Kaartkleuren

Algoritmes voor het inkleuren van grafen

door Kim de Bie, Jeroen de Jong & Rik Volger

Heuristieken, drs. Daan van den Berg (Universiteit van Amsterdam)

18 december 2015

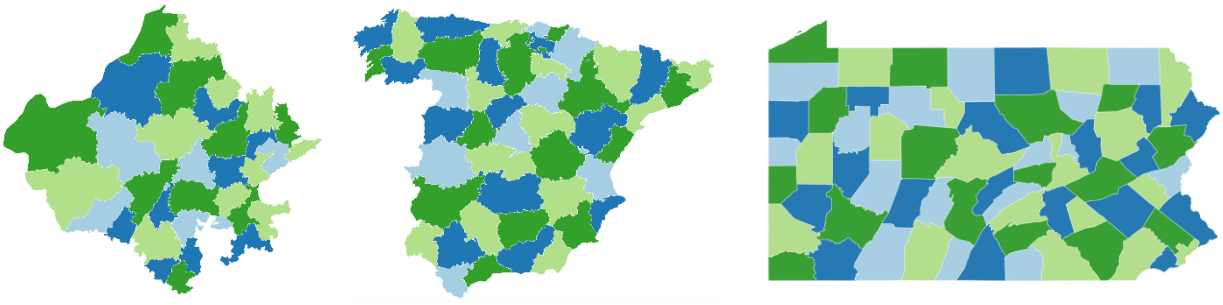
Samenvatting: In dit verslag wordt een aanpak van het grafenkleurprobleem gepresenteerd. De verschillende algoritmes en heuristieken die zijn gebruikt en toegepast op geografische kaarten en sociale netwerken worden uitgelegd en de resultaten worden met elkaar vergeleken. Het blijkt dat een depth-first aanpak gecombineerd met een greedy-coloring algoritme tot goede resultaten leidt. Daarnaast worden enkele voorbeelden van bredere, praktische toepassingen van dit algoritme opgesomd.

Introductie

Een graaf is een verzameling punten (of knopen) die verbonden worden door lijnen (of kanten). Grafen kunnen worden gebruikt om zeer uiteenlopende situaties in kaart te brengen, bijvoorbeeld de structuur van de atomen in een molecuul of de wereldwijde vliegverbindingen van een luchtvaartmaatschappij. Een belangrijk aspect van de grafentheorie is het kleuren van grafen. Een graaf moet zo ingekleurd worden dat geen enkel duo van knopen die direct met elkaar zijn verbonden dezelfde kleur krijgt, maar het kleurgebruik moet tegelijk zo efficient mogelijk zijn: het gehele netwerk moet ingekleurd worden met zo min mogelijk kleuren.

In dit verslag wordt het kleuren van grafen toegepast op twee verschillende grafentypen, namelijk geografische kaarten en sociale netwerken. De kaartvariant van het grafen-kleurprobleem is relevant voor cartografen: voor de oogrust en duidelijkheid is het belangrijk dat een kaart geen kleurconfetti is, maar tegelijk moeten buurlanden duidelijk van elkaar te onderscheiden zijn en dus een verschillende kleur hebben. In deze context wordt vaak het zogenaamde Four Color Theorem aangehaald, dat stelt dat grafen – mits ze voldoen aan een aantal eisen – ingekleurd kunnen worden met maximaal vier kleuren.[1] Het inkleuren van sociale netwerken heeft bredere toepassingen dan visualisatie alleen. Het toekennen van een (kleur)categorie die niet gedeeld wordt door verbonden knopen is bijvoorbeeld nuttig wanneer een netwerk blootgesteld wordt aan

advertenties, waarbij de marketeer het belangrijk vindt dat niemand dezelfde reclame te zien krijgt als haar connecties.



Figuur 1 – De drie kaarten waarop we ons algoritme hebben toegepast.

