Erdős-Rényi véletlen gráfok színezése

Felsmann Dániel

A kutatómunkámban véletlen gráfok színezésével foglalkoztunk. A $G = (V, E)$ gráf csúcsainak egy k színnel történő jólszínezésén V egy olyan $V_1, ..., V_k$ partícióját értjük, ahol egyik V_i csúcs-halmaz (színosztály) sem feszít élt. Egy v csúcs színe annak a partíció halmaznak az indexe, amelyben szerepel. A gráf $\chi(G)$ kromatikus száma a legkisebb olyan k amely esetén van jólszínezés k színnel. Arra a kérdésre szerettünk volna választ kapni, hogy mennyi lehet egy 100 csúcsú véletlen gráf kromatikus száma.
A vizsgált gráfok Erdős-Rényi véletlen gráfok voltak, amelyeket úgy kapunk, ha az adott csúcshalmazon minden csúcspár $p = 1/2$ valószínűséggel feszít élt, a többi csúcspártól függetlenül. Ezzel a választással minden 100 csúcsú gráfot azonos valószínűséggel kaphatunk meg (az izomorf gráfokat különbözőnek tekintve). Mivel véletlen gráfokról van szó, ilyenkor $\chi(G)$ valószínűségi változó. Ennek a valószínűségi változónak a jellemző értékeire a jelenlegi legjobb elméleti becslés egy aszimptotikus eredmény ($p = 1/2$ esetben) (Heckel, 2016):

$$\chi(G) = \frac{n}{2\log_2 n - 2\log_2 \log_2 n - 2\log_2 2 + o(1)}$$

A kifejezés értéke az $n = 100$ esetben 17.17, ami a viszonylag kevés csúcs miatt a mi esetünkben nem egészen pontos.
Annak eldöntése, hogy egy gráf k színnel jólszínezhető-e, NP-teljes feladat, így a kromatikus szám pontos meghatározására nem remélhetünk polinom idejű algoritmust. Ezért egzakt módszerek hiányában különböző heurisztikus algoritmusokat implementáltunk, és hasonlítottuk össze a teljesítményüket a gráfszínezési feladat megoldásakor. A három, általunk kipróbált módszer a mohó színező algoritmus, a tabu színezés és egy genetikus algoritmus voltak. Az utóbbival sajnos nem sikerült értékelhető eredményeket elérnünk, így csak a másik kettőt mutatjuk be részletesen.

Mohó színezés. Az algoritmus inputjai egy G gráf, és a csúcsok egy permutációja. A csúcsokat a kapott permutáció szerinti sorrendben színezzük ki úgy, hogy a soron következő csúcs a legkisebb olyan színt kapja, amellyel eddig egyetlen szomszédját sem színeztük. Az algoritmus fontos tulajdonsága, hogy mindig jólszínezést ad eredményül.
Az eredmény szempontjából kiemelkedően fontos a kapott csúcssorrend. Egyrészt igaz, hogy mindig van olyan csúcssorrend, amely esetén a mohó színezés megtalálja az optimumot. Ha például G-nek ismerjük egy jólszínezését $\chi(G)$ színnel, akkor a színosztályok szerinti sorrendet használva a mohó színezés is ugyanennyi színt használ. Másrészt az is előfordulhat, hogy az algoritmus a kromatikus számnál sokkal több színt használ. Erre példa a koronagráfok esete, amelyek 2 színnel színezhetők, de van olyan csúcssorrend is, amely esetén a mohó színezés $|V|/2$ színt használ. A csúcssorrend megválasztására különböző stratégiákat próbáltunk ki:

- k legjobb: k darab véletlen permutáción futtatjuk az algoritmust, és csak a legjobb megoldást tartjuk meg.

- largest first: a csúcsokat a fokszámok szerint csökkenő sorrendbe rakjuk. Az azonos fokszámú csúcsokat tetszőleges sorrendben soroljuk fel.

