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 Experiment

Algoritmul genetic a fost realizat folosind ca limbaj de programare Java, versiunea 19.0.2. Am ales pentru acest algoritm o populație de 200 de indivizi și un număr de 3000 de generații care trebuie efectuate. De asemenea, la nivel de operatori, am folosit o mutație de 0.1 și o încrucișare de 0.7.

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.

instanță	n	m	min	medie	max	best	sd	time
queen5_5	25	160	88	91.03	97	75	3.14	2s
queen7_7	49	476	244	254.1	268	196	7.21	7s
queen11_11	121	3960	928	976.71	1192	726	12.84	35s
myciel3	11	20	16	18.2	23	16	2.72	0.01s
myciel6	95	755	178	199.87	215	142	8.15	20s

4 Concluzie

Se poate observa faptul că, în urma experimentelor efectuate, algoritmul genetic clasic nu obtine rezultate foarte bune, pe instanțe relativ mici. Mai departe in evoluția algoritmului vom încerca optimizarea acestuia prin: optimizarea operatorilor (mutație și crossover) despre care am observat ca au un impact mare asupra algoritmului, căutarea unei metode adecvate pentru o hibridizare ce sperăm ca va aduce soluția mai aproape de optim și poate o încercare de elitism asupra populației noastre.

Pentru îndeplinirea obiectivelor enumerate în vederea unei optimizări a algoritmului, vom avea in atenție articolul:

`https://www.sciencedirect.com/science/article/abs/pii/S0020025516301335`