Toestandsruimte

Per definitie zijn alle kaarten en netwerken grafen zoals hierboven beschreven. Kaart 1, van een provincie in Noord-India (Rajasthan), bevat 33 knopen; kaart 2 (Spanje) bevat er 47; kaart 3 (Pennsylvania) bevat er 68. De netwerken bestaan respectievelijk uit 99, 98 en 100 knopen. De toestandsruimte, oftewel het aantal verschillende manieren om een casus (hier: het inkleuren van een graaf) op te lossen, wordt gegeven door C^V, waar V het aantal knopen (*vertices*) is en C het chromatisch getal: het minimaal aantal kleuren dat nodig is om de graaf in te kleuren. Het berekenen van het chromatisch getal is echter een zeer lastig rekenkundig probleem op zich[2]; in die zin is de exacte toestandsruimte dus onbekend, en als zodanig theoretisch oneindig groot. Echter, wanneer we uitgaan van het Four Color Theorem en veronderstellen dat een willekeurige kaart gemiddeld vier kleuren nodig heeft, komt de toestandsruimte voor de kaart van Noord-India (de graaf met het kleinste aantal connecties) bijvoorbeeld uit op 4^{33}, oftewel $7.4 * 10^{19}$. Dit is al bijna te groot om middels een 'brute force'[a] aanpak door te rekenen. Zelfs met drie kleuren, realistisch gezien het kleinste chromatisch getal dat mogelijk is, komen we voor Spanje uit op een toestandruimte van $3 * 10^{22}$. Hiermee is het grafen-kleurprobleem bij uitstek een probleem dat vraagt om een slimmere oplossing.

Methodes

Het doel van onze aanpak was het behapbaar maken van de toestandsruimte (die in theorie – zo lang het chromatisch getal onbekend is – oneindig groot is). In deze sectie beschrijven we de drie algoritmes die we hiervoor ontworpen hebben. De technieken die we geïmplementeerd hebben zijn uiteindelijk cumulatief toegepast; een nieuwe versie van ons algoritme is dan ook steeds een uitbreiding van de vorige (en niet een algoritme dat *from scratch* is opgebouwd). Deze aanpak stelt ons in staat om steeds het effect van nieuw toegevoegde heuristieken te kunnen analyseren.

Versie 1: depth-first, random volgorde met pruning

In het beginsel werken alle algoritmes die we ontworpen hebben op basis van een depth-first aanpak. Dit wil zeggen dat de knopen steeds één voor één worden ingekleurd. Als er na een kleurings-stap wordt vastgesteld dat de huidige kleuring niet tot een oplossing kan leiden, wordt de laatst gekleurde knoop

[a] Binnen een brute-force aanpak worden mogelijke oplossingen in een willekeurige volgorde doorlopen. De grens voor het doorrekenen middels brute force wordt gelegd op 10^{20}.

opnieuw (met een andere kleur) ingekleurd. In theorie wordt zo de hele toestandsruimte doorlopen tot er een oplossing gevonden is – een pure depth-first aanpak verkleint de toestandsruimte dus nog niet significant.

Om de toestandsruimte toch te kunnen verkleinen, hebben we binnen deze eerste versie van ons algoritme hebben we een vorm van 'pruning', oftewel het uitsluiten van bepaalde 'takken' van de toestandsruimte toegepast. Onze pruningtechniek werkt als volgt. Wanneer knoop A bijvoorbeeld rood gekleurd wordt, is het voor de knopen die in directe verbinding staan met A al onmogelijk geworden om óók rood gekleurd te worden. Op dit moment hebben we dus de kleur rood bij de buren van A verwijderd uit de lijst met mogelijke kleuren. Op deze manier worden delen van de toestandsruimte waarvan we *zeker* weten dat ze niet tot een oplossing leiden verwijderd. Het algoritme voert het kleuringsproces uit met een vastgesteld aantal kleuren. Als er met dit aantal kleuren geen oplossing mogelijk blijkt, wordt er niet automatisch een extra kleur toegevoegd; aan de andere kant wordt er ook niet geprobeerd om een graaf met *minder* kleuren in te kleuren zodra een oplossing gevonden is.

Zoals gezegd kleurt dit algoritme de knopen steeds één voor één met een bepaalde volgorde in. De volgorde waarin landen worden ingekleurd is willekeurig (de volgorde wordt stochastisch bepaald) en ligt niet vast. De kleuren die gebruikt worden staan wel in een vastgestelde volgorde. Dit betekent dat voor elke knoop steeds éérst kleur 1 wordt geprobeerd. Pas als dit niet meer mogelijk is, wordt kleur 2 gebruikt, en alleen als kleur 2 ook al gebruikt wordt door een aanliggende knoop, wordt er van kleur 3 gebruik gemaakt enzovoort. Deze vorm van het inkleuren van netwerken, waarbij gewerkt wordt met een vaste knoopvolgorde en geprobeerd wordt zoveel mogelijk knopen met één kleur in te kleuren, staat bekend als 'greedy coloring'.[3] Bij greedy coloring wordt de knopenlijst echter slechts eenmaal doorlopen en wordt er steeds een nieuwe kleur toegevoegd als er geen mogelijkheden met de al gebruikte kleuren bestaan. Ons algoritme werkt anders, in de zin dat het met een maximum aantal kleuren werkt, en een nieuwe knoopvolgorde uitprobeert als dit vastgestelde maximum bereikt is (en niet toereikend blijkt).

