Minimum sum coloring problem

Tudosă Eduard-Bogdan

April 6, 2023

1 Introducere

Minimum Sum Coloring Problem (MSCP) este o problemă de optimizare combinatorică care are drept scop atribuirea de culori unor elemente ale unui graf, astfel încât să fie minimizată suma culorilor atribuite. Fiecare nod din graf trebuie să fie colorat cu o culoare distinctă, iar două noduri adiacente nu trebuie să aibă aceeași culoare.

Această problemă are o multitudine de aplicații practice, cum ar fi planificarea de trasee pentru transportul public sau asignarea de frecvențe în telecomunicații. De asemenea, este o problemă NP-dificilă, ceea ce înseamnă că nu există un algoritm eficient care să găsească întotdeauna soluția optimă într-un timp rezonabil.

Deși există mai multe abordări pentru a rezolva MSCP, cum ar fi algoritmi genetici sau metode de programare liniară, această problemă continuă să fie o provocare importantă pentru cercetătorii din domeniul optimizării combinatorice. În acest articol, vom explora MSCP și vom analiza diferitele tehnici utilizate pentru a găsi soluții optime sau aproape-optime pentru această problemă.

Arbordarea aleasă va fi aceea de a implementa un algoritm genetic. Deși rezultatele nu vor fi cele mai bune, acest lucru se datorează faptului că este implementată varianta clasică a acestui algoritm, fără hibridizări sau îmbunătățiri specifice pentru un algoritm de colorare. Acest lucru lasă loc unor optimizări ulterioare ale algoritmului în cauză.

2 Algoritm

Un algoritm genetic clasic este o metodă de optimizare combinatorică inspirată din procesele de evoluție biologică. Acest algoritm începe prin generarea unei populații de soluții inițiale, numite cromozomi, care reprezintă posibile soluții la problema de optimizare.

În cazul MSCP, un cromozom poate fi reprezentat ca un șir de numere întregi, fiecare număr reprezentând culoarea atribuită unui nod în graf. În acest context, cromozomul este generat ca un șir de numere întregi aleatoare în intervalul [1, n], unde n este numărul de noduri din graf.

Algoritmul genetic continuă prin aplicarea unei selecții de cromozomi pentru a fi încrucișați, în vederea generării de noi soluții. În cazul algoritmului genetic clasic descris, se folosește o selecție turneu de rang 2, care selectează 2 cromozomi din populație pentru a fi încrucișați. Apoi, se folosește un operator de încrucișare de tipul "un punct de tăiere", care taie cei 2 cromozomi la același punct și combină părțile lor pentru a genera 2 noi cromozomi.

Următorul pas constă în aplicarea unui operator de mutație clasic, care poate să schimbe aleatoriu o parte a cromozomului pentru a genera o soluție nouă. În cazul MSCP, mutația poate fi aplicată la nivelul unui singur nod, schimbându-i culoarea.

Acești pași sunt apoi repetați de mai multe ori, până când se atinge un criteriu de oprire, cum ar fi numărul maxim de generații sau faptul că nu s-au obținut îmbunătățiri în soluție în ultimele câteva generații.

În final, se selectează cel mai bun cromozom din ultima generație, care reprezintă soluția optimă sau cea mai bună soluție găsită până în acel moment.

Selcția cromozomilor se face după o funcție fitness. Această funcție fitness am ales-o ca fiind suma culorilor atribuite plus o penalitate pentru fiecare pereche de noduri ce sunt vecine si au aceeași culoare. Vom dori ca această funcție să fie minimizată. Însă această abordare nu pare a fi cea mai decentă, de aceea va trebui revenit ulterior cu o altă funcție fitness mult mai complexă.

Pentru a obține totuși rezultate mai bune, înainte de a se efectua algoritmul genetic, am realizat o colorare greedy pentru a cunoaste un posibil număr cromatic de la care să plecăm pentru a alcătui cromozimi noștri, deoarece daca am fi plecat de la numărul de noduri, algoritmul ar fi trebuit să fie lăsat foarte mult să evolueze pentru a obține rezultate cât de cît normale.