- smallest-last: minden lépésben keresünk egy minimális fokú csúcsot, amit a sorrend végére teszünk, majd töröljük a gráfból ezt a csúcsot. A következő lépésben a kisebb csúcsszámú gráfban ismételjük az eljárást. A sorrend alapötlete az, hogy minimalizáljuk a korábbi csúcsokba vezető élek számát, így remélhetőleg kisebb színeket kaphatnak a csúcsaink a mohó színezés során.

- saturation largest first: ebben az esetben nem határozzuk meg előre a csúcsok sorrendjét, hanem az éppen következő csúcsot a színezés közben, dinamikusan választjuk ki. A gráf egy v csúcsának telítettségi foka (saturation degree) a v-vel szomszédos csúcsokon felhasznált színek száma. Az algoritmus minden lépésében megkeressük a legnagyobb telítettségi fokú csúcsot, és ennek adjuk a lehető legkisebb színt. Ha a legnagyobb telítettségű csúcs nem egyértelmű, akkor a legnagyobb telítettségi fokúak közül (az egyik) legnagyobb fokszámú csúcsot választjuk.

A mohó színezés eredménye javítható, ha felhasználjuk egy egyszerű megfigyelést. Ha a csúcssorrendben $V_1, V_2, ..., V_k$ független halmazok (színosztályok) követik egymást, akkor az algoritmus legfeljebb k színt használ. Azonban az is előfordulhat, hogy a mohó színező algoritmus kisebb színosztályokat egyesít. Ha tehát egy korábban kapott jólszínezés alapján úgy állítunk fel egy csúcssorrendet, hogy a színosztályok csúcsai egymást követik, akkor előfordulhat, hogy egy kevesebb színnel történő jólszínezést kapunk. Ilyenkor az a kérdés, hogy a színosztályokat milyenk sorrendbe érdemes tenni? Azt figyeltük meg, hogy nincs jelentős különbség aközött, hogy a színosztályok mérete szerint növekvő, vagy egy véletlen színosztály sorrendet használunk. Mivel az utóbbi gyorsabban megvalósítható, ezért mi azt választottuk. Azt is megfigyeltük, hogy érdemes többször is ismételni ezt a javító eljárást a jobb eredmények elérése érdekében.
A kísérleti eredményeket bemutató táblázat celláinak értékei azt mutatják, hogy a megfelelő algoritmus milyen arányban talált adott színnel jólszínezést (négy tizedes jegyre kerekítve). Az adatok 100000 darab $G_{100,1/2}$ gráfra vonatkoznak. Az utolsó oszlop az egy gráfra vonatkozó futásidőt tartalmazza ezredmásodpercben. Minden algoritmus-színszám párhoz két érték is tartozik. A cella alsó értéke a színosztályok szerinti sorrenddel javító algoritmus 10 futtatása utáni eredményeket mutatja, míg a felső érték a csak a mohó módszer alkalmazásával kapott eredmény.
Az eredmények azt mutatják, hogy érdemes a véletlen sorrend helyett más stratégiákat használni, mert így vagy gyorsan kaphatunk megfelelő színezésszámokat, mint például a largest first (csökkenő fokszámok) sorrenddel, vagy egész jó színezésszámokat kapunk, mint a saturation largest first algoritmussal, csak kicsit hosszabb idő alatt. A javító módszerünk használata minden esetben jelentős javulást eredményezett. Itt külön nem írtam a futásidőt, minden esetben 10 mohó algoritmust futtattunk a javításhoz, ezért az ehhez szükséges idő nagyjából a 10 legjobb algoritmus futásideje.

	16	17	18	19	20	>20	Futásidő (ms)
10 legjobb	-	-	0.0006	0.0201	0.1661	0.8132	0.9111
	0.0002	0.0275	0.3832	0.5301	0.0588	0.0002	
100 legjobb	-	-	0.0006	0.0199	0.1643	0.8152	9.0694
	0.0001	0.0257	0.3843	0.5301	0.0596	0.0002	
Largest first	-	0.0010	0.0415	0.3020	0.4608	0.1947	0.1654
	0.0003	0.0438	0.4659	0.4586	0.0314	0.0001	
Smallest last	-	0.0002	0.0168	0.1722	0.4312	0.3796	1.9934
	0.0004	0.0482	0.4741	0.4473	0.0299	0.0001	
Saturation largest first	0.0036	0.1317	0.5108	0.3133	0.0394	0.0012	2.7447
	0.0058	0.2074	0.6088	0.1749	0.0031	-	

1. táblázat: A mohó színezés eredményei.