Versie 2: single-option prioriteit

De tweede versie van ons algoritme is een vrij simpele uitbreiding van het vorige. We constateerden dat relatief veel problemen werden veroorzaakt doordat het algoritme knopen 'insloot' met alle mogelijke kleuren; er ontstonden zo regelmatig situaties waarbij er voor een knoop geen kleur-opties meer overbleven. Daarom hebben we ervoor gezorgd dat in deze volgende versie van ons algoritme een knoop prioriteit krijgt zodra deze nog maar in één kleur ingekleurd kan worden: de knoop wordt in deze situatie onmiddellijk ingekleurd. Het aantal mogelijke kleuren konden we steeds vaststellen doordat we – ook in de eerste versie van het algoritme, aangezien dit nodig is voor het prunen en het greedy-coloring mechanisme – bij elke kleurstap registreren welke kleur-opties er wegvallen voor de buren van de zojuist gekleurde knoop (en dus ook welke kleur-opties er nog over zijn). Door het prioriteren van knopen met nog maar één kleurmogelijkheid denken we situaties voor te zijn waarin er geen kleur-opties meer over zijn voor een knoop.

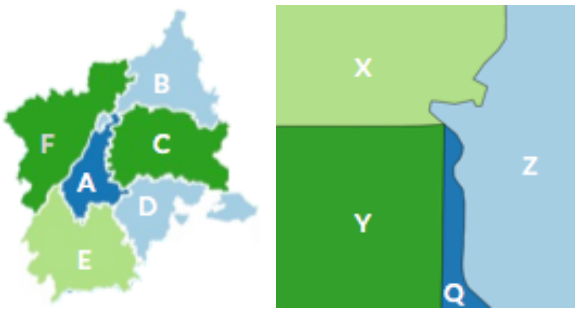
Versie 3: sorteren op aantal connecties

In de derde versie van ons algoritme, dat opnieuw een uitbreiding is van de vorige versie, hebben we het greedy-coloringmechanisme (oftewel: kleuren in een lijst afwerken, en zo lang mogelijk dezelfde kleur gebruiken) verder geperfectioneerd. Waar in versie twee gewerkt wordt met een willekeurige lijstvolgorde,

hebben we die in deze versie steeds gemuteerd: de lijst is gesorteerd op het aantal connecties dat een knoop heeft. Clusters van knopen waartussen een relatief hoog aantal verbindingen bestaat leveren relatief snel problemen op; vaak zijn alle kleur-opties nodig op deze dichtbevolkte kruispunten, en het werkt dus extra vertragend als de kleuropties beperkt worden doordat aanliggende knopen al een cruciale kleur bezetten. Dit soort situaties hopen we opnieuw voor te zijn. Deze vorm van greedy coloring, waarbij het aantal connecties van een knoop leidend is voor de lijstvolgorde, staat bekend als het Welsh-Powell algoritme.[4] Ons algoritme is hier opnieuw een variatie op, omdat het gecombineerd wordt met een depth-first aanpak, en dus doorzoekt naar een oplossing ook nadat de lijst éénmaal doorlopen is. Het prunen en het prioriteren van knopen met nog maar één kleurmogelijkheid blijft overigens overeind zoals in de hierboven beschreven versies van het algoritme.

