând pe măsură ce algoritmul converge.
4. **Managementul Stagnării:** O tehnică de "resetare" a populației pentru a reintroduce diversitatea.

Cadrul de testare (Grid Search) rulează sistematic mii de experimente pentru a determina cea mai bună combinație de parametri (ex: rată de crossover, metodă de selecție) pentru acest algoritm, colectând date statistice și generând grafice pentru analiză.

**2. Arhitectura Algoritmului: Un GA Hibrid și Adaptiv 🧠**

Puterea acestui script stă în *sinergia* celor patru tehnici principale, care rezolvă problemele clasice ale algoritmilor genetici simpli (cum ar fi convergența prematură).

**2.1. Hibridizare (Algoritm Memetic): GA + Hill Climbing**

Acesta nu este un GA pur, ci un **Algoritm Memetic**. El combină:

* **Explorarea Globală (GA):** Folosește operatorii de selecție, crossover și mutație pentru a explora regiuni vaste ale spațiului de soluții.
* **Exploatarea Locală (Hill Climbing):** Folosește funcția hill\_climbing\_gray pentru a lua o soluție promițătoare și a o "rafina", căutând intens în vecinătatea sa imediată o soluție și mai bună.

**Cum funcționează (Evoluție Lamarckiană):**

1. După ce se creează o nouă generație de "copii", algoritmul ia cel mai bun individ (elite) din generația *părinților*.
2. Acest individ este "antrenat" intens prin hill\_climbing\_gray, rezultând un refined\_elite.
3. Acest individ "îmbunătățit" este apoi inserat direct în noua generație, înlocuind cel mai slab copil.

Această strategie, în care îmbunătățirile dobândite în timpul "vieții" (Hill Climbing) sunt moștenite direct, accelerează masiv convergența.

**2.2. Codificarea Gray: Rezolvarea "Faliei Hamming"**

Aceasta este poate cea mai importantă îmbunătățire tehnică pentru un GA binar care folosește căutarea locală.

Problema (Binar Standard):

În codificarea binară standard, două numere apropiate pot fi foarte diferite.

* 7 = 0111
* 8 = 1000

Trecerea de la 7 la 8 necesită schimbarea a 4 biți. Pentru un algoritm de mutație sau Hill Climbing, aceste două numere par a fi în colțuri opuse ale universului, deși sunt vecine. Această discrepanță se numește "Falia Hamming" (Hamming Cliff).

**Soluția (Codificarea Gray):**

* 7 = 0100
* 8 = 1100

În Codul Gray, trecerea de la 7 la 8 necesită schimbarea unui singur bit. Această proprietate se menține pentru orice pereche de numere consecutive.

De ce se numește "Gray"? 💡

Numele provine de la inventatorul său, Frank Gray, un cercetător la Bell Labs. În anii 1940, el a inventat acest "cod binar reflectat" pentru a rezolva o problemă la comutatoarele electromecanice. Dacă un senzor rotativ trecea de la 7 la 8, în binar standard (0111 -> 1000), exista riscul ca senzorii să se schimbe la momente ușor diferite, raportând valori complet eronate (ex: 1111 = 15) în timpul tranziției. Codul Gray, schimbând un singur bit, elimină acest risc.

**Impactul asupra algoritmului:** Datorită Codificării Gray, operatorul hill\_climbing\_gray (care schimbă un singur bit) explorează *cu adevărat* vecinii apropiați ai soluției, făcând rafinarea locală extrem de eficientă.

**2.3. Adaptare: Mutația Non-Uniformă**

**Problema:** O rată de mutație fixă este un compromis slab.

* **Rată Mare:** Bună la început pentru explorare, dar "distruge" soluțiile bune spre final.
* **Rată Mică:** Bună la final pentru rafinare, dar blochează algoritmul la început (explorare slabă).

**Soluția:** Rata de mutație (effective\_r\_mut) este dinamică. Ea pornește de la o valoare mare (r\_mut\_initial) și scade exponențial pe măsură ce generațiile avansează (progress = gen / n\_iter), apropiindu-se de o valoare finală mică.

* **La început (Generația 0):** Explorare agresivă.
* **La sfârșit (Generația n\_iter):** Rafinare fină (exploatare).

**2.4. Diversitate: Gestionarea Stagnării (Re-inițializare)**

**Problema:** Chiar și cu toate aceste tehnici, algoritmul se poate bloca într-un optim local (convergență prematură).

Soluția: (Aplicată doar pentru dimensiuni mari, dim >= 30).

Dacă cea mai bună soluție nu se îmbunătățește timp de stagnation\_limit generații:

1. Se păstrează o mică "elită" (cei mai buni 10% indivizi).
2. Restul de 90% din populație este șters și înlocuit cu indivizi noi, generați aleatoriu.

Această "explozie" de diversitate forțează algoritmul să caute în regiuni complet noi ale spațiului.