A javító algoritmusunk annyira jól működött, ami önálló algoritmusként is alkalmazhatóvá tette. Első lépésben a mohó színezéssel kiszínezzük a gráfot egy véletlen csúcssorrend szerint. Ezután a színosztályok egy véletlen sorrendje szerint rakjuk sorba a gráf csúcsait, és eszerint színezünk a mohó algoritmussal. A második lépést ismételjük k-szor. A következő táblázat ennek az algoritmusnak az eredményeit mutatja 10000 gráfra különböző k értékek mellett. A kapott színezésszámok javulnak, a nem javított mohó módszerekhez képest 1-2 szín javulást jelentve, de a futásidő jelentősen megnő, hiszen minden esetben (k+1 alkalommal kell mohón színeznünk a gráfot. Összehasonlításul a $k := 100$ eset futásideje nagyjából a 100 legjobb algoritmuséval azonos. Azt tapasztaltuk, hogy $k := 1000$ iteráció felett már nem tapasztalható lényeges javulás, azonban kiemelendő, hogy ebben a legjobban színező esetben már sokszor 15 vagy 16 szín is elég, és egyszer sem használtunk 17 színnél többet.

	15	16	17	18	19	>19	Teljes futásidő (s)
$k := 10$	-	-	0.0216	0.4223	0.5297	0.0264	7.85
$k := 100$	-	0.0367	0.6582	0.3051	-	-	77.42
$k := 500$	0.0018	0.3451	0.6521	0.0010	-	-	385.172
$k := 1000$	0.0073	0.6072	0.3855	-	-	-	768.753

2. táblázat: A javított mohó színezés eredményei.

Tabu színezés. Az általunk kipróbált algoritmusok közül a tabu színezés (vagy más néven tabu keresés) tudta a gráfokat a legkevesebb színnel jólszínezni. A tabu keresés egy heurisztikus optimalizáló algoritmus, amelynek az általános működése a következőképpen foglalható össze. A lehetséges megoldások S halmazán szeretnénk minimalizálni az $f : S \to \mathbb{R}$ célfüggvényt. Minden $s \in S$ megoldáshoz definiáljuk a megoldás $N(s)$ szomszédait. Valamilyen $s_0 \in S$ megoldással indulunk. Ezután minden lépésben kiválasztjuk az aktuális s_i megoldás legkisebb célfüggvény-értékű s^* szomszédját, amennyiben s^* nem szerepel a tabu listán. Ugyanis a futás során végig fenntartunk egy rögzített hosszú T tabu listát, amelyet folyamatosan frissítünk: ha az s megoldásból s'-be léptünk, akkor az $s' \to s$ lépés bekerül a tabu listába, a legrégebben T-be került lépést pedig töröljük. Ha a legjobb szomszédos megoldás szerepel a tabu listán, akkor egy másikat választunk helyette. Ez azt hivatott elősegíteni, hogy a keresés ne térjen vissza korábban már felkeresett megoldásokba, és ne ragadjon meg egy-egy lokális optimumhelyen. Az általános lépés a gyakorlatban úgy történik, hogy adott s_i megoldás esetén s_i-nek csak rögzített számú szomszédját generáljuk le, és ha véletlenül a tabu listán szereplő szomszédot generáltunk,

akkor ezt figyelmen kívül hagyjuk, és helyette generálunk egy másikat. Az így kapott szomszédos megoldások közül a legjobbat választjuk, frissítjük a tabu listát, és folytatjuk az iterációt. Az algoritmus akkor áll le, ha elértünk egy kívánt f^* célfüggvény értéket, vagy egy előre definiált maximális iteráció számot.

