KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Oplossingsmethodes voor het Maximum Happy Vertices probleem

Florian Peeters

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdoptie Artificiële intelligentie

Promotoren:

Prof. dr. Patrick De Causmaecker
Dr. Pieter Leyman

Assessoren:

Prof. dr. Marc Denecker
Dr. Pieter Smet

Begeleider:

Dr. Pieter Leyman

Academiejaar 2019 – 2020

Voorwoord

Dit voorwoord wil ik graag gebruiken om de mensen te bedanken die me ondersteund hebben bij het afwerken van deze thesis. Ik wil prof. dr. Patrick De Causmaecker en dr. Pieter Leyman bedanken voor de wekelijkse afspraken, en de uitgebreide begeleiding die ze me gegeven hebben bij het uitwerken van deze thesis. Beide hebben me enorm geholpen door nuttige suggesties te geven en me in de juiste richting te sturen door verschillende nuttige papers aan te wijzen. Pieter Leyman wil ik ook nog extra bedanken voor het uitgebreid nalezen van de thesistekst en om hierbij verschillende opmerkingen en verbeteringen voor te stellen.

Bij het afwerken van deze thesis werd met de (co)promotoren afgesproken om tijdens de zomer van 2020 over de behaalde resultaten ook een paper te schrijven en deze voor te dragen voor publicatie. Ik wil hen bij voorbaat bedanken voor deze voor mij interessante opportuniteit.

Prof. dr. Marc Denecker en dr. Pieter Smet wil ik bij voorbaat bedanken voor het optreden als assessor voor deze thesis.

Tot slot wil ik mijn ouders bedanken voor het luisteren naar mijn uitleg over verschillende voor hen niet zo voor de hand liggende onderwerpen, en voor het geven van andere inzichten in de problemen die ik tegengekomen ben bij het afwerken van deze thesis.

Florian Peeters

Inhoudsopgave

Voorwoord **i**

Samenvatting **v**

Lijst van figuren **vi**

Lijst van tabellen **vii**

Lijst van afkortingen en symbolen **ix**

1 Inleiding **1**
1.1 Optimalisatie 1
1.2 Doel van de thesis 2
1.3 Structuur van de tekst 3

2 Probleemdefinitie **5**
2.1 Definitie 5
2.2 Voorbeeld 6

3 Literatuuroverzicht **7**
3.1 Complexiteit van MHV 7
3.2 Algemene oplossingsmethoden 8
3.3 Praktische applicaties 8

4 Reductie van het probleem **9**
4.1 Reductiemethode van Thiruvady et al. 9
4.2 Referentie-gebaseerde reductie 11
4.3 Vergelijking reductie-algoritmen 16
4.4 Besluit 17

5 Exacte solver **19**
5.1 IP model 19
5.2 Alternatief IP model 20
5.3 Besluit 22

6 Constructieve algoritmen **23**
6.1 Constructieve algoritmen van Li en Zhang 23
6.2 Exact algoritme voor 2-reguliere grafen 27

7 Metaheuristisch algoritme **31**
7.1 Simulated Annealing 31
7.2 Componenten 32

7.3 Besluit 39

8 Opstelling van de experimenten **41**
8.1 Aanmaken van MHV-instanties 41
8.2 Beschouwde deel van de instantie-ruimte 43
8.3 Tuning van algoritme-parameters 45

9 Resultaten **49**
9.1 Vergelijking reductie-methodes 50
9.2 Resultaten exacte solver 54
9.3 Analyse gedrag selector in Growth-MHV 57
9.4 Prestaties metaheuristiek 59
9.5 Conclusies 63

10 Besluit **65**

Bijlagen **67**

A Lineaire kansverdeling **69**
A.1 Eenvoudige lineaire kansverdeling 69
A.2 Uitbreiding voor grotere bias 71

Bibliografie **75**

Samenvatting

In deze thesis worden verschillende oplossingsmethodes bekeken voor het Maximum Happy Vertices probleem. In dit combinatorische optimalisatieprobleem moet er voor elke knoop in een ongerichte graaf een kleur gevonden worden die zoveel mogelijk knopen Happy maakt. Een knoop is pas Happy als alle verbonden knopen dezelfde kleur hebben. In deze thesis worden twee reeds bestaande constructieve algoritmen geanalyseerd, en wordt een nieuw metaheuristisch algoritme voorgesteld dat gebruik maakt van het Simulated Annealing framework. Er wordt ook een nieuwe reductietechniek voorgesteld die een originele probleeminstantie zoveel mogelijk verkleint door knopen waarvoor de kleur triviaal kan worden bepaald te verwijderen uit de graaf. Verder worden twee verschillende Integer Programming modellen geanalyseerd. De prestaties van al deze technieken worden geëvalueerd aan de hand van twee reeds bestaande instantiegeneratoren en een nieuw ontwikkelde generator. De resultaten van de uitgevoerde experimenten geven aan dat de prestaties van de combinatie van articulatiereductie met zowel het Greedy algoritme als het nieuwe metaheuristische algoritme zeer dicht bij de prestaties van de meest recente oplossingsmethode uit de literatuur liggen.

Lijst van figuren

2.1 Voorbeeld van een optimale kleuring voor een graaf. Knopen met een '*' zijn voorgekleurd. 6

4.1 Voorbeeld van de reductiemethode van Thiruvady et al. Figuur overgenomen uit [25](Fig. 3) 10
4.2 Voorbeeld van een reductie. 13
4.3 Voorbeeld van articulatie-knopen in een graaf. Gekleurde knopen zijn articulatie-knopen. 14
4.4 Voorbeeld van knopen die kunnen uitgesloten worden met referentie-gebaseerde reductie. Figuur bewerkt overgenomen uit [25] (Fig. 3). 16
4.5 Voorbeeld van het geval dat niet wordt gereduceerd door de articulatie-methode. Knopen a en g zijn voorgekleurd met dezelfde kleur. 17

6.1 Voorbeeld van een 2-reguliere graaf. Vierkante knopen zijn voorgekleurd. 27
6.2 Een groep van knopen in een 2-reguliere graaf, begrensd door 2 voorgekleurde knopen. 28

7.1 Voorbeeld van een gekleurde groep in een graaf die opgesplitst kan worden. 37

8.1 Grafen met $n = 20$ knopen en met een gemiddelde graad van $d = 2$ gegenereerd met de lineaire verdeling generator met verschillende waarden van α. 44

9.1 Gemiddeld aantal knopen extra gereduceerd door beide nieuwe reductiemethodes in vergelijking met de methode van Thiruvady et al. in functie van de gemiddelde graad. 52
9.2 Aantal knopen gereduceerd door de articulatiemethode in functie van de gemiddelde graad van de graaf. 53
9.3 Gemiddelde uitvoeringstijd van beide IP-modellen in functie van de gemiddelde graad van de graaf. 56
9.4 Gemiddeld aantal Happy knopen relatief ten opzichte van het maximum aantal Happy knopen voor verschillende selectiemethodes. 58
9.5 Relatieve gemiddelde prestaties van de vier algoritmen in functie van de gemiddelde graad van de graaf. 60

9.6 Relatieve gemiddelde prestaties van Greedy, Simulated Annealing en Tabu-search in functie van de gemiddelde graad van de graaf. 61
9.7 Uitvoeringstijd en aantal kleurveranderingen in beide algoritmen in functie van de gemiddelde graad van de instantie. 62

A.1 **Lineair**(α) voor verschillende waarden van α. 71
A.2 **Lin**(α) voor verschillende waarden van $\alpha > 0$. Merk op dat dit slechts een overzicht schetst van het verloop van de verdeling voor verschillende waarden, de curves voor $\alpha = 1.25$ en $\alpha = 1.75$ zijn geschaald zodat hun maximum samenvalt met die voor $\alpha = 1$, wat geen geldige verdeling meer is aangezien de oppervlakte onder de curve niet gelijk is aan 1. 72
A.3 Eenvoudige afbeelding van de uitgebreide lineaire verdeling voor een waarde van $\alpha > 1$, met $beta = \alpha - 1$ 73

Lijst van tabellen

5.1 Overzicht van de grootte van beide IP-modellen voorgesteld in dit hoofdstuk. 21

6.1 Beslissingstabel voor de verschillende mogelijke gevallen voor een groep. 28

7.1 Alle parameters van het metaheuristisch algorimte. 40

8.1 Overzicht van de gebruikte parameters voor de instantie-generatoren. . . 45
8.2 Beschouwde waarden voor alle parameters van het metaheuristisch algorimte. 46
8.3 Gebruikte waarden voor alle parameters van het metaheuristisch algoritme. 47

