KU LEUVEN kulak

Bachelor in de Informatica
Gegevensstructuren&Algoritmen
Jan Goedgebeur & Jarne Renders

Onderzoek van het Tabu 1-exchange algoritme voor het Grafenkleuringsprobleem

Academiejaar 2020 – 2021

Dieter Demuynck

Inleiding

Het Grafenkleuringsprobleem (graph coloring problem of GCP) is een probleem in de grafentheorie met vele applicaties. In het grafenkleuringsprobleem probeert men aan elke knoop in een graaf een kleur toe te kennen, zodat elke 2 verbonden knopen een verschillend kleur hebben, door zo weinig mogelijk kleuren te gebruiken. Een exacte oplossing voor dit probleem vraagt echter vaak veel rekenwerk en tijd, waardoor er meestal heuristische algoritmen worden gebruikt om op een efficiënte manier een geldige kleuring te vinden.

In deze paper gebruiken we een combinatie van een reductiealgoritme, een constructiealgoritme en een stochastisch lokale zoekalgoritme om een kleuring te vinden voor bepaalde grafen.

1 De gebruikte algoritmen

In deze sectie worden de verschillende gebruikte algoritmen besproken om een kleuring te vinden van een graaf.

1.1 Het reductiealgoritme

Het eerste toegepaste algoritme is het reductiealgoritme. Het doel van het reductiealgoritme is om knopen uit de graaf te verwijderen, zodat er minder rekenwerk nodig is om een kleuring te vinden, en ook zonder het uiteindelijke kleurenaantal te wijzigen. Het gebruikte algoritme werkt als volgt: De knopen in de graaf worden twee aan twee vergeleken. Als de twee knopen verbonden is met exact dezelfde knopen, of één knoop is verbonden met alle knopen die verbonden zijn met de tweede knoop (dit kan alleen wanneer de twee knopen niet verbonden zijn met elkaar), dan weten we dat deze knopen zeker een gelijk kleur kunnen hebben. Dit omdat de kleur van de knoop met het grootste aantal verbindingen verschillend moet zijn met de kleuren van de verbonden knopen. In de implementatie wordt dit gedaan door elke knoop te vergelijken met elke andere knoop, en als de buren van de knopen overeenkomen, of de buren van een knoop een deelverzameling is van de buren van de andere knoop, dan geven we aan die knoop een label dat deze gereduceerd is, en met welke knoop deze overeenkomt, en dan wordt de knoop uit een kopie van de verzameling van alle knopen verwijdert door hem daarin aan null gelijk te stellen. Na alle knopen te vergelijken, worden alle null-elementen verwijderd.

Function preprocess(graph)

```
graph.validVertices = copy(graph.vertices);
foreach vertex1 ∈ graph.vertices do
    foreach vertex2 ∈ graph.vertices do
        if vertex1.adjacentVertices ⊆ vertex2.adjacentVertices then
            vertex1.reduceTo(vertex2);
            graph.validVertices.set(vertex1, null);
        end
    end
end
graph.validVertices.removeAll(null);
```

1.2 Het constructiealgoritme

Het volgend algoritme dat toegepast wordt op de graaf is een constructiealgoritme, nl. het graadsaturatie-algoritme (Degree Saturation of DSATUR). Een constructiealgoritme is bedoeld om een kleuring van de graaf op te stellen, onafhankelijk of deze volledig is (alle knopen krijgen een kleur) of geldig is (verbonden knopen hebben een verschillend kleur). Het DSATUR algoritme creëert een volledige en geldige kleuring op de volgende manier: Eerst sorteert het alle knopen aflopend op graad. Daarna wordt de knoop met hoogste saturatie uit die verzameling gekozen. De saturatie van een knoop is het aantal verschillende kleuren van de verbonden knopen. Als twee knopen een gelijke saturatie hebben, dan wordt de knoop met hoogste graad gekozen. Als beide knopen dan ook een gelijke graad hebben, wordt willekeurig een knoop gekozen. In de implementatie echter wordt de eerste knoop gevonden in de lijst, aangezien dit zo goed als willekeurig is. De gekozen knoop krijgt dan een kleur, verschillend van al zijn buren. Als de knoop verbonden is met elk kleur dat al bestaat, dan krijgt hij een nieuw kleur. Er worden steeds nieuwe knopen gekozen tot elke knoop een kleur heeft. De knoop met grootste saturatie wordt gevonden door elke knoop te overlopen en per knoop de saturatie ervan uit berekenen, door het aantal verschillende kleuren bij zijn buren te berekenen. Of een kleur al gevonden is of niet in deze laatste stap, wordt bijgehouden in de implementatie in een BitSet die even groot is als het aantal kleuren tot nu toe. De index van de BitSet stelt een kleur voor, en de status van de bit stelt voor of het kleur met die index al gevonden is bij een knoop zijn buren.

```
Function constructColoring(graph)
    uncoloredCount ← graph.vertices.size;
    graph.colorCount ← 0;
    while uncoloredCount ≠ 0 do
        vertex ← maxSaturation(graph);
        connectedColors ← vertex.connectedColors;
        minimalColor = graph.colorCount;
        for color ∈ 0, ..., colorCount − 1 do
            if color ∉ connectedColors then
                minimalColor = color;
                break;
            end
        end
        if minimalColor == graph.colorCount then
            graph.colorCount += 1;
        end
        vertex.color ← minimalColor;
        uncoloredCount -= 1;
    end
```

```
Function maxSaturation(graph)
    maxSaturatedVertex ← null;
    maxSaturation ← -1;
    for vertex ∈ graph.vertices do
        if vertex is uncolored AND vertex.saturation(graph.colorCount) > maxSaturation then
            maxSaturatedVertex ← vertex;
        end
    end
    return maxSaturatedVertex;
```

