Universiteit Twente

Module 7 - 201400433

Discrete Structuren & Efficiënte Algoritmes

Preprocessing methoden

Het efficiënt maken van het Graaf Isormorfisme probleem

Projectgroep 1

Auteurs:
Joeri Kock
Remco Brunsveld
Frank Bruggink
Daniël Schut

Projectbegeleider:
Prof. Dr. J.C. van de Pol

April 10, 2015

Inhoudsopgave

1 Inleiding **2**

2 Het Graaf Isomorfisme (GI) probleem **3**
2.1 Relevantie van het GI probleem 4
2.2 Speciale gevallen 4

3 Het algoritme voor Color Refinement **5**

4 Preprocessing **7**
4.1 Methoden 7
4.1.1 Grootte 7
4.1.2 Connectiviteit 8
4.1.3 False twins 9
4.1.4 Modules 9
4.2 Validatie preprocessing methoden 9

5 Testen **11**

6 Conclusie **12**

1 Inleiding

Dit paper zal gaan over het Graaf Isomorfisme probleem, en hoe dit efficiënt gemaakt kan worden door middel van preprocessing. Het Graaf Isomorfisme probleem is een bekend probleem in de wiskunde en de informatica, wat het belangrijk maakt om dit op een zo snel mogelijke manier op te lossen. Als eerste wordt hierover (en over het algoritme wat wij hiervoor gebruiken) het een en ander uitgelegd. We zullen ook verschillende manieren van preprocessing bespreken, en toelichten waarom deze methoden geldig zijn binnen dit probleem. Daarna zullen we door middel van een experiment testen of deze methoden werken en wat het effect hiervan is op de snelheid en efficiëntie van ons algoritme. We sluiten dit document af met de conclusie.

We hebben in dit paper als hoofdvraag: "In hoeverre heeft het gebruik van preprocessing invloed op de tijdsduur van het vinden van isomorfe grafen met het Color Refinement-algoritme?". We hopen natuurlijk dat het gebruik van preprocessing van positieve invloed is op de tijdsduur, d.w.z. dat het geheel minder lang duurt.

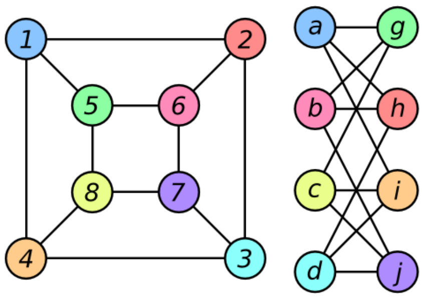
2 Het Graaf Isomorfisme (GI) probleem

Een graaf bestaat uit een verzameling punten, vertices genoemd, waarvan sommige verbonden zijn door lijnen, de edges. Afhankelijk van de toepassing kunnen de lijnen gericht zijn, dan worden ze ook wel pijlen genoemd, men spreekt dan van een gerichte graaf. Ook worden wel gewichten aan de lijnen toegekend door middel van getallen, deze stellen dan bijvoorbeeld de afstand tussen twee punten voor. Een graaf met gewichten noemt men een gewogen graaf. Wanneer we in dit paper een graaf noemen, hebben we het over een ongewogen ongerichte enkelzijdige graaf.

Een isomorfisme is een bijectieve afbeelding $f : V(G) \to V(H)$ zodanig dat $\forall u, v \in V(G)$ er geldt dat $uv \in E(G) \Leftrightarrow f(u)f(v) \in E(H)$.

Het Graaf Isomorfisme probleem is het probleem dat gaat over het vaststellen of twee grafen isomorf zijn, dat wil zeggen structureel gelijk aan elkaar. Als twee wiskundige objecten isomorf zijn, dan is elke eigenschap, waarvan de structuur bewaard blijft door een isomorfisme en die geldt voor een van de twee wiskundige objecten, ook geldig voor het andere wiskundige object.