9.1 Gemiddeld aantal knopen meer gereduceerd door beide nieuwe reductiemethodes in vergelijking met de methode van Thiruvady et al. in functie van het type graaf en het aantal kleuren. 51
9.2 Gemiddeld aantal knopen extra gereduceerd in grafen aangemaakt met de lineaire verdeling generator door beide nieuwe reductiemethodes in vergelijking met de methode van Thiruvady et al. in functie van α. . . . 51
9.3 Gemiddelde uitvoeringstijd van beide IP-modellen voor verschillende types grafen. 55
9.4 Gemiddelde uitvoeringstijd van beide IP-modellen voor verschillende voorkleuringen. 55

Lijst van afkortingen en symbolen

Afkortingen

MHV	Maximum Happy Vertices
DFS	Depht-first search
BFS	Breadth-first search
SA	Simulated Annealing
IP	Integer Programming
pdf	Probability Density Function, kansdichtheid
cdf	Cumulative Density Function, cumulatieve kansdichtheid

Symbolen

$G(V, E)$	Een graaf, met V de set van knopen en E de set van bogen
$\Gamma(v)$	Alle knopen verbonden met v
$\Gamma^n(v)$	Alle knopen verbonden met v door een pad met lengte $\leq n$
$\#A$	Kardinaliteit van set A, het aantal elementen in A
$A \backslash B$	Verschil van twee sets, $A \backslash B = \{a \mid a \in A \wedge a \notin B\}$

Hoofdstuk 1

Inleiding

1.1 Optimalisatie

Het vinden van de beste oplossing voor een moeilijk probleem is een veel voorkomend probleem. Dergelijke optimalisatieproblemen bestaan uit een doelfunctie waarvan de maximale (of minimale) waarde gevonden moet worden, en uit verschillende beslissingsvariabelen. Deze beslissingsvariabelen stellen elk een keuze voor die gemaakt moet worden om tot een oplossing te komen. Een optimalisatieprobleem wordt opgelost door te zoeken naar de waarden voor de beslissingsvariabelen die de optimale waarde van de doelfunctie bekomen. Bij verschillende optimalisatieproblemen komen hier nog beperkingen bij, die aangeven welke combinaties van waarden voor de beslissingsvariabelen mogelijk of verboden zijn.

Combinatorische optimalisatieproblemen zijn een speciaal type binnen alle mogelijke optimalisatieproblemen. Bij dit soort problemen is het domein van de beslissingsvariablen eindig, en komt het zoeken naar het optimum neer op het vinden van een optimaal object. Naar dit object kan aan de hand van verschillende technieken gezocht worden. De eenvoudigste methode gaat alle mogelijke objecten af, en zal op die manier altijd het optimale object vinden. Hoewel het theoretisch mogelijk is om dit te doen, is dit in de praktijk vaak onmogelijk door het grote aantal mogelijke oplossingen. Dit zorgt er voor dat andere technieken nodig zijn om op een slimmere manier te zoeken naar een optimaal object. Het Travelling Salesman Probleem[11], het Knapsack Probleem[15] en het Graph Colouring Probleem[27] zijn enkele van de meest bekende combinatorische optimalisatieproblemen.

Er zijn optimalisatieproblemen waarvoor algoritmes bekend zijn die zeer snel en efficiënt een optimale oplossing vinden, maar aangezien er veel problemen zijn die NP-compleet of NP-hard zijn, bestaat er vaak zo geen algoritme. Een veelgebruikt alternatief hierop is een algoritme gebruiken dat relatief snel een oplossing vindt die goed genoeg is. Dit zijn heuristische algoritmen, die aan de hand van slimme beslissingen een zo goed mogelijk optimum vinden in beperkte tijd. Eén van de meest bekende heuristieken is waarschijnlijk de heuristiek die gebruikt wordt in

routeplanners: de afstand in vogelvlucht tussen twee plaatsen is een relatief goede indicator voor de lengte van de kortste weg tussen deze plaatsen. Zoals dit voorbeeld ook aangeeft zijn heuristieken methodes om een schatting te maken van de optimaliteit van een oplossing, maar is er geen enkele garantie dat deze schatting ook correct is.

Heuristieken kunnen ook op een hoger niveau gebruikt worden in zoekalgoritmen, dit soort technieken worden metaheuristieken genoemd. In een metaheuristiek wordt een heuristiek gebruikt om het zoek-proces van het onderliggende zoekalgoritme te sturen. Een nadeel van gewone heuristieken is dat deze vaak voor een specifiek probleem ontworpen zijn, en dus niet bruikbaar zijn voor een ander probleem. Metaheuristieken daarentegen zijn technieken die bijna geen assumpties maken over het probleem zelf, en dus voor elk soort probleem kunnen gebruikt worden. Om deze reden wordt vaak gesproken over een metaheuristisch framework. Een metaheuristisch framework bestaat meestal uit verschillende componenten die met elkaar samenwerken om de oplossing te vinden. Om een metaheuristiek te gebruiken voor een specifiek probleem moet er enkel een probleem-specifieke versie van elk van deze componenten gemaakt worden, wat het ontwerpen van een oplossingsmethode eenvoudiger maakt. Er bestaan verschillende metaheuristieken, een aantal voorbeelden zijn Local Search, Genetische Algoritmen[6], Tabu-search[9] en Simulated Annealing[14].

1.2 Doel van de thesis

In deze thesis worden een combinatorisch optimalisatieprobleem en verschillende oplossingsmethoden voor dit probleem geanalyseerd. Het Maximum Happy Vertices probleem is een relatief nieuw probleem, en er zijn bijgevolg nog maar weinig oplossingsmethodes bekend. Dit probleem wordt verder in de tekst in detail voorgesteld.

Het doel van deze thesis bestaat uit twee delen. In het eerste deel wordt gekeken naar de algemene oplossingsmethodes die reeds voorgesteld zijn in de literatuur. Deze algoritmen worden hier opnieuw beschreven en geanalyseerd, om een duidelijk beeld te vormen van het probleem. Het tweede deel van het doel bestaat uit het ontwerpen van nieuwe methodes om het probleem aan te pakken. Hoofdzakelijk wordt gekeken naar een metaheuristische aanpak, omdat dit op het moment dat het werk aan deze thesis begon hier nog geen onderzoek naar gedaan was. In het begin van 2020 werd een paper gepubliceerd waarin wel een metaheuristisch algoritme wordt voorgesteld om het Maximum Happy Vertices probleem aan te pakken, maar hierin wordt een ander metaheuristisch framework gebruikt dan het framework dat voor het metaheuristisch algoritme dat in deze thesis wordt voorgesteld wordt gebruikt.

1.3 Structuur van de tekst

Beide delen van het doel komen door elkaar voor in de tekst, en er wordt altijd aangegeven of het gaat over een reeds beschreven component, of dat een nieuwe methode of component wordt voorgesteld.

De structuur van de tekst volgt het logische proces dat men doorloopt om een probleem op te lossen. Hoofdstuk 2 definieert het Maximum Happy Vertices (MHV) probleem, en geeft een voorbeeld van een probleeminstantie. In hoofdstuk 3 wordt een overzicht gegeven van het werk dat al eerder is uitgevoerd rond het MHV probleem.

Nadat het probleem en de historiek erachter bekend zijn, wordt de eerste stap van het oplossen van het probleem besproken in hoofdstuk 4. Hier worden methodes besproken die toelaten een probleeminstantie kleiner te maken door triviaal oplosbare stukken eruit te verwijderen. Een eerste mogelijke manier om het probleem op te lossen wordt besproken in hoofdstuk 5, waar Integer Programming modellen worden voorgesteld om de optimale oplossing te berekenen. In de meeste gevallen echter zijn deze exacte methodes te traag, en is het dus aangewezen (meta)heuristische algoritmen te gebruiken.

In hoofdstuk 6 worden reeds bestaande constructieve algoritmen beschreven en geanalyseerd. Hier wordt ook een exact algoritme beschreven voor 2-reguliere grafen. Hoofdstuk 7 beschrijft het nieuwe metaheuristisch algoritme ontworpen in deze thesis.

Hoofdstuk 8 beschrijft hoe alle experimenten worden opgezet. Dit houdt in hoe de probleem-instanties worden aangemaakt, welke instanties worden bekeken en hoe de parameters van het metaheuristische algoritme worden getuned. De resultaten van deze experimenten worden besproken in hoofdstuk 9. Tot slot worden algemene conclusies geformuleerd in hoofdstuk 10.

In appendix A wordt een lineaire kansverdeling voorgesteld. Deze wordt zowel gebruikt in hoofdstuk 6 als hoofdstuk 8, maar de volledige uitwerking van de kansverdeling is niet nodig om de tekst zelf te volgen.