3 Îmbunătățiri

O primă încercare de îmbunătățire a fost să implementez anumite părți din algoritmul prezentat în articolul `https://www.sciencedirect.com/science/article/abs/pii/S0020025516301335`. Însă nu am reusit în timp util să ajung la o soluție favorabilă, de aceea am venit cu îmbunătățiri originale asupra operatorilor de mutație si încrucișare, care au adus niște îmbunătățiri.

Pentru mutație, atunci când se alege o genă a fi mutată, în loc să schimb culoarea acelei gene, voi alege să schimb culorile tuturor vecinilor nodului respectiv care au aceeași culoare cu el (colorarea fiind una nevalidă) cu o culoare random permisă, diferită de cea anterioară. Pentru a reduce spațiul de căutare, voi alege acel numar pentru mutație ca fiind un număr random care să nu depășească numărul cromatic obținut de colorarea greedy folosită pentru a obține cromozomul inițial.

Pentru încrucișare, am ales ca aceasta să se efectueze cu o probabilitate de 0.8 cu un singur punct de tăiere și cu 0.2 cu mai multe puncte de tăiere alese aleator.

4 Experiment

Algoritmul genetic a fost realizat folosind ca limbaj de programare Java, versiunea 19.0.2. Am ales pentru acest algoritm o populație de 100 de indivizi și un număr de 2000 de generații care trebuie efectuate. De asemenea, la nivel de operatori, am folosit o mutație de 0.4 și o încrucișare de 0.6.

Pentru a se obține datele statistice, algoritmul a fost rulat pentru 30 de epoci.

Pentru experiment am ales mai multe instanțe de test, rezultatele mele fiind comparate cu cele mai bune rezultate cunoscute în literatură pentru problema în cauză. În urma rezultatelor obținute din cele 30 de rulări vom obține 2 seturi de date pentru cele 2 implementări (varianta optimizată și neoptimizată a algoritmilor) și vom efectua următoarele teste statistice: T-test, interval de încredere, cât și anumite grafice de vizualizare a datelor.

4.1 Set de test - myciel3

Variantă neoptimizată:

n	m	min	medie	max	best	sd	greedy	time
11	20	16	19.06	22	16	1.72	22	7s

Variantă optimizată:

n	m	min	medie	max	best	sd	greedy	time
11	20	16	16	16	16	0	22	9s

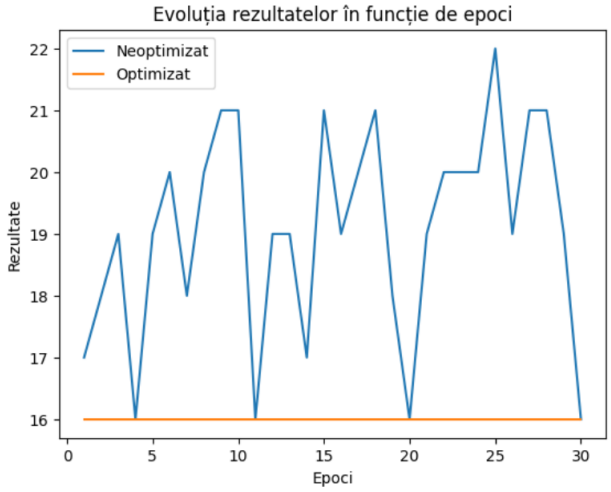

Figure 1

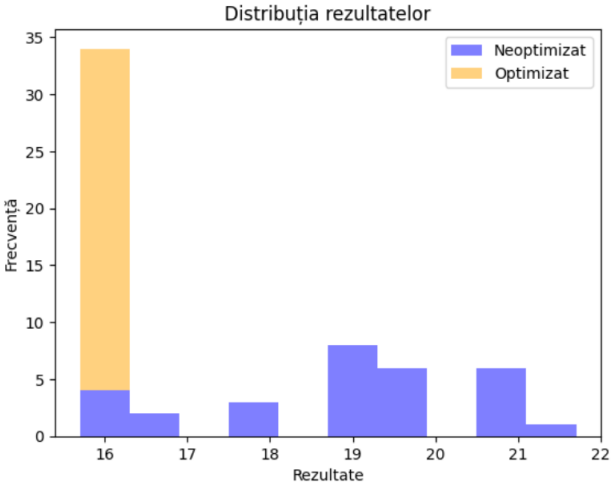

Figure 2

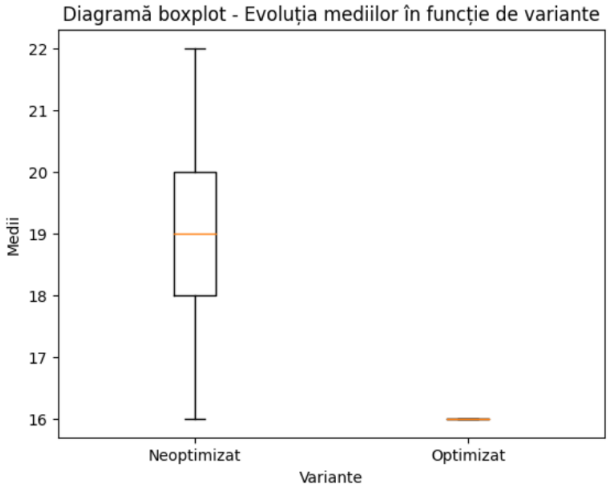

Figure 3

4.1.1 Testul-T:

P-valoarea rezultatului testului t: 7.53617559890954e-14, unde avem că H_0 = nu există diferență semnificativă între rezultatele algoritmului neoptimizat și cele ale algoritmului optimizat și H_a = există diferențe semnificative. P-valoarea fiind foarte mică putem concluziona că există o diferență semnificativă între rezultatele celor două variante ale algoritmului genetic.

4.1.2 Intervale de încredere:

Intervalul de încredere pentru media este: (18.43, 19.69), de unde se poate observa că media noastră se află în acest interval.

4.2 Set de test - myciel6

Variantă neoptimizată:

n	m	min	medie	max	best	sd	greedy	time
95	755	155	202.66	215	142	16.21	215	164s

Variantă optimizată:

n	m	min	medie	max	best	sd	greedy	time
95	755	143	172.80	204	142	19.69	215	252s

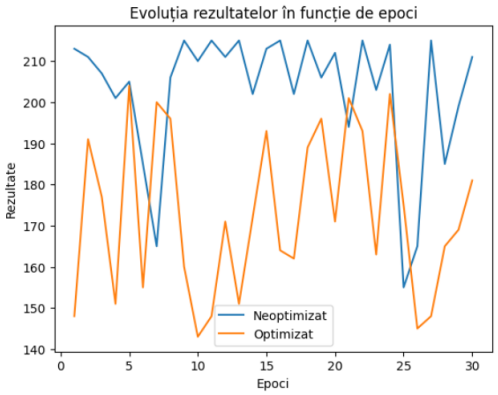

Figure 4

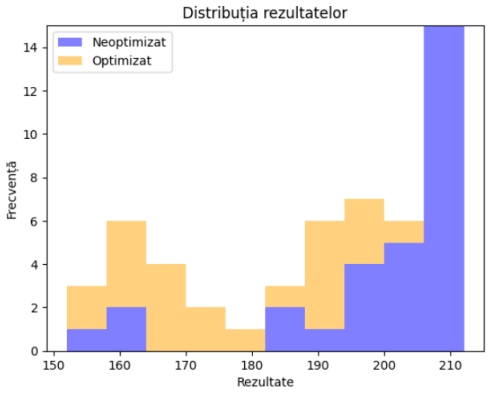

Figure 5

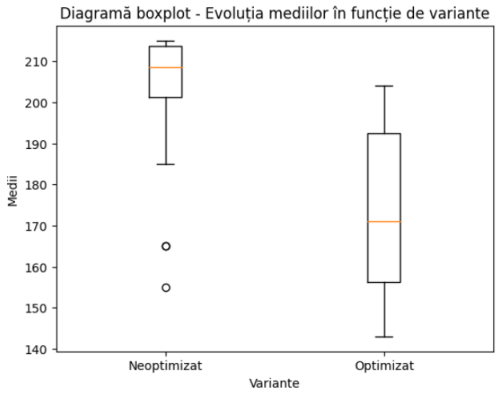

Figure 6

4.2.1 Testul-T:

P-valoarea rezultatului testului t: 2.8259828423812503e-08, unde avem că H_0 = nu există diferență semnificativă între rezultatele algoritmului neoptimizat și cele ale algoritmului optimizat și H_a = există diferențe semnificative. P-valoarea fiind foarte mică putem concluziona că există o diferență semnificativă între rezultatele celor două variante ale algoritmului genetic.