**3. Cadrul de Cercetare (Grid Search) 🔬**

Scriptul este un laborator automatizat. Acesta nu rulează algoritmul o singură dată, ci efectuează o cercetare sistematică (Grid Search) pentru a înțelege cum funcționează.

**3.1. Funcțiile Benchmark (Problemele)**

Sunt folosite 4 funcții clasice, fiecare testând o altă slăbiciune a algoritmilor:

* rastrigin: Extrem de multimodală ("cutie de ouă"). Testează abilitatea de a evita miile de optimi locali.
* griewangk: Similară, dar cu o componentă globală care poate induce în eroare.
* rosenbrock: O "vale" parabolică lungă și îngustă. Testează eficiența rafinării locale (exploatarea).
* michalewicz: Are minime foarte abrupte și înguste. Testează abilitatea de a "găsi" aceste regiuni mici.

**3.2. Punctul Central al Cercetării: Ruletă 🆚 Turnir**

Scriptul testează două metode de selecție (SELECTION\_METHODS) pentru a răspunde la o întrebare cheie: **"Ce nivel de *presiune de selecție* este optim pentru acest algoritm hibrid?"**

* **Selecția Ruletă (selection\_roulette\_wheel):**
  + **Cum funcționează:** Proporțional cu fitness-ul. Un individ de 2 ori mai bun are șanse de 2 ori mai mari de a fi selectat.
  + **Caracteristică:** Presiune de selecție *mare* și *variabilă*.
  + **Risc:** Dacă un "super-individ" apare devreme, el va domina rapid populația (primind 90% din "roată"), ducând la o pierdere catastrofală a diversității și la convergență prematură. Este o strategie "lacomă".
* **Selecția Turnir (selection\_tournament, k=3):**
  + **Cum funcționează:** Alege 3 indivizi la întâmplare. Cel mai bun din cei 3 câștigă.
  + **Caracteristică:** Presiune de selecție *medie* și *constantă*.
  + **Avantaj:** Este mult mai robustă. Un "super-individ" nu poate domina populația; el poate câștiga doar turnirele în care intră. Un individ mediocru are încă o șansă de a fi selectat dacă nimerește într-un turnir cu doi indivizi și mai slabi.
  + **Rezultat:** Menține diversitatea genetică mult mai mult timp.

**Ipoteza de cercetare:** Grid Search-ul va determina dacă strategia "lacomă" a Ruletei (care găsește rapid un optim local) combinată cu Hill Climbing este mai rapidă, sau dacă strategia "robustă" a Turnirului (care explorează mai mult) oferă o șansă mai bună de a găsi optimul *global*.

**4. Analiza Componentelor Cheie ale Codului (Code Walkthrough) 💻**

**4.1. Decodificarea (Fluxul de date)**

Inima algoritmului este decode\_population\_gray\_vectorized. Iată fluxul de date pentru un individ:

1. **Cromozom Binar:** O listă lungă de biți, ex: [1,0,1,...0,1,1] (lungime n\_dim \* n\_bits).
2. **Remodelare:** Vectorul este "tăiat" în bucăți: (n\_dim, n\_bits). Ex: 10 dimensiuni, 17 biți/dimensiune.
3. **Conversie în Întreg Gray:** Fiecare șir de 17 biți este convertit într-un număr întreg (folosind pop\_reshaped @ powers\_of\_2). Acest întreg este interpretat ca fiind în cod Gray.
4. **Gray -> Binar Standard:** Funcția gray\_to\_binary\_vectorized convertește matematic întregul Gray în întregul binar standard corespunzător. (Acesta este pasul "magic" de decodificare).
5. **Scalare:** Întregul (ex: 65000) este scalat din intervalul [0, 2^17-1] în intervalul real al funcției (ex: [-5.12, 5.12]).
6. **Rezultat:** Un vector de n\_dim valori reale (ex: [1.23, -4.01, ..., 0.87]), gata de a fi evaluat de rastrigin.

**4.2. Ciclul Evolutiv (Logica bga\_improved)**

Aceasta este bucla principală (for gen in range(n\_iter)):

1. **Calcul Mutație:** Calculează effective\_r\_mut pentru generația curentă.
2. **Evaluare:** Decodează (Gray) și evaluează întreaga populație.
3. **Elitism:** Găsește și salvează cel mai bun individ *global* (best\_chromosome).
4. **Anti-Stagnare:** Verifică stagnation\_counter și, dacă e cazul, re-inițializează 90% din populație.
5. **Selecție:** Creează o nouă "populație de părinți" (selected\_pop) folosind Turnir sau Ruletă.
6. **Reproducere:** Generează "copiii" (children) prin crossover și mutație (folosind rata de mutație dinamică).
7. Pasul Hibrid (Memetic):

a. Ia cel mai bun individ din generația actuală (elite = pop[best\_current\_idx]).

b. Îl rafinează intens: refined\_elite = hill\_climbing\_gray(...).

c. Găsește cel mai slab copil (worst\_child\_idx).

d. Îl înlocuiește: children[worst\_child\_idx] = refined\_elite.