Hoofdstuk 2

Probleemdefinitie

Het Maximum Happy Vertices (MHV) probleem is een labeling probleem gedefinieerd voor ongerichte grafen. Het bekendste graaf-labeling probleem is waarschijnlijk het Graph Colouring probleem. Hierbij moet elke knoop een label, of kleur, krijgen, zodat twee verbonden knopen nooit dezelfde kleur hebben. Het MHV probleem probeert eigenlijk het omgekeerde te bekomen, aangezien het probeert zoveel mogelijk knopen met dezelfde kleur te verbinden.

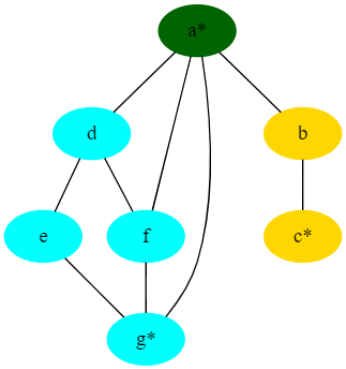
Het speciale concept in het MHV probleem is het concept van Happiness van een knoop. Een knoop is Happy als deze enkel verbonden is met knopen van dezelfde kleur. Het doel van het probleem is het zoeken naar de kleuring waarbij zoveel mogelijk knopen Happy zijn.

2.1 Definitie

Beide definities hieronder zijn vertalingen van definitie 1 uit [28] en definitie 1 en 2 uit [19]. $\Gamma(v)$ is de set van nodes die met een boog verbonden zijn met v.

Definitie 2.1. De Happiness van een knoop in een ongerichte graaf $G = (V, E)$ met een volledige kleuring $c : V \to \{1, ..., k\}$ wordt gedefinieerd als volgt: een knoop $v \in V is$ Happy $\iff \forall u \in \Gamma(v) : c(v) = c(u)$.

Definitie 2.2. Het probleem vertrekt van een ongerichte graaf $G = (V, E)$ met n knopen en m bogen, een natuurlijk getal k, een subset van knopen $V' \subseteq V$ met $|V'| \geq k$ en een partiële kleuring $c : V' \to \{1, ..., k\}$ (de voorgekleurde knopen) zodat $\forall i \in \{1, ..., k\} : \exists v \in V' : c(v) = i$. Het doel is c uit te breiden tot een volledige kleuring $\bar{c} : V \to \{1, ..., k\}$ zodat het aantal Happy knopen gemaximaliseerd wordt.



Figuur 2.1: Voorbeeld van een optimale kleuring voor een graaf. Knopen met een '*' zijn voorgekleurd.

2.2 Voorbeeld

In figuur 2.1 is een voorbeeld te zien van een probleeminstantie van het MHV probleem. De graaf heeft 7 knopen en 9 bogen, er zijn 3 voorgekleurde knopen en ook 3 kleuren in totaal. Deze kleuring (die maximaal is) maakt 2 knopen Happy, c en e. Alle andere knopen zijn minstens verbonden met één knoop die een andere kleur heeft, en zijn dus Unhappy.

Hoofdstuk 3

Literatuuroverzicht

Dit hoofdstuk schetst een overzicht van het werk dat al gedaan is rond het Maximum Happy Vertices probleem. Aangezien het nog een relatief jong probleem is, zijn er slechts een beperkt aantal werken over geschreven. Origineel is het probleem beschreven door Li en Zhang in 2015 [28] in het kader van homofilie in netwerken. Zij tonen hoe het probleem praktisch nut heeft als het gebruikt wordt voor het toekennen van knopen in een netwerk aan verschillende clusters, indien een aantal knopen al een cluster toegewezen kreeg. Zij hebben ook twee constructieve algoritmen beschreven die verder worden uitgelegd in hoofdstuk 6. De rest van dit hoofdstuk geeft een overzicht van verschillende algoritmen en resultaten beschreven in de literatuur voor het MHV probleem.

3.1 Complexiteit van MHV

Li en Zhang hebben aangetoond dat het probleem in het algemeen NP-hard is[28], als het aantal kleuren minstens 3 is. Agrawal[1] heeft aangetoond dat het probleem Fixed Parameter Tractable is wanneer het geparametriseerd wordt door verschillende parameters, zoals het aantal kleuren en de treewidth van de graaf[1]. Dit betekent dat hoewel het algemene probleem NP-hard is, het probleem kan opgelost worden in $f(k)|x|^{\mathcal{O}(1)}$ tijd, met f een berekenbare functie en k een parameter in het probleem, hier de treewidth. Hetzelfde geldt voor vertex cover gecombineerd met de distance-to-clique parameter. Deze resultaten werden ook aangetoond in een paper van Aravind[4], waarin ook exacte algoritmen worden voorgesteld om het probleem op te lossen in $\mathcal{O}(2^n)$.

Voor specifieke grafen is het probleem wel oplosbaar in polynomiale tijd. Problemen met slechts 2 kleuren zijn een eerste klasse die eenvoudiger is op te lossen.

[1] De treewidth van een graaf is de grootte van de grootste knopen-set in de boom-ontbinding van die graaf. Een boom-ontbinding van een graaf G is een boom, waarvan elke knoop in de boom een subset is van de knopen van G, waarbij elke knoop van G minstens één keer voorkomt, alle boom-knopen die dezelfde knoop uit G bevatten verbonden zijn met elkaar, en er voor elke boog in G een boom-knoop is die beide knopen van de boog bevat.

Aravind et al.[3] hebben een exact algoritme beschreven om het probleem op te lossen voor boom-grafen. In dat geval is het probleem polynomiaal oplosbaar voor elk aantal kleuren k. Ook hebben ze aangetoond dat het probleem polynomiale complexiteit heeft voor grafen met begrensde treewidth of begrensde neighbourhood diversity.

3.2 Algemene oplossingsmethoden

Lewis et al.[19] hebben verschillende methoden voorgesteld om boven- en ondergrenzen te vinden voor het aantal happy knopen in een graaf. Verder tonen ze hoe een probleem zou kunnen opgesplitst worden in verschillende kleinere problemen door de graaf in verschillende stukken op te delen. Ze beschrijven ook twee IP voorstellingen voor het MHV probleem. Deze worden beide verder besproken in hoofdstuk 5.

Op het moment dat het werk voor deze thesis begonnen is, was er nog geen verder werk gebeurd rond algemene oplossingsmethoden voor het MHV probleem. Begin 2020 werd een paper gepubliceerd door Thiruvady et al.[25] (dezelfde auteurs als [19]) waarin een metaheuristisch algoritme gebaseerd op Tabu-search[9] beschreven wordt. In hoofdstuk 9 wordt de vergelijking gemaakt tussen dit algoritme en de metaheuristiek ontworpen in deze thesis. Verder wordt er in deze paper ook een methode voorgesteld om het probleem te verkleinen, namelijk door knopen die zeker (Un)Happy zijn extra voor te kleuren. Deze methode, en de verschillen met de methode voorgesteld in deze thesis worden verder besproken in hoofdstuk 4.

3.3 Praktische applicaties

Een praktische toepassing van het probleem is aangegeven in de paper van Lewis[19]: het vinden van een optimale planning voor het plaatsen van gasten aan verschillende tafels bij een groot evenement, zoals een trouwfeest. Elke gast heeft connecties met andere gasten indien deze elkaar kennen. Als er al enkele gasten aan een tafel zijn toegewezen, geeft het MHV probleem de optimale planning zodat er zoveel mogelijk gasten aan tafels zitten waar ze andere gasten kennen.

Meer algemeen geeft het MHV probleem een oplossing voor het toewijzen van items aan verschillende clusters in een netwerk, als er al een aantal toewijzingen zijn gebeurd. In dit geval staan de bogen in de graaf voor sterke relaties tussen verschillende items, en zoekt men naar de toekenning waarbij zoveel mogelijk gerelateerde items in dezelfde cluster zitten. De items kunnen bijvoorbeeld bestanden op een gedistribueerd opslagsysteem zijn, waar bestanden die sterk met elkaar gerelateerd zijn best op dezelfde cluster worden opgeslagen.