Hoewel het op het eerste gezicht niet zo lijkt te zijn, geldt er een isomorfisme tussen de twee grafen hiernaast. Voor elke vertex van de eerste graaf geldt dat de buren daarvan exact hetzelfde zijn als de buren van diezelfde vertex in de tweede graaf.



Figuur 1: Twee isomorfe grafen

Een methode om isomorfisme vast te kunnen stellen tussen deze twee grafen is color refinement. Dit algoritme kent kleuren toe aan de verschillende vertices, die een indicatie geeft welke buren deze vertex heeft. De groene vertex bijvoorbeeld heeft in de eerste graaf drie buren: de lichtblauwe, de roze en de gele. Dit geldt ook voor de tweede graaf. Als dit voor elke vertex in de twee grafen geldt èn elke vertex heeft een unieke kleur, zijn de twee grafen isomorf. Dit kan als volgt gecontroleerd worden: wanneer de lijst met alle kleuren (zonder duplicaten) van de eerste graaf gelijk is aan die van de tweede graaf, kan er gezegd worden dat ze isomorf zijn ten opzichte van elkaar. In de afbeelding hierboven is dat ook het geval. Dit is een goede manier om isomorfisme te controleren, omdat het op het eerste gezicht bij deze afbeelding lijkt alsof de twee grafen niet isomorf zijn. Ook is deze methode nuttig bij grotere grafen met veel vertices.

2.1 Relevantie van het GI probleem

Nu duidelijk gemaakt is wat het GI probleem inhoudt en hoe het gecontroleerd kan worden, blijft de vraag hoe relevant dit probleem is voor de Discrete Wiskunde. Waarom willen we weten of twee grafen (of zelfs wiskundige problemen in het algemeen) isomorf zijn ten opzichte van elkaar?

Isomorfisme heeft al voor het oplossen van vele wiskundige problemen gezorgd. Dat kan als volgt gedaan worden: een nog op te lossen wiskundig probleem X kan herleid worden tot een eenvoudiger probleem Y, wat makkelijker te begrijpen is. Neem een reeds opgelost probleem Z. Als aangetoond kan worden dat Y en Z isomorf zijn ten opzichte van elkaar, kan de conclusie getrokken worden dat op dat moment probleem X ook opgelost is (omdat probleem X, Y en Z structureel hetzelfde zijn). In de Discrete Wiskunde zijn er al veel problemen met deze methode opgelost, wat het controleren op isomorfisme erg relevant maakt. Naast de wiskunde is het ook nuttig voor de informatica, omdat daar veel structuren als grafen weergegeven worden (denk aan netwerksystemen).

Een goed voorbeeld hiervan in de informatica zijn vingerafdrukscanners. Wanneer een computer een vingerafdruk scant, wordt deze versimpeld naar enkele lijnen. Elke lijn representeert een vertex in de graaf, en elke edge representeert een relatie in de omgeving van deze lijn(en). Met deze techniek wordt een vingerafdruk versimpeld naar een graaf met vertices en edges. Als men wil controleren of twee vingerafdrukken gelijk zijn aan elkaar, moet men een isomorfisme bewijzen tussen de twee grafen die bij die vingerafdrukken horen.

2.2 Speciale gevallen

Er is een aantal speciale gevallen waarvan bewezen is dat het in polynomiale tijd opgelost kan worden. Het eerste geval is een graaf die een boomstructuur heeft, dat wil zeggen: elk punt is met een ander punt verbonden, zonder cycli. De tweede is een planaire graaf, dat is een graaf die zo in het platte vlak getekend kan worden dat geen van de randen een andere rand kruist. Ook als je een restrictie legt op het aantal keren dat de randen elkaar minimaal kruisen, of als je een restrictie legt op de maximale graad van alle hoekpunten is het in polynomiale tijd op te lossen.

3 Het algoritme voor Color Refinement

Hier zullen we een korte uitleg geven over de implementatie van ons color refinement-algoritme.