1.3 Het stochastisch lokale zoekalgoritme

Het laatste algoritme dat op de graaf wordt toegepast is een stochastisch lokale zoekalgoritme, dat dient om de gevonden kleuring door gebruik te maken van een constructiealgoritme te verbeteren. Het zoekt dus naar een kleuring met minder kleuren. Het gebruikte algoritme in deze implementatie is het tabu-1-exchange algoritme. Dit algoritme probeert een betere kleuring van de graaf te vinden door steeds het kleur van 1 knoop om te wisselen, meer specifiek een knoop die op dat moment verbonden is met een knoop met hetzelfde kleur. Echter moet wel nog worden besloten wat een betere kleuring is. Een kleuring wordt geëvalueerd door het aantal bogen die eindigen in éénzelfde kleur op te tellen. Een kleuring is beter wanneer dit aantal lager is. Het DSATUR algoritme geeft een volledige en geldige kleuring van de graaf. Dus, om het tabu-1-exchange algoritme toe te passen moet de kleuring eerst ongeldig worden gemaakt, met de bedoeling het kleurenaantal te verlagen. Er wordt daarom 1 kleur van de graaf verwijderd, en elke knoop die deze kleur had krijgt een willekeurig nieuw kleur dat al gebruikt wordt. Daarna wordt het algoritme toegepast tot de kleuring wordt geëvalueerd met waarde 0, en er dus geen bogen zijn die eindigen in éénzelfde kleur. In deze implementatie slaat het programma dan deze kleuring op, waarna het dan opnieuw een kleur verwijdert om daarna het tabu-1-exchange op toe te passen. Het kan echter gebeuren dat we geen betere kleuring vinden na het wisselen van 1 kleur, waardoor het algoritme mogelijks in een oneindige loopt terechtkomt waar steeds 1 knoop wisselt tussen twee kleuren. Om dit te vermijden houdt het algoritme bij wanneer een bepaalde knoop van kleur wisselde, en welk kleur deze had, en verbiedt het om die knoop terug te verwisselen naar die kleur voor een bepaald aantal iteraties, tenzij het de kleuring zo zou verbeteren dat die beter is dan alle vorige kleuringen. Op die manier blijft het algoritme steeds verder zoeken naar een betere kleuring. Het aantal iteraties dat een knoop niet mag worden verwisseld wordt bij onze implementatie bijgehouden in de knoop. Dit aantal, genoteerd met tt, wordt berekend aan de hand van volgende formule:

$tt = \texttt{random}(A) + \delta \times |V^C|$

waarbij A en δ parameters en $|V^C|$ het aantal knopen die verbonden zijn met een knoop met dezelfde kleur. $\texttt{random}(A)$ is daarbij een uniform willekeurig gekozen geheel getal van $0, ..., A$.

Function improveColoring(graph, A, d)

```
tabooClock ← 0 infeasibleEdgeCount ← 0;
conflictCount ← 0;
colorSwitchCount ← 0;
graph.saveCurrentColoring();
timeNotDepleted ← true;
while timeNotDepleted do
    graph.saveCurrentColoring();
    graph.removeColor();
    infeasibleEdgeCount ← graph.infeasibleEdges();
    conflicCount ← graph.conflictCount();
    startTime ← now();
    while infeasibleEdgeCount ≠ 0 do
        graph.switchMostPromisingVertexColor(tabooClock, A, d, conflictCount)
        if now() - startTime > 10000 then
            timeNotDepleted ← false;
            break;
        end
    end
end
graph.loadSavedColoring();
```

2 Datastructuur

In deze sectie worden de gebruikte datastructuren uitgelegd. Er zijn twee klasses aangemaakt voor dit project, nl. een klasse Vertex en een klasse Graph. Vertex stelt een knoop in de graaf voor, en houdt volgende elementen bij:

- id: een natuurlijk getal dat uniek is voor elke knoop.
- adjacentVertices: een lijst met objecten van de klasse Vertex, die de verbonden knopen of buren voorstellen van de knoop. Dit houdt dus in dat we een adjacency list gebruiken als datastructuur.
- color: een natuurlijk getal dat het kleur van de knoop voorstelt. Dit is -1 wanneer deze nog geen kleur heeft.
- conflictCount: het aantal knopen waarmee de knoop in conflict is, of dus het aantal buren met dezelfde kleur.
- tabooTimer: een getal dat de iteratie voorstelt tot wanneer de knoop nog niet mag worden verwisseld van kleur tijdens het tabu-1-exchange algoritme.

De klasse Graph is dan iets simpeler op zich, deze houdt enkel de lijst van knopen bij in vertices, een lijst van geldige knopen na reductie in validVertices, en het aantal kleuren dat wordt gebruikt in de graaf in colorCount.

3 Experimenten

Na het uitvoeren van het programma op verschillende grafen, merk ik dat het stochastisch lokaal zoekalgoritme weinig effect heeft op het aantal kleuren van de graaf vergeleken met het kleurenaantal na constructie. Het kleurenaantal lijkt praktisch altijd behouden te blijven, en dit voor elke graaf in de gegeven DIMACS grafen verzameling. Er kan dus enkel van uitgegaan worden dat er een fout in de code zit die maar niet gevonden wordt. Aangezien er een fout zit in het programma, kan er dus ook geen conclusies worden getrokken over de effectiviteit van het tabu-1-exchange algoritme.

Besluit

Er moet dus nog meer onderzoek worden gedaan naar de effectiviteit van het tabu-1-exchange algoritme, en naar het effect dat de parameters A en δ hebben op de efficiëntie van het algoritme.