Hoofdstuk 4

Reductie van het probleem

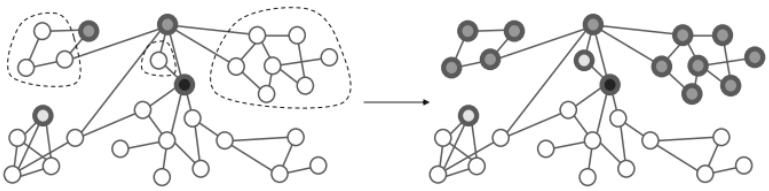
Een eerste stap in het oplossen van een optimalistatieprobleem is kijken of de zoekruimte kan verkleind worden door bepaalde eigenschappen van de instantie uit te buiten. Voor het MHV probleem kan men dit doen door te zoeken naar knopen waarvoor de kleur triviaal kan worden vastgelegd, en deze uit de zoekruimte uit te sluiten. In de paper van Thiruvady et al.[25] wordt een methode voorgesteld die extra kopen voorkleurt. Om de verschillen tussen hun methode en de nieuwe methode aan te kunnen tonen, legt sectie 4.1 hun aanpak uit. Sectie 4.2 legt de methode ontwikkeld in deze thesis, referentie-gebaseerde reductie, uit. Tot slot worden beide methodes algoritmisch vergeleken in sectie 4.3.

4.1 Reductiemethode van Thiruvady et al.

Deze sectie bestaat uit een beschrijving van de methode beschreven in sectie 2.4 van de paper van Thiruvady et al.[25]. Het basisidee achter de methode is dat de kleur van bepaalde knopen kan vastgelegd worden, zonder het aantal Happy knopen te veranderen. De kleur van een knoop vastleggen komt neer op het toevoegen van een extra voorgekleurde knoop aan het probleem. Dit zorgt voor een kleiner aantal vrije (niet voorgekleurde) knopen, en dus ook een kleinere zoekruimte.

De methode beschrijft twee types knopen waarvoor de kleur kan vastgelegd worden. Het eerste type zijn knopen die slechts aan voorgekleurde knopen verbonden zijn met één kleur. Beschouw een vrije knoop v, en de set S van alle vrije knopen die bereikbaar zijn in de subgraaf van vrije knopen vanuit v. De set W van alle knopen verbonden met een knoop in S, en die zelf niet in S zitten, zijn dus enkel voorgekleurde knopen. Als W leeg is, dan is deze groep niet geconnecteerd met een voorgekleurde knoop, en zal deze groep altijd volledig Happy zijn, zolang alle knopen dezelfde kleur krijgen. Als de set W enkel knopen bevat met één kleur, dan kan de hele groep Happy zijn door deze kleur te krijgen, en dit heeft geen invloed op de Happiness van de aangrenzende knopen, aangezien deze ook die kleur hebben.

Het tweede type knopen zijn knopen die enkel verbonden zijn met voorgekleurde knopen die niet meer Happy kunnen zijn. Deze knopen kunnen eender welke kleur krijgen, aangezien al hun buren al gegarandeerd Unhappy zijn.



Figuur 4.1: Voorbeeld van de reductiemethode van Thiruvady et al. Figuur overgenomen uit [25](Fig. 3)

Figuur 4.1 toont een situatie waarin verschillende knopen extra kunnen voorgekleurd worden. De groep linksboven is verbonden met twee voorgekleurde knopen, maar deze hebben dezelfde kleur, zodat de groep ook die kleur kan krijgen. Hetzelfde geldt voor de groep rechtsboven. De knoop in het midden is verbonden met twee voorgekleurde knopen met een verschillende kleur en deze zijn beide verbonden met elkaar. Dit betekent dat deze knopen nooit Happy kunnen gemaakt worden, en de knoop zelf kan dus eender welke kleur toegewezen krijgen.

4.1.1 Algoritme

In de paper van Thiruvady et al. wordt niet beschreven hoe deze methode kan geïmplementeerd worden, maar ze voorzien wel een implementatie in hun source code[18]. Hun algoritme wordt in algoritme 4.1 in pseudocode voorgesteld om de vergelijking met het alternatieve algoritme in sectie 4.3 eenvoudiger te maken.

De eerste stap overloopt alle knopen in de graaf G, en controleert voor alle vrije knopen of ze van het eerste type zijn. Dit wordt gedaan door te kijken naar de vrije component waar v in zit. Deze component is eenvoudig te berekenen a.d.v. Breadth-first search (BFS) in de subgraaf van vrije knopen. De aangrenzende kleuren van S kunnen dan eenvoudig worden berekend door iteratief alle knopen in S te overlopen en de voorgekleurde knopen te verzamelen. Deze set zou al kunnen opgevuld worden in de BFS fase, maar dit is niet geïmplementeerd, ze wordt aangemaakt met een tweede iteratie over alle knopen in de component.

De tweede stap, vanaf lijn 12, berekent eerst de status voor elke knoop in de graaf. Deze status is ofwel ‘Unhappy’, voor voorgekleurde knopen die verbonden zijn met een andere voorgekleurde knoop met een andere kleur, ofwel ‘voorbestemd Unhappy’, voor vrije knopen die verbonden zijn met voorgekleurde knopen met minstens 2 verschillende kleuren, ofwel ‘overig’. De volgende loop controleert voor alle voorbestemd Unhappy knopen of ze enkel verbonden zijn met Unhappy knopen, en geeft ze in dat geval een random kleur.

Algoritme 4.1 Toevoegen van extra voorgekleuringen

```
1: for all v ∈ G do
2:   if !isPrecolored(v) then
3:     S ←Vrije component van v {Berekend met Breadth-first search}
4:     W ←Aangrenzende kleuren van S
5:     if #W = 0 then
6:       ∀x ∈ S : c(x) = random(1..k) {Eender welke kleur}
7:     else if #W = 1 then
8:       ∀x ∈ S : c(x) = g {g is een verbonden voorgekleurde knoop}
9:     end if
10:  end if
11: end for
12: for all v ∈ G do
13:   status[v] ← getStatus(v) {Unhappy, voorbestemd Unhappy, of overig}
14: end for
15: for all v ∈ G do
16:   if status[v] = voorbestemd Unhappy then
17:     if !∃x ∈ Γ(v) : status[x] = overig then
18:       c(v) = random(1..k) {Eender welke kleur}
19:     end if
20:   end if
21: end for
```

4.2 Referentie-gebaseerde reductie

De aanpak van het reductie-algoritme hier voorgesteld is anders dan in de methode van Thiruvady et al., en kan bekeken worden als een veralgemening van deze methode. Het gebruikte concept houdt in dat knopen waarvan de kleur triviaal bepaald kan worden, uit de probleemgraaf gehaald worden voor de zoekfase. Deze knopen worden opgeslagen met een referentie naar een knoop in de gereduceerde graaf. Nadat de (optimale) kleuring voor de overblijvende knopen bepaald is, worden de gereduceerde knopen terug toegevoegd aan de oplossing, en ze nemen de kleur aan van hun referentieknoop. Dit laat toe om ook knopen te elimineren waarvoor de exacte kleur niet triviaal kan bepaald worden, maar wel kan gelijkgesteld worden met de kleur van een andere knoop.

Formeel komt deze reductie neer op het opstellen van een gereduceerde graaf $G'(V', E')$ en de referentiefunctie $R : V \backslash V' \to V'$. De graaf G' is een subgraaf van de originele graaf G, en wordt opgesteld door knopen te verwijderen uit V. De referentiefunctie R mapt elke knoop die niet in de gereduceerde graaf zit op een knoop die er wel in zit. Als dan een oplossing voor het MHV probleem berekend is voor G': $\bar{c}'$, moet deze uitgebreid worden naar een volledige kleuring van G: $\bar{c}$. Dit gebeurt aan de hand van vergelijking 4.1. Elke verwijderde knoop krijgt de kleur van de knoop waar zijn referentie naar wijst.

$$\begin{aligned} \forall v' \in V' : &\bar{c}(v') = \bar{c}'(v') \\ \forall v \in V \backslash V' : &\bar{c}(v) = \bar{c}'(R(v')) \end{aligned} \tag{4.1}$$

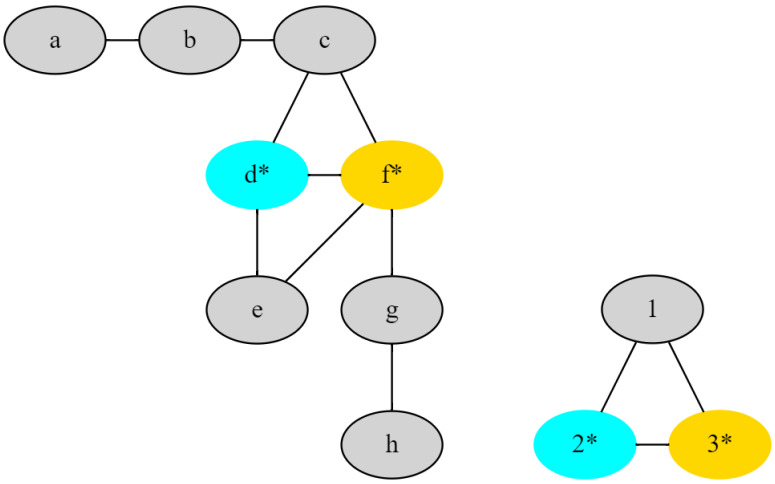
Het opstellen van de gereduceerde graaf en de referentiefunctie kan op verschillende manieren. De triviale reductie bijvoorbeeld verwijdert geen enkele knoop, en de complete reductie verwijdert elke vrije knoop uit de graaf en heeft als referentiefunctie de volledige oplossing van het probleem. Deze reducties hebben beide geen praktisch nut, aangezien ze het probleem niet eenvoudiger maken. De meest interessante reducties zijn degene die eenvoudig te berekenen zijn, en de zoekruimte toch aanzienlijk verkleinen.