Voordat we color refinement toepassen, geven we eerst elke vertex van de graaf een kleur die gelijk staat aan zijn degree. Het kleuren gebeurt niet in de graaf zelf, maar wordt apart bijgehouden door 2 arrays. De eerste array houdt voor elke kleur bij welke vertices die kleur hebben. De tweede array houdt voor elke vertex bij welke kleur hij heeft. Op deze manier hebben we de graaf zelf (bijna) niet meer nodig en kunnen we alle bewerkingen op de twee arrays uitvoeren, wat het proces sneller en overzichtelijker maakt.
Na de initiële kleuring voeren we color refinement uit over de lijsten. Als blijkt dat er nu geen duplicate kleuren meer zijn binnen elke graaf (d.w.z. binnen een graaf bestaan er geen twee vertices met dezelfde kleur) is het klaar en kunnen we controleren welke graven isomorf zijn. Zijn er wel duplicate kleuren, dan moet 'individual refiment' toegepast worden. De pseudo-code voor het eerste deel staat hieronder:

```
Data: Een lijst met initiele kleuring van de vertices en een lijst met
      vertices per kleur
Result: Een lijst met stabiele kleuring van de vertices en een lijst met
        vertices per kleur
for Alle vertices v do
 |  for Alle vertices k die de kleur van v hebben do
 |   |  nieuweKleuren = []
 |   |  if De kleuren van de neighbors van v ongelijk zijn aan de neigbors
 |   |  van k then
 |   |   |  for Alle elementen q in nieuweKleuren do
 |   |   |   |  if De neighbors van k gelijk zijn aan die van q then
 |   |   |   |   |  Kleur van k = q
 |   |   |   |  else
 |   |   |   |   |  Maak een nieuwe kleur voor k aan
 |   |   |   |   |  Voeg de kleur toe aan de lijst nieuweKleuren
 |   |   |   |  end
 |   |   |  end
 |   |  end
 |  end
end
if De nieuwe lijsten ongelijk zijn aan de oude lijsten then
 |  Recursie van het algoritme met nieuwe lijsten als data
end
return De twee aangepaste lijsten
```

Individual refinement wordt alleen toegepast op de graven met duplicate kleuren. Het algoritme werkt als volgt: hij zoekt eerst een kleur op die bezet wordt door meerdere vertices. Vervolgens neemt hij één van deze vertices, en vervangt zijn kleur. Bij beide grafen wordt hiervoor een andere (nieuwe) kleur gekozen.Hierna wordt color refinement opnieuw toegepast. Als er nog steeds duplicate kleuren bestaan wordt color refinement nogmaals toegepast. Als er een isomorfisme aangetoond kan worden, wordt individual refinement over een ander koppel graven toegepast. Is er geen isomorfisme dan wordt de herkleuring van graaf 1 en graaf 2 teruggedraaid en wordt er een andere duplicate kleur van graaf 2 vervangen, waarna het algoritme zich weer herhaalt.

```
Data: Een lijst met stabiele kleuring van de vertices en een lijst met
      vertices per kleur
Result: Een lijst met unieke kleuring van de vertices per graaf en een
        lijst met vertices per kleur
dupVertices = dictionary met alle vertices van een dubbele kleur per graaf
for Alle graven g in dupVertices do
    for Alle vertices v van g do
        Verander de eerste kleur van de eerste graaf in dupVertices in een
        nieuwe kleur
        Verander de kleur van v in een nieuwe kleur
        Pas colorrefinement toe op de twee lijsten
        if De kleuren van de twee aangepaste grafen zijn gelijk then
            if Er zijn geen grafen zonder unieke kleuring meer then
                return de nieuwe lijsten
            else
                Pas individual refinement opnieuw toe
                if Er is geen isomorfisme gevonden then
                    Zet de staat van de lijsten terug naar het origineel
                end
            end
        end
    end
end
return De lijsten met unieke kleuring
```