4.2.2 Intervale de încredere:

Intervalul de încredere pentru varianta neoptimizată este: (196.71, 208.62), de unde se poate observa că media noastră se află în acest interval. Pentru varianta optimizată avem intervalul (165.57, 180.02), unde la fel se poate observa că media noastră se află în acest interval.

4.3 Set de test - queen5_5

Variantă neoptimizată:

n	m	min	medie	max	best	sd	greedy	time
25	160	78	84.13	88	75	3.03	88	27s

Variantă optimizată:

n	m	min	medie	max	best	sd	greedy	time
25	160	75	77.20	81	75	2.12	88	41s

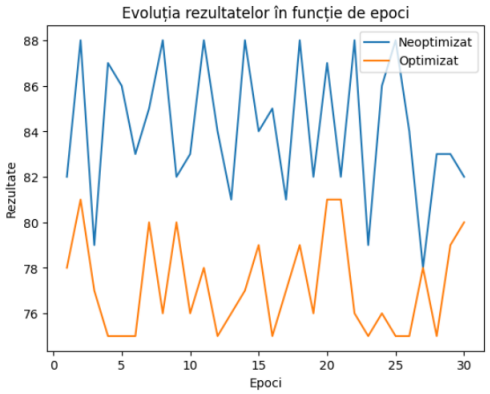

Figure 7

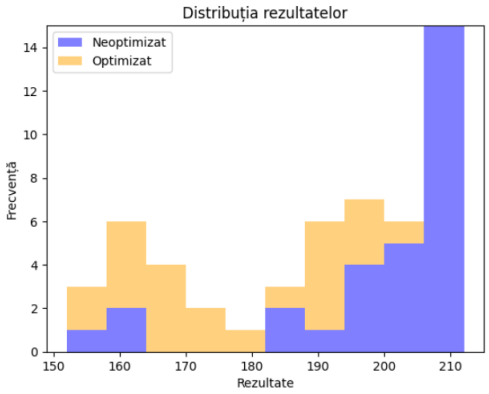

Figure 8

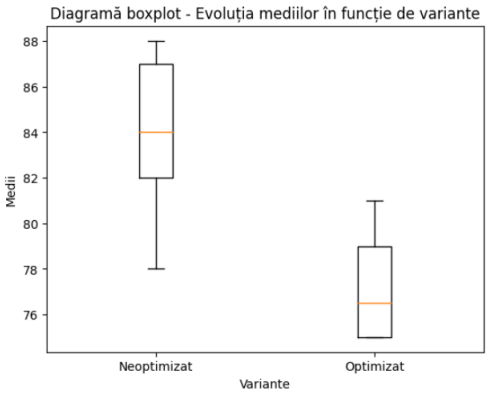

Figure 9

4.3.1 Testul-T:

P-valoarea rezultatului testului t: 1.2449570377371555e-14, unde avem că H_0 = nu există diferență semnificativă între rezultatele algoritmului neoptimizat și cele ale algoritmului optimizat și H_a = există diferențe semnificative. P-valoarea fiind foarte mică putem concluziona că există o diferență semnificativă între rezultatele celor două variante ale algoritmului genetic.

4.3.2 Intervale de încredere:

Intervalul de încredere pentru varianta neoptimizată este: (83.01, 85.24), de unde se poate observa că media noastră se află în acest interval. Pentru varianta optimizată avem intervalul (76.42, 77.97), unde la fel se poate observa că media noastră se află în acest interval.

4.4 Set de test - queen7_7

Variantă neoptimizată:

n	m	min	medie	max	best	sd	greedy	time
49	476	215	239.53	244	196	7.90	244	83s

Variantă optimizată:

n	m	min	medie	max	best	sd	greedy	time
49	476	199	216.66	234	196	9.55	244	121s

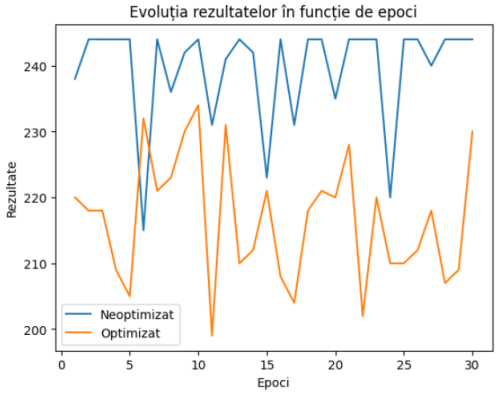

Figure 10

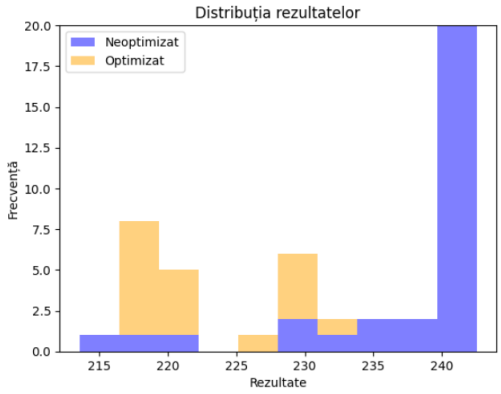

Figure 11

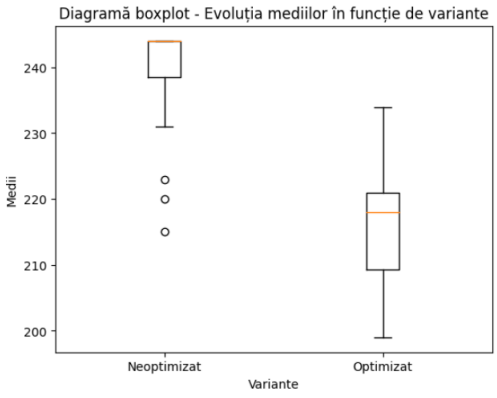

Figure 12

4.4.1 Testul-T:

P-valoarea rezultatului testului t: 2.150058856235666e-14, unde avem că H_0 = nu există diferență semnificativă între rezultatele algoritmului neoptimizat și cele ale algoritmului optimizat și H_a = există diferențe semnificative. P-valoarea fiind foarte mică putem concluziona că există o diferență semnificativă între rezultatele celor două variante ale algoritmului genetic.

4.4.2 Intervale de încredere:

Intervalul de încredere pentru varianta neoptimizată este: (236.63, 242.43), de unde se poate observa că media noastră se află în acest interval. Pentru varianta optimizată avem intervalul (213.15, 220.17), unde la fel se poate observa că media noastră se află în acest interval.

5 Concluzie

În urma algoritmului genetic implementat, putem observa că pe instanțele noastre de test alese, optimizarea operatorilor de mutație și încrucișare a avut un rezultat notabil, reusind pe cateva dintre instanțe să se atingă chiar valoarea minimă cunoscută în literatură. Din păcate instanțele sunt de dimensiuni mici si medii, prin urmare algoritmul nu este chiar în cea mai bună variantă a sa. S-au încercat si executarea algoritmului optimizat pe instanîe de test mai mari însă rezultatele sunt foarte slabe, iar timpul de execuție este foarte mare. Prin urmare acest algoritm, în continuare va trebui revizuit și încercat în aducerea sa la o variantă mult mai bună.

6 Bibliografie

References

[1] `https://www.geeksforgeeks.org/graph-coloring-set-2-greedy-algorithm/`

[2] Eiben, Agoston E., Jan K. Van Der Hauw, and Jano I. van Hemert. "Graph coloring with adaptive evolutionary algorithms." Journal of Heuristics 4.1 (1998): 25-46.

[3] `https://www.geeksforgeeks.org/project-idea-genetic-algorithms-for-graph-colouring/`

[4] Galinier, Philippe, and Jin-Kao Hao. "Hybrid evolutionary algorithms for graph coloring." Journal of combinatorial optimization 3.4 (1999): 379-397.

[5] Abbasian, R. and Mouhoub, M., 2011, July. An efficient hierarchical parallel genetic algorithm for graph coloring problem. In Proceedings of the 13th annual conference on Genetic and evolutionary computation (pp. 521-528).

[6] https://www.sciencedirect.com/science/article/abs/pii/S0020025516301335

[7] https://profs.info.uaic.ro/~eugennc/teaching/ga/

[8] https://profs.info.uaic.ro/~mionita/aea/index.html