4.2.1 Opstellen van de reductie

Een eerste reductie die eenvoudig te berekenen is, zijn de extra voorkleuringen uit de methode van Thiruvady et al. De knopen van het eerste type (vrije component slechts verbonden met één kleur) worden vervangen door een referentie naar één van de verbonden voorgekleurde knopen. Als de component niet geconnecteerd is met een voorgekleurde knoop, kan een triviale referentie worden ingevoegd, omdat elke kleur optimaal is. Knopen van het tweede type (knopen enkel verbonden met Unhappy voorgekleurde knopen met verschillende kleuren) kunnen ook vervangen worden door een triviale reductie, omdat voor deze knopen de kleur niet uitmaakt.

De volgende reductie die beschouwd wordt kijkt naar vrije knopen met graad 1. Deze knopen kunnen triviaal Happy gemaakt worden door ze de kleur van hun enige buur te geven. Dit werkt ook als hun buur niet voorgekleurd is, dankzij het referentiesysteem. Deze stap kan iteratief herhaald worden, aangezien het verwijderen van knopen met graad 1 de graad van hun buren verlaagt. Om dit iteratieve proces te vermijden, kan er in de originele graad gezocht worden vanuit de knopen met graad 1 naar vrije knopen met graad 2. Deze knopen vormen een ketting, die begint bij een knoop met graad 1, verder bestaat uit geen of meer knopen van graad 2, tot deze eindigt bij een knoop met een graad hoger dan 2, of een voorgekleurde knoop. Deze 'anker'-knoop is dan de referentie voor alle knopen in de ketting.

Figuur 4.2 toont hoe een graaf (a) met deze methode kan worden gereduceerd tot een kleinere graaf (b) aan de hand van de referenties in (c). Knopen g en h zijn van het eerste type, en krijgen dus een referentie naar de knoop waar hun component mee verbonden is (f). Knoop e krijgt geen referentie omdat elke kleur optimaal is, (type twee). Knopen a en b krijgen een referentie naar c, aangezien ze een ketting vormen met c als ankerknoop. Verder is het ook belangrijk om te zien dat ook de knopen die niet gereduceerd worden (c, d en f) een referentie krijgen naar de overeenkomstige knoop in de kleinere gereduceerde graaf (respectievelijk 1, 2 en 3). Deze worden berekend in een tweede stap, waar alle niet-gereduceerde knopen worden overgezet



(a) Originele graaf (b) Gereduceerde graaf

Originele knoop	Reductie-referentie	Referentie
a	c	1
b	c	1
c		1
d		2
e	/	/
f		3
g	f	3
h	f	3

(c) Referenties

Figuur 4.2: Voorbeeld van een reductie.

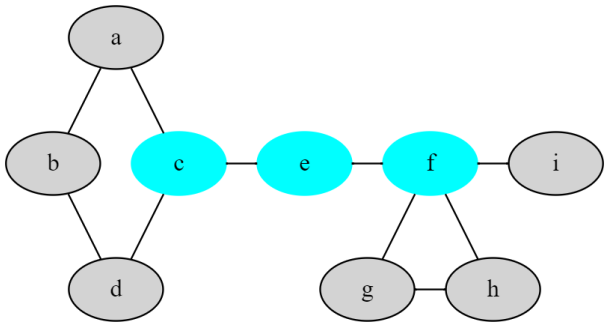
naar de nieuwe graaf. Daar worden ook de reductie-referenties omgezet naar de uiteindelijke referenties naar knopen in de gereduceerde graaf.

4.2.2 Articulatie-gebaseerd algoritme

Dit concept kan veralgemeend worden aan de hand van articulatie-knopen. Dit zijn knopen die als ze uit de graaf verwijderd zouden worden het aantal componenten in de graaf omhoog doen stijgen. (Zie figuur 4.3 voor een voorbeeld) Deze articulatie-knopen verdelen de graaf in 2-geconnecteerde componenten. Elke 2-geconnecteerde component die slechts met één articulatie-knoop verbonden is, en geen voorgekleurde knopen bevat, kan gereduceerd worden door deze een referentie te geven naar de articulatie-knoop. Een mogelijk probleem bij deze aanpak is dat knopen in een ketting zoals voordien niet in één 2-geconnecteerde component zitten. Elke knoop in de ketting met graad 2 is namelijk een articulatie-knoop, omdat als deze zou verwijderd worden, het uiteinde van de ketting niet meer geconnecteerd is met de graaf. Een andere manier om dit te bekijken is dat articulatie-knopen met graad 2

altijd verbonden zijn met twee andere articulatie-knopen, of met knopen met graad 1. Het is eenvoudig om deze knopen te ontwijken door enkel articulatie-knopen met graad groter dan 2 te beschouwen. Voorgekleurde knopen zijn niet noodzakelijk articulatie-knopen, maar omdat deze de graaf op gelijkaardige wijze in stukken verdelen, worden deze ook beschouwd als grensknopen die de graaf in componenten verdelen.

Algoritme 4.2 toont pseudocode om een graaf te reduceren aan de hand van de articulatie-knopen. Deze kunnen berekend worden aan de hand van het algoritme opgesteld door Hopcroft en Tarjan[12]. Dit algoritme komt neer op Depth-First search (DFS) waarbij extra informatie wordt bijgehouden. De articulatie-knopen zijn dus eenvoudig en efficiënt te berekenen. In de pseudocode wordt verondersteld dat de informatie over de articulatie-knopen al berekend is en beschikbaar is in de *articulation* array.



Figuur 4.3: Voorbeeld van articulatie-knopen in een graaf. Gekleurde knopen zijn articulatie-knopen.

De vrije component van v wordt op gelijkaardige wijze berekend als in algoritme 4.1, aan de hand van BFS. Hier wordt er echter niet alleen rekening gehouden met voorgekleurde knopen als grenspunten, maar ook met de articulatie-knopen. Tijdens BFS kan de informatie over de verbonden voorgekleurde en articulatie-knopen al worden opgebouwd. Deze informatie bepaalt of een component gereduceerd kan worden.

Als een component niet verbonden is met voorgekleurde knopen of articulatie-knopen, is het een vrije component. Deze kan gereduceerd worden met een triviale referentie, aangezien elke kleur leidt tot de optimale oplossing voor de hele component. Als de component verbonden is met één articulatie-knoop, en geen enkele voorgekleurde knoop, dan kan de component gereduceerd worden met een referentie naar deze articulatie-knoop. Het laatste geval is dat de component verbonden is met voorgekleurde knopen van één kleur, en met geen enkele articulatie-knoop. Dan kan de component gereduceerd worden met een referentie naar één van de voorgekleurde knopen. In alle andere gevallen kan er niets algemeen gezegd worden over de compo-

nent.

Algoritme 4.2 Referentie-reductie met ariculatie-knopen

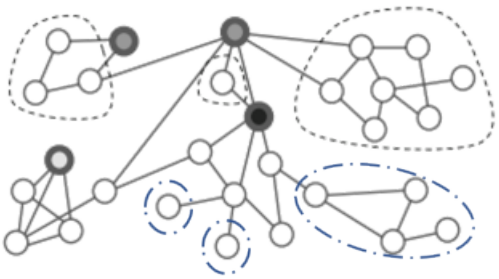
```
1: for all v ∈ V do
2:   if !articulation[v] and !isPrecolored(v) then
3:     C ← Vrije component van v
4:     A ← Articulatie-knopen verbonden met C
5:     G ← Aangrenzende kleuren van C
6:     if #A = 0 and #G = 0 then
7:       ∀x ∈ C : reference[x] = −1 {Eender welke kleur}
8:     else if #A = 1 and #G = 0 then
9:       ∀x ∈ C : reference[x] = a|A = {a}
10:    else if #A = 0 and #G = 1 then
11:      ∀x ∈ C : reference[x] = g {g is een verbonden voorgekleurde knoop}
12:    end if
13:  end if
14: end for
```

Het is mogelijk dat na een eerste iteratie van het algoritme nog knopen overblijven die kunnen gereduceerd worden, omdat als een component verwijderd wordt de articulatie-knoop waarmee deze verbonden is mogelijk geen articulatie-knoop meer is. Het is mogelijk om het algoritme verschillende keren na elkaar toe te passen op een graaf, en zo een steeds kleinere graaf te bekomen. Elke iteratie introduceert een aantal referenties, en de originele graaf kan bekomen worden door deze in de omgekeerde volgorde toe te passen.