4 Preprocessing

Om het algoritme een handje te helpen bij het efficiënt oplossen van het GI probleem, kunnen er voor het color refinement-algoritme al stappen ondernomen worden, zodat het algoritme zelf minder tijd nodig heeft en dus het algoritme sneller en efficiënter wordt. Voor het algoritme kunnen er namelijk grafen geëlimineerd worden uit de lijst die niet in aanmerking komen om isomorf te zijn met een andere graaf uit de lijst. Als je van tevoren deze grafen uit de lijst haalt, hoeft het color refinement-algoritme minder grafen te behandelen, waardoor het efficiënter wordt.
Voor het testen gebruik je een lijst van grafen, bijvoorbeeld een lijst met 4 grafen. Als er één graaf is in de lijst die een van de eigenschappen van het preprocessing niet heeft en de andere drie wel, kan deze graaf uit de lijst gehaald worden. Het is dan namelijk niet meer mogelijk dat deze graaf isomorf is met één van de andere grafen. Maar als twee grafen de eigenschappen wel hebben en twee niet, zetten we deze paren van grafen in twee aparte lijsten, waar vervolgens het color refinement-algoritme op uitgevoerd kan worden.

4.1 Methoden

We controleren de grafen van tevoren op de volgende eigenschappen:

4.1.1 Grootte

De grootte van de graaf, d.w.z. het aantal vertices en het aantal edges. Als er in de lijst n graaf is met een ander aantal vertices of edges dan de andere grafen, kan deze graaf uit de lijst gehaald worden. Hij kan dan niet meer isomorf zijn met een andere graaf. De pseudo-code voor het testen op grootte staat hieronder:

Data: Een lijst van grafen
Result: Een lijst van grafen die mogelijk isomorf zijn
resultaat = lege lijst
for *Elke graaf i in de lijst* **do**
    for *Elke andere graaf j in de lijst* **do**
        if *Het aantal vertices van graaf i en j zijn gelijk* **then**
            if *Het aantal edges van graaf i en j zijn gelijk* **then**
                Voeg i en j toe aan het resultaat
            end
        end
    end
end
return resultaat

4.1.2 Connectiviteit

Als een graaf als enige in de lijst niet connected is (en de andere grafen dus wel), kan deze ook geëlimineerd worden. Met een combinatie van deze eerste twee eigenschappen wordt ook gelijk gecontroleerd op het feit of een graaf een boom is of niet. Als twee connected grafen een gelijk aantal vertices en edges hebben, zijn ze namelijk beide een boom of beide niet. Hier hoeft dus niet meer op gecontroleerd te worden. De pseudo-code voor het testen op connectiviteit staat hieronder:

```
Data: Een lijst van grafen
Result: Een lijst van grafen die mogelijk isomorf zijn
resultaat = lege lijst
for Elke graaf i in de lijst do
    for Elke andere graaf j in de lijst do
        if Grafen i en j zijn beide connected then
            Voeg i en j toe aan het resultaat
        end
    end
end
return resultaat
```

In de If-statement staat dat we controleren of beide grafen i en j connected zijn. We hebben een algoritme geschreven (dat je op een graaf kunt aanroepen) dat teruggeeft of een graaf connected is of niet. De pseudo-code hiervan staat hieronder beschreven:

```
Data: De graaf G
Result: True als hij connected is, anders False
x = een willekeurige vertex
L = een lijst van vertices bereikbaar vanaf x
K = een lijst van vertices die nog ontdekt moeten worden
Aan het begin van het algoritme geldt L = K = x
while K is niet leeg do
    Vind een vertex y in K en haal deze eruit
    for elke edge (y,z) do
        if z zit niet in L then
            Voeg z toe aan L en K
        end
    end
end
if L heeft minder elementen dan het aantal vertices van G then
    return False
else
    return True
end
```

Nog een extra mogelijkheid bij het controleren op connectiviteit is het splitsen van grafen. Als een graaf niet connected is, kan deze opgesplitst worden in een connected deel en de rest van de graaf. Het eerste deel is nu wel connected, en kan om die reden dus vergeleken worden met andere connected grafen in de lijst (mits het aantal vertices en edges gelijk zijn aan die van de andere grafen). Helaas is het niet gelukt om deze functionaliteit te implementeren in onze code.