Versie 4: automatisch bepalen van het aantal kleuren

In de vorige versies van ons algoritme werd het aantal kleuren steeds handmatig vastgelegd. Zoals genoemd heeft dit als resultaat dat het algoritme ofwel geen resultaat vindt, ofwel een resultaat vindt waarvan niet met zekerheid gezegd kan worden dat het het chromatisch getal heeft gebruikt. In deze laatste versie van ons algoritme hebben we een oplossing voor dit probleem geïntroduceerd. Het algoritme stelt eerst vast welk aantal kleuren minimaal nodig is om de graaf te kleuren. Het criterium dat hiervoor gebruikt wordt is het volgende. Op het moment dat een knoop die niet aan de rand van een gebied zit en een oneven aantal aanliggende knopen heeft, is het chromatisch getal minimaal gelijk aan vier. Op figuur 2A is te zien dat knoop A een unieke kleur heeft, want alle andere knopen zijn daarmee verbonden. De omliggende knopen (B, C, D, E, F) moeten dus andere kleuren krijgen. Omdat dit aantal oneven is (vijf), kan dit niet met twee kleuren gedaan worden (om en om), maar zijn er minimaal drie kleuren vereist. Deze methode van het chromatisch getal bepalen houdt geen stand bij veel andere toepassingen (waaronder de netwerken) omdat bij geografische kaarten per definitie sprake is van 2D. Elke omliggend land heeft hierbij minimaal twee connecties die ook verbonden zijn met het centrale land. Dat hoeft niet zo te zijn bij bijvoorbeeld een sociaal netwerk. Een techniek die wel altijd wordt toegepast om het chromatisch minimum te bepalen is het bijhouden van het aantal direct verbonden knopen, oftewel knopen die allemaal direct met elkaar in verbinding staan (clusters). Het chromatisch getal is minstens gelijk aan het grootste cluster wat zich in de graaf bevindt. In figuur 2A wordt dit duidelijk m.b.t. (onder andere) knoop A, B en C. Aangezien deze knopen allemaal onderling verbonden zijn, is het chromatisch getal minstens drie. Knopen zijn niet aan elkaar verbonden als het om een raakpunt gaat. Figuur 2B geeft dit weer. Q en X zijn geen direct verbonden knopen, en ook Y en Z niet. (Het minimum chromatisch getal voor dit kaartfragment is dus twee.)



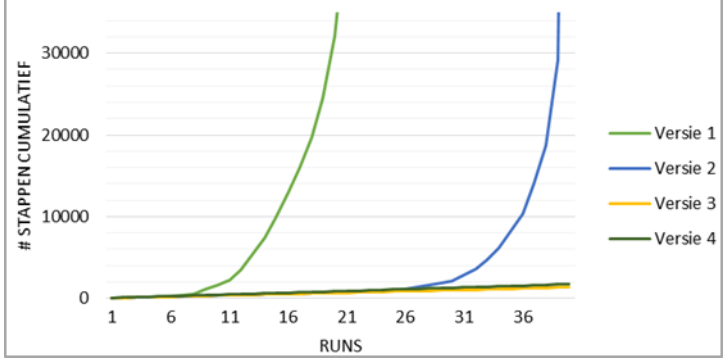
Figuur 2A en 2B – Oneven aantal buurknopen en verbindingspunten.

Merk hierbij op dat het aantal vereiste kleuren direct gerelateerd is aan de toestandsruimte: deze wordt groter als er veel kleuren nodig zijn, ofwel als er veel connecties zijn met overlappende connecties (grote 'clusters'). Het algoritme neemt de lage inschatting van het chromatische getal als vertrekpunt, en probeert eerst of de graaf ingekleurd kan worden met dit aantal kleuren. Op het moment dat er geen oplossing gevonden wordt met dit chromatisch getal wordt er een kleur toegevoegd. Op deze manier vindt het algoritme gegarandeerd de oplossing met het laagste aantal kleuren mogelijk.