A tabu keresés gráfszínezésre történő alkalmazásakor Hertz és de Werra 1987-es cikkét vettük alapul. A lehetséges megoldások $s = \{V_1, V_2, ..., V_k\}$ ahol a V_i halmazok $V(G)$ egy partícióját alkotják. Így minden megoldás egy k színnel színezésnek felel meg. Az induló megoldás egy véletlen színezés k színnel. Mivel az induló megoldás választása nagy hatással van az algoritmus eredményére, minden gráf színezésekor 10 különböző induló megoldással is futtattuk az algoritmust. Az algoritmus hátránya, hogy k értékét előre meg kell adnunk, az output pedig nem biztos, hogy jólszínezés lesz.
A célfüggvényben minden partíció halmazra a partíción belüli élek számát összegezzük:

$$f(s) = \sum_{i=1}^{k} |E(G[V_i])|$$

, ahol $G[V_i]$ a V_i pontok által feszített részgráf. A célfüggvény értéke tehát a gráf olyan éleinek száma, amelyeknek mindkét végpontja azonos színű (ezek a "rossz" élek). A cél így egy olyan megoldás megkeresése, amelyre $f = 0$. Ez pontosan akkor teljesül, ha s jólszínezés. Egy megoldás szomszédait úgy kapjuk meg, hogy $V(G)$ valamelyik s-ben V_i-beli pontját áttesszük a V_j partíció halmazba. Mivel f minimalizálása a cél, csak azok a szomszédok az érdekesek számunkra, amikor valamelyik partíción belüli él egyik végpontját rakjuk át másik partíció halmazba. A tabu listát úgy frissítjük, hogy ha a v csúcsot tesszük át az aktuális lépésben az i-edik színosztályból a j-edikbe, akkor (v, i) lesz a tabu lista legújabb eleme: a v csúcsot egy darabig nem tehetjük vissza V_i-be. A tabu lista legrégebbi elemét töröljük.
Az algoritmus paraméterei a felhasználható színek k száma, a lépésenként generált szomszédos megoldások N száma, a tabu lista $|T|$ mérete, és a maximálisan engedélyezett iterációk száma. Nekünk az $N = 30$, $|T| = 7$ választás, és a maximum 3000 iteráció vált be leginkább.

Input	$G = (V, E)$
	k a felhasználható színek száma
	$\|T\|$ a tabu lista mérete
	N a lépésenként generált szomszédok száma
	maxiter: az iterációk maximális száma
Inicializálás	s véletlen színezés generálása
	i := 0
	$T := \emptyset$
while $f(s) > 0$ és $i <$ maxiter	
	s N szomszédjának generálása
	s^* ezek közül a legjobb, ami nem tabu
	$s := s^*$
	T frissítése
	$i := i + 1$
Output	Ha $f(s) = 0$, akkor egy színezés k színnel
	ha $f(s) > 0$, akkor nem találtunk jó színezést

3. táblázat: A tabu színezés algoritmus működése.

A tabu színezés eredményeit bemutató táblázatban a sorok a színszámokat jelölik. Az első oszlop azt mutatja, hogy milyen arányban sikerült a tabu színezésnek az adott számú színnel jó színezést találni. A következő oszlopban és a teljes futásidő szerepel. Az adatok 1000 gráfra vonatkoznak.
Az eredmények alapján majdnem mindig sikerült 15 színnel színeznünk a gráfokat, 16 színnel pedig mindig. Ritkán az is előfordult, hogy 14 szín elég volt. A tabu színezés futásideje jóval hosszabb bármelyik korábbi mohó algoritmusnál, azonban a kapott színszámok is jobbak lettek. Összegezve az eddigieket a sejtésünk az, hogy $G_{100,1/2}$ kromatikus száma nagy valószínűséggel legfeljebb 15 lehet.

Színszám	Sikeres színezések aránya	Teljes futásidő (s)
14	0.023	3206.44
15	0.922	774.97
16	1	41.82

4. táblázat: A tabu színezés eredményei.

Hivatkozások

- A. Hertz, D. de Werra - Using Tabu Search Techniques for Graph Coloring, 1987
- A. Heckel - Coloring of Random Graphs - PhD dolgozat, 2016