1. **Succesiune:** Populația de copii devine noua populație de părinți (pop = children) și ciclul se reia.

**5. Metodologia de Evaluare și Rezultate 📊**

**5.1. Robustețe Statistică (N\_RUNS = 30)**

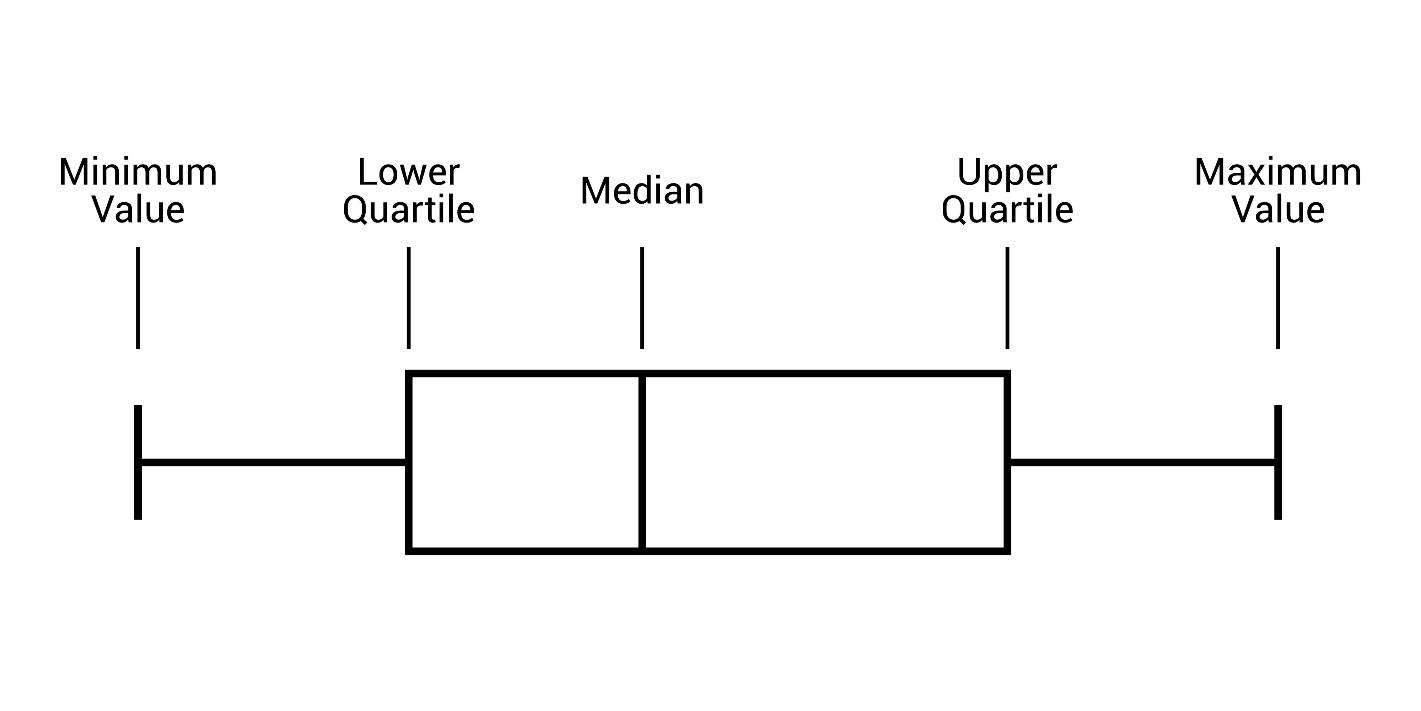
Algoritmii genetici sunt **stocastici** (aleatorii). O singură rulare poate fi norocoasă sau ghinionistă. Pentru a obține o concluzie științifică, scriptul rulează fiecare experiment (fiecare combinație de parametri) de **30 de ori**. Acest lucru permite calcularea unor statistici valide:

* **Cel mai bun scor:** Performanța ideală.
* **Scorul mediu:** Performanța așteptată.
* **Deviația standard:** Robustețea (cât de predictibil este algoritmul).

**5.2. Artefactele de Ieșire (Ce înseamnă graficele?)**

Pentru fiecare experiment, scriptul generează 3 fișiere de analiză:

1. summary.txt: Fișierul text cu datele statistice brute (medie, min, std.dev) și parametrii folosiți.
2. convergence.png: Un grafic linie care arată evoluția celui mai bun scor de-a lungul generațiilor (pentru *cea mai bună* rulare din cele 30). Răspunde la întrebarea: "Cât de repede converge?"
3. boxplot.png:



Shutterstock

Un grafic esențial care arată distribuția tuturor celor 30 de scoruri finale.

\* O cutie mică și joasă: Ideal. Înseamnă că algoritmul este robust (găsește constant aceeași soluție bună).

\* O cutie înaltă: Problematic. Înseamnă că algoritmul este instabil (uneori găsește soluția, alteori eșuează complet).