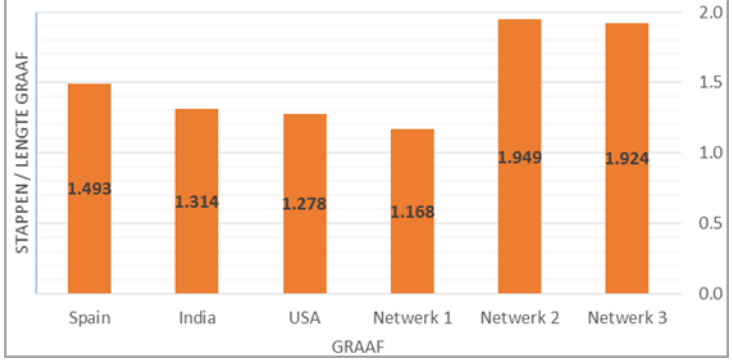
Resultaten

Figuur 3 vergelijkt de performance van de vier verschillende algoritmes. Het aantal kleurstappen dat nodig was om een oplossing te vinden in veertig runs wordt cumulatief weergeven. De figuur toont aan dat de heuristieken en pruningtechnieken zoals doorgevoerd in de verschillende versies van ons algoritme het inkleurings-proces steeds iets versnellen. De figuur toont dus aan dat het prunen en het manipuleren van de volgorde van de knopen (zowel door te sorteren op aantal connecties als het prioriteren van landen met nog maar één mogelijkheid) het gewenste effect heeft. Wel zien we dat de laatste versie van het algoritme een kleine vertraging doorvoert. Dit ligt binnen de lijn van verwachting: het algoritme begint met het absolute minimum aantal kleuren dat nodig is. Wanneer het werkelijke chromatisch getal hoger ligt, moet alsnog de gehele toestandsruimte (min de geprunede takken) van het geschatte minimum doorlopen worden. De winst die behaald wordt met dit algoritme weegt echter op tegen de vertraging: in tegenstelling tot de eerste drie versies garandeert deze laatste versie van het algoritme dat een zo goed mogelijke oplossing gevonden wordt. Een verbetering zou dus nog doorgevoerd kunnen worden door het chromatisch getal exacter te schatten, en daardoor niet onnodig toestandsruimten met een onmogelijk aantal kleuren door te rekenen.

Figuur 3 - Performance van alle versies



Figuur 4 - Gemiddelde performance versie 4



Figuur 4 geeft weer hoe de laatste versie van ons algoritme presteert op de verschillende grafen. Om de verschillende grafen goed te kunnen vergelijken, is het gemiddeld aantal kleurstappen (over een steekproef van vijftig runs) dat het algoritme nodig heeft om tot een oplossing te komen gedeeld door de lengte van de graaf. Deze grafiek toont aan dat het aantal knopen niet per se bepalend is voor de moeilijkheidsgraad van het inkleuren van een graaf – althans, via de methoden in het beschreven algoritme. Netwerk 1 heeft (samen met netwerk 2 en 3) het hoogste aantal knopen, maar wordt toch relatief het snelst ingekleurd. Aangezien

netwerken (die gemiddeld genomen wel moeilijker lijken in te kleuren dan kaarten; zie netwerk 2 en 3) doorgaans grotere clusters kunnen bevatten dan geografische kaarten – 8 landen grenzen nu eenmaal niet vaak allemaal aan elkaar – lijkt het zo te zijn dat het aantal clusters wel een factor is in het bepalen van de moeilijkheidsgraad. Verdere analyse moet verder aantonen wat de eenvoudigere grafen van de moeilijkere onderscheidt.

Conclusies

Hoewel kaarten en sociale netwerken het theoretisch vertrekpunt vormden voor onze aanpak, kan het ontworpen algoritme toegepast worden op een veel groter aantal problemen. Een kant (verbinding) tussen twee punten kan gezien worden als een *conflict*: twee verbonden knopen moeten ingedeeld worden in een verschillende (kleur)categorie en verbonden knopen mogen niet in dezelfde categorie terechtkomen. Een interessant voorbeeld van een praktische toepassing hiervan is het samenstellen van roosters. Als bijvoorbeeld vakken die door dezelfde klas worden gevolgd worden gezien als verbonden knopen, is het minimaal aantal kleuren dat nodig is om de vakken-graaf in te kleuren (het chromatisch getal) gelijk aan het aantal tijdvakken dat nodig is om deze vakken in te roosteren.[4] Een andere situatie waarin graaf-kleurtheorie praktisch wordt toegepast is bij het toekennen van radiofrequenties. Wanneer het signaal van zendmasten elkaar overlapt, moeten deze op een unieke frequentie uitzenden: verbonden punten moeten van een unieke categorie zijn.[5] Het hierboven beschreven algoritme is flexibel genoeg om ook voor deze problemen – hopelijk nog steeds binnen een redelijke tijdsspanne – een zo goed mogelijke oplossing te vinden.

Referenties

1. Gonthier, G. A computer-checked proof of the four colour theorem. (2005). at <http://research.microsoft.com/~gonthier/4colproof.pdf>
2. Karp, R. in *50 Years of Integer Programming 1958-2008* at <https://www.infona.pl/resource/bwmeta1.element.springer-2dcfb85e-11c2-3500-87e4-870a4515cdc2>
3. Graph algorithms. (2009). at <http://www.cs.cornell.edu/courses/cs3110/2009sp/recitations/rec22.html>
4. Welsh, D. J. A. & Powell, M. B. An upper bound for the chromatic number of a graph and its application to timetabling problems. *Comput. J.* **10,** 85–86 (1967).
5. Schulz, M. & Eisenblätter, A. Solving frequency assignment problems with constraint programming. (2012). at <http://www.zib.eu/groetschel/students/mathias_schulz_diplom.pdf>