De enige knopen die nog gereduceerd kunnen worden als het algoritme geen veranderingen meer toevoegt zijn voorgekleurde knopen zonder verbindingen, omdat deze nooit in een component kunnen zitten, en knopen van het type twee van Thiruvady's methode. Deze kunnen ook nog gereduceerd worden door een extra check over alle overblijvende knopen, die enkel voor deze gevallen controleert.

4.2.3 Voorbeeld

Figuur 4.4 is een aangepaste versie van figuur 4.1. De omcirkelde knopen links- en rechtsboven worden door de methode van Thiruvady et al. extra voorgekleurd, dit effect kan ook bekomen worden door deze knopen te verwijderen en ze elk te bewaren met een referentie naar de voorgekleurde knoop waarmee ze verbonden zijn. Voor vrije componenten of knopen zoals de middelste omcirkelde knoop kan een triviale referentie toegevoegd worden, aangezien elke kleur optimaal is. De eerste extra reductie die voorgesteld wordt, knopen met graad 1 en de verbonden kettingen, zou de twee knopen onderaan en de ene knoop in de omcirkelde groep rechtsonder ook reduceren tot een referentie naar hun verbonden knoop. De methode met articulatie-knopen maakt al deze reducties, en reduceert ook nog de groep rechtsonder. In een



Figuur 4.4: Voorbeeld van knopen die kunnen uitgesloten worden met referentie-gebaseerde reductie. Figuur bewerkt overgenomen uit [25] (Fig. 3).

eerste iteratie zal het enkel de knoop met graad 1 reduceren, in de tweede iteratie zal het de buitenste 2 knopen reduceren, en in de derde iteratie zal het de laatste knoop reduceren. Uiteindelijk zullen alle vier knopen een referentie hebben naar de knoop waarmee de component verbonden is.

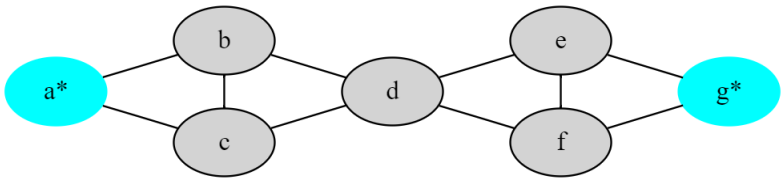
4.3 Vergelijking reductie-algoritmen

Hier wordt een algoritmische vergelijking gemaakt tussen beide varianten van referentie-gebaseerde reductie en de originele methode van Thiruvady et al. Voor de vergelijking van de verschillende methodes gebaseerd op experimentele resultaten, zie sectie 9.1.

De eerste eenvoudige variant van referentie-gebaseerde reductie is een directe uitbreiding op de originele methode, aangezien hetzelfde algoritme gebruikt wordt. Dit is ook duidelijk te zien in de experimentele resultaten. Deze methode is in snelheid zeer vergelijkbaar met de originele methode, aangezien het slechts één extra controle uitvoert in de eerste loop over alle knopen.

De tweede variant, die gebruik maakt van articulatie-knopen, is bijna een directe uitbreiding op de eenvoudige variant. Er is slechts één geval dat door de eenvoudige variant zou gereduceerd worden, en niet gedetecteerd wordt in deze methode. Figuur 4.5 toont een voorbeeld van dit geval. Alle ongekleurde knopen behoren tot één vrije component, en zijn dus knopen van het eerste type en kunnen gereduceerd worden met een referentie naar a of g. Door de specifieke structuur van de vrije component zal deze niet gereduceerd worden door de articulatie-methode. Knoop d is namelijk een articulatie-knoop, en verdeelt dus de component in twee kleinere componenten ($\{b, c\}$ en $\{e, f\}$). Deze worden beide begrensd door één voorgekleurde knoop en één articulatie-knoop, en kunnen dus niet gereduceerd worden.

Er is dus een extreem geval waarvoor de uitgebreide methode faalt ten opzichte van de eenvoudige methode, maar dit komt praktisch gezien bijna nooit voor. De specifieke configuratie die benodigd is, komt niet snel voor, en als er een ander pad



Figuur 4.5: Voorbeeld van het geval dat niet wordt gereduceerd door de articulatie-methode. Knopen a en g zijn voorgekleurd met dezelfde kleur.

bestaat tussen a en g in de graaf is d geen articulatie-knoop en is er dus geen probleem.

Zoals ook in de experimentele resultaten te zien is, is de articulatie-methode praktisch gezien een uitbreiding op de eenvoudige methode, ondanks het probleem-geval. Om deze meestal kleine verbetering te bekomen, moeten er wel veel extra berekeningen worden gemaakt. Waar de eenvoudige methode in totaal twee keer alle knopen in de originele graaf overloopt, gebeurt dit in de articulatie-methode veel meer. Elke iteratie overloopt de knopen twee keer, eerst om de articulatie-knopen te berekenen en daarna om de componenten op te bouwen en te controleren. Het kan zijn dat er meerdere iteraties nodig zijn, die elk opnieuw alle overblijvende knopen twee keer overlopen. Nadat de articulatie-iteraties afgelopen zijn, moeten de overblijvende knopen nog twee keer overlopen worden om te controleren op niet-geconnecteerde knopen, en knopen van het tweede type. Dit betekent dat hoewel beide reductie-methodes lineaire complexiteit hebben, de articulatie-methode in praktijk altijd een stuk trager zal zijn dan de eenvoudige methode.

4.4 Besluit

Het idee van referentie-gebaseerde reductie voorgesteld in dit hoofdstuk laat toe om de methode beschreven door Thiruvady et al. uit te breiden tot de twee methoden die hier beschreven worden. De eenvoudige methode is zoals de naam aangeeft eenvoudig om te berekenen, en levert toch een substantiële verbetering op.

De tweede methode maakt gebruik van articulatie-knopen, om de graaf te verdelen in componenten. Deze methode komt praktisch gezien neer op een uitbreiding van de eenvoudige methode, hoewel er één geval is (Figuur 4.5) dat niet wordt gedetecteerd voor reductie, ondanks dat het een eenvoudig geval is in de eerste methode. De kleine verbetering die deze methode realiseert ten opzichte van de eenvoudige methode komt wel ten koste van extra complexiteit en uitvoeringstijd. Sectie 9.1 bespreekt alle methodes verder aan de hand van experimenten.

Aangezien de extra benodigde uitvoeringstijd relatief ten opzichte van de uitvoeringstijd van het metaheuristisch algoritme zeer beperkt is, zullen probleem-grafen in de rest van deze thesis, behalve voor de analyse van de constructieve algoritmen in hoofdstuk 6, altijd eerst gereduceerd worden aan de hand van de articulatie-methode.

Hoofdstuk 5

Exacte solver

Naast constructieve en metaheuristische algoritmen is het ook mogelijk een optimalisatieprobleem op te lossen door gebruik te maken van Integer Programming (IP). Als het probleem kan vertaald worden naar een IP model, kunnen krachtige solvers gebruikt worden om dit model op te lossen. IP is namelijk een gebied waarvoor al uitgebreid onderzoek gevoerd is, bijgevolg zijn er verschillende zeer snelle softwarepaketten beschikbaar om IP modellen te optimaliseren.

De term 'exact' die gebruikt wordt om deze solver te beschrijven, verwijst naar het feit dat dit type solvers zoekt naar het globale optimum, de exacte optimale oplossing. Als deze oplossing niet gevonden wordt in het beschikbare tijdsbudget, wordt een voorlopige oplossing opgesteld, samen met een schatting van het verschil dat nog bestaat tussen deze oplossing en het optimum.

Sectie 5.1 bespreekt het IP model voorgesteld door Lewis et al. [19] dat ze ook gebruiken voor hun resultaten. Sectie 5.2 bespreekt het alternatieve model dat ook in hun paper wordt voorgesteld, maar dat ze niet gebruiken. Deze sectie behandelt ook de verschillen tussen beide modellen. De implementatie van deze modellen in de source code van deze thesis[21] maakt gebruik van *CPLEX* [1] (versie 12.9), de solver ontwikkeld door IBM, om alle experimenten uit te voeren.