4.1.3 False twins

Een andere mogelijkheid is het controleren of een graaf 'False Twins' heeft. False twins zijn twee vertices die exact dezelfde buren hebben, maar niet met elkaar verbonden zijn. Als het aantal false twins in graaf G ongelijk is aan het aantal false twins in graaf H, kunnen G en H niet meer isomorf zijn. Helaas is het hier ook niet gelukt om deze methode binnen de tijd te implementeren, maar het bewijs voor de validatie is wel in de volgende sectie te vinden.

4.1.4 Modules

Een module is een groep vertices die buiten de module dezelfde buren hebben. Als een vertex X (buiten de module) verbonden is met een vertex in een bepaalde module, zijn alle vertices in die module verbonden met X. Natuurlijk is elke vertex zelf ook een module, net als de gehele graaf. Als graaf G een ongelijk aantal modules heeft aan graaf H, kunnen G en H niet meer isomorf zijn. Dit hebben we niet geïmplementeerd in onze code en ook niet bewezen in de volgende sectie (vanwege de tijd).

4.2 Validatie preprocessing methoden

Een isomorfisme is een bijectieve afbeelding $f : V(G) \to V(H)$ zodanig dat $\forall u, v \in V(G)$ er geldt dat $uv \in E(G) \Leftrightarrow f(u)f(v) \in E(H)$.

We hebben 3 preprocessing stappen gemaakt.

1. Als $|V(G)| \neq |V(H)|$ dan kan er geen isomorfisme bestaan.
2. Als $|E(G)| \neq |E(H)|$ dan kan er geen isomorfisme bestaan.
3. Als een graaf wel 'connected' is en de andere niet, dan kan er geen isomorfisme bestaan.
4. Als het aantal 'false twins' van G een H niet gelijk is, kan er geen isomorfisme bestaan.

Nu volgt het bewijs dat twee grafen niet isomorf zijn als aan een van de volgende 3 voorwaarden wordt voldaan:

1. Als $|V(G)| \neq |V(H)|$ dan kan je geen bijectie definiëren, want een bijectie moet one-to-one zijn, en daarvoor moeten beide sets evenveel elementen bevatten.

2. Als $|E(G)| \neq |E(H)|$ dan is er een tegenspraak met de eis dat $uv \in E(G) \Leftrightarrow f(u)f(v) \in E(H)$.

3. Neem aan dat H connected is en G niet. Zonder verlies van algemeenheid kunnen we aannemen dat G bestaat uit twee delen die beide connected zijn, noem ze G_1 en G_2.

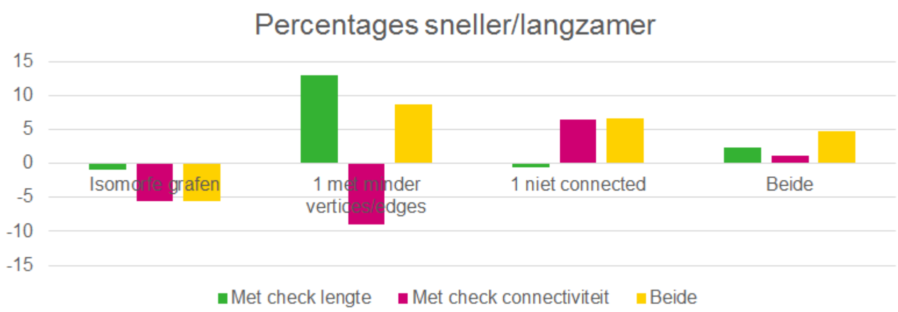
4. Twee vertices $u, v \in V(G)$ zijn false twins als $N(u) = N(v)$. Als graaf G en graaf H niet hetzelfde aantal false twins hebben, dan kan er geen isomorfisme bestaan tussen G en H, omdat een isomorfisme de edges behoudt, en dus ook de neighborhood.