5.1 IP model

Het model dat door Lewis et al.[19] gebruikt wordt om hun resultaten te bekomen definieert voor elke knoop in de MHV instantie een variabele x_i met domein $\{1, \ldots, k\}$ en een binaire variabele y_i met domein $0, 1$. De x_i variabelen geven aan wat de kleur van knoop i is, en y_i geeft aan of de knoop Happy is of niet, y_1 is gelijk aan 0 als de knoop Happy is, en gelijk aan 1 als de knoop Unhappy is. De doelfunctie van het MHV probleem, het maximaliseren van het aantal Happy knopen, wordt weergegeven

[1] `www.ibm.com/analytics/cplex-optimizer`

in vergelijking 5.1.

$$minimize \quad \sum_{i=0}^{n} y_i \tag{5.1}$$

Hier moeten nog beperkingen aan toegevoegd worden, die de structuur van de graaf en de voorkleuringen vertalen naar een IP formulering. Vergelijking 5.2 vertaalt alle voorkleuringen ($V' = \{v | v \in V \wedge isPrecoloured(v)\}$) in een beperking die aangeeft dat de kleur van de knoop in kwestie vast ligt. De Happiness van een knoop wordt bepaald door de constraints in vergelijking 5.3. Voor elke knoop is er een beperking per verbonden knoop die aangeeft dat deze knoop Unhappy is ($y_i = 1$) als de kleuren van beide knopen verschillen. In het model van Lewis et al. is de noemer in deze beperking gelijk aan n, in plaats van k. Dit maakt eigenlijk geen verschil, aangezien elk getal groter dan of gelijk aan k hiervoor zou werken, het verschil tussen twee x_i's is namelijk maximaal $k - 1$. Hier is de keuze gemaakt voor k, om eventuele afrondingsfouten uit te sluiten in het geval dat n veel groter is dan k.

$$\forall v_i \in V' : c(v_i) \tag{5.2}$$

$$\forall v_i \in V : \forall v_j \in \Gamma(v_i) : y_i \geq \frac{|x_i - x_j|}{k} \tag{5.3}$$

In totaal wordt een MHV instantie met n knopen, e bogen en p voorgekleurde knopen omgezet in een IP model dat gebruik maakt van $2n$ variabelen en waarvan de doelfunctie n termen heeft. Hier komen p beperkingen bij voor de voorgekleurde knopen, en $2e$ beperkingen die de bogen voorstellen. Dit is op zichzelf geen lineair IP-model, omdat het gebruikt maakt van absolute waarden. Dit is namelijk geen lineaire operator, maar deze kan wel omgezet worden in twee lineaire beperkingen om het geheel lineair te maken [10]. Dit wordt automatisch uitgevoerd door de meeste solvers, en wordt bijgevolg niet gespecifieerd in de definitie van het IP model.

5.2 Alternatief IP model

Lewis et al. stellen ook een alternatief IP model voor om het MHV probleem voor te stellen. Zij gebruikten dit model eerst, maar uit hun experimenten met beide modellen a.h.v. hun implementatie in Gurobi [2] bleek dat het bovenstaande model betere resultaten behaalde.

Dit model gebruikt meer variabelen, voor elke knoop worden k binaire variabelen x_{ij} geïntroduceerd die aangeven of knoop i kleur j heeft, en een extra binaire variabele y_i die aangeeft of de knoop Unhappy is. Dit geeft dezelfde doelfunctie als in het model hierboven, maar er zijn wel andere beperkingen nodig.

[2] `www.gurobi.com/`

$$\forall v_i \in V', c(v_i) = j : x_{ij} = 1 \tag{5.4}$$

$$\forall v_i \in V : \sum_{j=1}^{k} x_{ij} = 1 \tag{5.5}$$

$$\forall v_i \in V : \forall v_j \in \Gamma(v_i) : \forall l \in \{1, \ldots, k\} : y_i \geq |x_{il} - x_{jl}| \tag{5.6}$$

Vergelijking 5.4 toont de alternatieve versie van vergelijking 5.2. Hier wordt ook de kleur van een voorgekleurde knoop vastgelegd, maar de beperking maakt hierbij gebruik van de binaire variabelen. Vergelijking 5.5 voegt een extra beperking toe voor elke knoop die garandeert dat een knoop slechts één kleur kan hebben. Dit is een inherente beperking in het eerste model, aangezien de variabele die daarin de kleur van een knoop aangeeft slechts één waarde kan hebben. Tot slot geeft vergelijking 5.6 de alternatieve versie van de beperking in vergelijking 5.3.

Dit model maakt gebruik van meer variabelen dan het model voorgesteld in de vorige sectie. In totaal zijn er $(k+1)n$ variabelen nodig in plaats van $2n$. Het verschil is wel dat deze variabelen allemaal binair zijn, waarbij de helft van de variabelen in het eerste model natuurlijke getallen zijn. De doelfunctie is identiek voor beide modellen.

Het alternatieve model heeft p beperkingen voor de voorgekleurde knopen, n beperkingen om te garanderen dat een knoop slechts één kleur heeft en $2ek$ beperkingen die de bogen voorstellen. Een volledig overzicht wordt gegeven in tabel 5.1, waar de grootte van beide modellen naast elkaar wordt gezet. Dit laat zien dat het alternatieve model altijd $(k-1)n$ variabelen en $n + (k-1)e$ beperkingen meer heeft dan het eerste model. Dit komt volledig door het feit dat binaire variabelen gebruikt worden in het alternatieve model, in plaats van de gehele variabelen in het eerste model.

	Eerste model	**Alternatief model**
# variabelen	$2n$	$(k+1)n$
# termen in doelfunctie	n	n
# beperkingen	$p + 2e$	$p + n + 2ek$

Tabel 5.1: Overzicht van de grootte van beide IP-modellen voorgesteld in dit hoofdstuk.

5.3 Besluit

In dit hoofdstuk worden twee verschillende Integer Programming modellen voorgesteld die origineel beschreven zijn door Lewis et al.[19]. Het eerste model, dat door Lewis et al. wordt gebruikt voor hun resultaten, is een stuk compacter dan het alternatieve model, maar gebruikt variabelen met een groter domein.

Lewis et al. vermelden dat hun implementatie van het alternatieve model, gebruik makend van een implementatie van deze modellen in Gurobi, voor bepaalde MHV instanties betere resultaten geeft, maar dat ze niet hebben kunnen afleiden of hier een patroon in zit. Omdat bleek dat het alternatieve model minder betrouwbaar is, gebruiken zij voor de al hun experimenten enkel het eerste model.

In sectie 9.2 worden beide modellen experimenteel vergeleken a.d.h.v. een implementatie in CPLEX. Dit laat toe om de bevindingen van Lewis et al. over het alternatieve model na te gaan voor een andere solver, en om de bovenstaande vergelijking van beide modellen in tabel 5.1 ook experimenteel te controleren.

Hoofdstuk 6

Constructieve algoritmen

Een constructief algoritme voor een optimalisatieprobleem bouwt een oplossing in één keer, zonder later veranderingen aan te brengen. Dit betekent dat dit soort algoritmen vaak zeer snel een oplossing berekent, maar dat de oplossing slechts een benadering is van de optimale oplossing.

Dit hoofdstuk bespreekt eerst de twee constructieve algoritmen beschreven door Li en Zhang[28], Greedy-MHV en Growth-MHV. Deze algoritmen vormen een basis voor het metaheuristisch algoritme dat besproken wordt in hoofdstuk 7. Sectie 6.2 bespreekt een nieuw exact algoritme voor 2-reguliere grafen. Dit algoritme is gebaseerd op het Growth-MHV algoritme, maar maakt enkel optimale kleuringen en bekomt zo altijd de optimale oplossing.

6.1 Constructieve algoritmen van Li en Zhang

Li en Zhang [28] hebben bij de initiële voorstelling van het MHV probleem twee constructieve algoritmen geïntroduceerd. Beide algoritmen hebben slechts een beperkte uitvoeringstijd, aangezien ze slechts één enkele iteratie nodig hebben om de volledige kleuring te berekenen. Dit laat toe om deze oplossing(en) te gebruiken als initiële oplossing bij de metaheuristiek.

Omdat deze algoritmen uitgebreid gebruikt worden in de rest van deze thesis, worden ze hieronder volledig gedefinieerd en geanalyseerd.