Neem nu aan dat er een isomorfisme tussen G en H bestaat. Deel H op in twee delen, H_1 en H_2, zodanig dat $\forall g_1 \in V(G_1)$ geldt dat $f(g_1) \in V(H_1)$ en $\forall g_2 \in V(G_2)$ geldt dat $f(g_2) \in V(H_2)$. Dan $\exists g_1 \in G_1$ en $g_2 \in G_2$ zodanig dat $g_1 g_2 \in E(G)$, terwijl $f(g_1)f(g_2) \notin E(H)$. Dit levert een tegenspraak op.
Dus de grafen moeten beide wel of beide niet connected zijn.

5 Testen

We hebben onze code uitvoerig getest op verschillende delen. Als eerste hebben we testklassen gemaakt in Python die uitgevoerd kunnen worden en (met behulp van wat randgevallen) testen of onze preprocessing-methoden naar behoren werken. Dit zijn zogenaamde 'unittests' die als volgt werken: we testen eerst of onze functie `isConnected` werkt. Dit doen we door deze functie uit te voeren op vier grafen, waarvan er één niet connected is. Hier voeren we dus een drietal `assertTrue`- en één `assertFalse`-functie(s) op uit.
Bovendien kijken we of het aanpassen van de lijsten werkt. Immers, als er één graaf als enige niet connected is of een ander aantal vertices of edges heeft, moet deze uit de lijst gehaald worden. We voeren hiervoor de betreffende methoden uit, en we controleren naderhand of de aangepaste lijst overeenkomt met de lijst die we voorspeld hadden.

We hebben naast de werking ook het effect van onze algoritmes getest. Met behulp van verschillende lijsten van grafen (waaronder we er een aantal zelf geschreven hebben) hebben we experimenten uitgevoerd waar we hebben gekeken of de preprocessing-methoden ook daadwerkelijk effect hadden op de tijd die het algoritme erover doet. Het resultaat van dit experiment is te zien in de grafiek hieronder.



De waarden in de grafiek staan voor hoeveel sneller of langzamer het algoritme er procentueel over doet t.o.v. zonder de preprocessing. Wat erg duidelijk te zien is, is dat als we één grafiek een ander aantal vertices geven, we een significante verbetering zien wanneer we hierop controleren d.m.v. preprocessing. Dit is bij de connectiviteit ook het geval. Wanneer de grafen al isomorf zijn, heeft het natuurlijk enkel een negatief effect. Je voert namelijk extra controles uit op eigenschappen waarvan je weet dat dit niet hoeft, aangezien de grafen al isomorf zijn.

6 Conclusie

Op basis van de gemeten testresultaten kunnen we een conclusie trekken over de gebruikte methoden van preprocessing. Dat is dat ze wel degelijk verschil maken in de uiteindelijke tijd, mits het ook noodzakelijk is deze controles uit te voeren. Als er in de lijst een graaf is die als enige connected is, wordt deze uit de lijst gehaald. Dit scheelt het Color Refinement-algoritme erg veel tijd. Heb je echter een lijst van grafen die allemaal evenveel vertices en edges hebben èn allemaal connected of niet connected zijn, kost het algoritme een (klein) beetje extra tijd, omdat je controles uitvoert die niet nodig zijn.

Referenties

[1] Gaurav Rattan Oleg Verbitsky V. Arvind, Johannes Kbler, *Graph isomorphism, color refinement, and compactness*, 2015.

[2] N. M.; Tyshkevich R. I. Zemlyachenko, V. N.; Korneenko, *Graph isomorphism problem*, 1985.