6.1.1 Greedy-MHV

De greedy aanpak wordt vaak gebruikt als eenvoudige techniek om snel een oplossing te genereren voor een optimalisatie-probleem. Een klassiek greedy algoritme bouwt zijn oplossing door bij elke beslissing altijd de keuze te maken die op dat moment het doel zo veel mogelijk verbetert. Dit levert algemeen gezien bijna nooit de optimale oplossing op, maar is zeer eenvoudig te berekenen.

Het Greedy-MHV algoritme bouwt zijn oplossing door alle vrije knopen in de graaf dezelfde kleur toe te kennen. Dit kan nog op k, het aantal kleuren, verschillende manieren. De uiteindelijke oplossing wordt bepaald door elke kleur uit te proberen. De kleur met het grootste aantal Happy knopen bepaalt dan de oplossing van het algoritme.

Dit algoritme is zeer eenvoudig, deterministisch, en heeft duidelijk slechts polynomiale tijd nodig. Voor elke kleur moet het de Happiness controleren van elke knoop, wat neerkomt op het afgaan van alle verbonden knopen. De complexiteit van dit algoritme is bijgevolg $\mathcal{O}(k * e)$, met e het aantal bogen in de graaf.

6.1.2 Growth-MHV

Het Growth-MHV algoritme van Li en Zhang, ook subset-growth genoemd, probeert de kleur-klassen V_i te laten groeien door er zo veel mogelijk knopen aan toe te voegen die Happy kunnen worden op dat moment. Dit komt ook neer op een greedy aanpak, omdat in elke stap de knoop gekozen wordt die het aantal Happy knopen zoveel mogelijk verhoogt.

Praktisch gebruikt het algoritme een labeling om eenvoudiger te kunnen redeneren over de status waarin een knoop zich bevindt tijdens het algoritme.

Definitie 6.1. Mogelijke labels voor knopen die al een kleur hebben:

- v is een H-knoop als alle verbonden knopen dezelfde hebben kleur, i.e. de knoop is Happy.
- v is een U-knoop als minstens één verbonden knoop een andere kleur heeft.
- v is een P-knoop als deze knoop Happy kan worden. Dit betekent dat alle reeds gekleurde verbonden knopen dezelfde kleur hebben.

Definitie 6.2. Alle knopen die nog geen kleur hebben zijn L-knopen. Aan de hand van de status van verbonden knopen zijn er nog verschillende onderverdelingen:

- v is een LP-knoop als deze verbonden is met een P-knoop.
- v is een LH-knoop als deze niet verbonden is met een P-knoop, en enkel verbonden is met U-knopen van één kleur en dus Happy kan worden.
- v is een LU-knoop als deze niet verbonden is met een P-knoop, en verbonden is met U-knopen met verschillende kleuren, en dus gegarandeerd is om Unhappy te zijn.
- v is een LF-knoop als deze niet verbonden is met een gekleurde knoop.

Het algoritme zelf werkt in verschillende iteraties, door in elke iteratie één of meerdere knopen te kleuren. Algoritme 6.1 toont de pseudocode voor Growth-MHV. Dit is een licht aangepaste versie van het originele algoritme gedefinieerd in [28]. Deze pseudocode volgt de definitie van Lewis et al.[19]. Het originele algoritme veronderstelt dat het algoritme enkel gebruikt wordt voor geconnecteerde grafen, omdat niet-geconnecteerde grafen kunnen opgesplitst worden in geconnecteerde componenten. Voor geconnecteerde grafen is het onmogelijk om op lijn 9 terecht te komen in het algoritme, aangezien een LF-knoop enkel kan bestaan als deze verbonden is met minstens één andere L-knoop. Dit betekent dat er dus een LP-, LH- of LU-knoop moet bestaan, aangezien er altijd minstens één knoop gekleurd is, en de graaf geconnecteerd is.

Algoritme 6.1 Growth-MHV

```
1:  while ∃ L-knoop do
2:    if ∃ P-knoop v then
3:      Kleur alle LP-knopen in Γ(v) met de kleur van v.
4:    else if ∃ LH-knoop v then
5:      Kleur v en alle andere L-knopen in Γ(v) met de kleur van een U-knoop
        verbonden met v.
6:    else if ∃ LU-knoop v then
7:      Kleur v met de kleur van één van de U-knopen verbonden met v.
8:    else
9:      Kleur v, een LF-knoop, met een random kleur.
10:   end if
11:   Herbereken de labels voor alle relevante knopen.
12: end while
```

Een ander verschil tussen deze pseudocode en de originele, is de specificatie van de knopen waarvoor het label moet worden herberekend. Dit verschilt voor de vier verschillende gevallen van de status van de aangepaste knoop in het algoritme. Voor geval 3 en 4 (LU en LF) moet enkel het label van de knoop zelf, en alle verbonden knopen herberekend worden. Voor geval 1 en 2 (P en LH) moeten meer knopen worden nagekeken. Het originele algoritme zegt dat dit alle knopen in $\Gamma^2(v)$ zijn. Zoals Lewis et al. opmerken, is dit niet correct.

Veronderstel het volgende voorbeeld (overgenomen uit [19]): er is een pad van vier knopen in een graaf (v_1, v_2, v_3, v_4) en knopen v_1 en v_3 zijn voorgekleurd met verschillende kleuren. (P, LP, P, LP) is dan een mogelijke labeling voor de knopen. Als v_1 door het algoritme geselecteerd wordt, zal v_2 de kleur van v_1 krijgen. Dit zorgt ervoor dat v_3 een U-knoop wordt en dat v_4 een LH- of een LU-knoop wordt. Voor geval 1 en 2 moeten dus alle knopen in $\Gamma^3(v)$ bekeken worden.

In elke iteratie probeert het algoritme de beste keuze op een greedy manier te maken. Voor het eerste en het tweede geval komt dit neer op het Happy maken van

v. De keuze voor de knoop in het derde geval maakt op dat ogenblik niet meer uit, aangezien alle buren van deze knoop al Unhappy zijn, onafhankelijk van de kleur van deze knoop. Het vierde geval heeft geen invloed op de totale Happiness van de graaf, aangezien dit geval enkel kan voorkomen als de graaf uit verschillende componenten bestaat, en één van die componenten geen voorgekleurde knopen bevat.

Een extra keuze die in elke iteratie moet gemaakt worden, is welke knoop behandeld wordt. Als er een P-knoop bestaat, kunnen er ook meerdere bestaan, en moet er dus gekozen worden welke gebruikt wordt. Dit maakt dat het algoritme inherent niet-deterministisch is, wat door zowel Li en Zhang als Lewis et al. niet expliciet wordt opgemerkt. Indien de selectiemethode met een random selectie-operator wordt uitgevoerd, is het bijgevolg mogelijk om het algoritme meerdere keren uit te voeren, en hierbij mogelijk verschillende resultaten te bekomen.

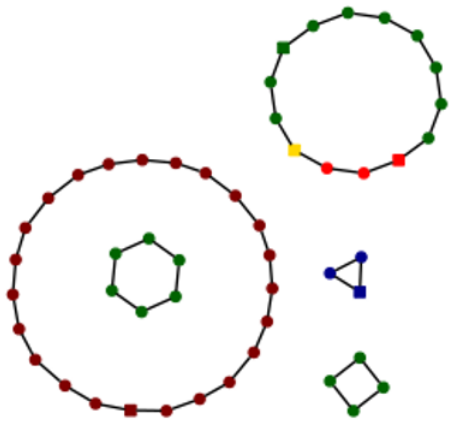
De implementatie van Growth-MHV die in deze thesis gebruikt wordt[21], is bijna volledig overgenomen uit de source code van Lewis et al.[16]. Deze implementatie definieert een volgorde voor de knopen en kiest altijd de eerste P- en LH-knoop, en de laatste LU of LF knoop. Samen met de graaf-generator van Lewis et al.[17], die de knopen sorteert volgens dalende graad, zorgt dit voor een zekere bias bij de selectie. Knopen met hoge graad worden eerst Happy gemaakt, en indien dit niet lukt worden knopen met lage graad eerst Unhappy gemaakt.

Lewis et al. geven aan dat de keuze voor deze specifieke selectiemethode gemaakt is omdat dit tot een snellere uitvoeringstijd leidt, aangezien er in beide eerste gevallen meer knopen worden gekleurd wanneer de graad van de gekozen knoop hoger is. Om te onderzoeken hoe goed de resultaten van deze methode zijn, is er een extra selectiemethode geïmplementeerd die de knoop kiest aan de hand van een kansverdeling over de knopen gesorteerd volgens afnemende graad. De gebruikte kansverdeling heeft een lineaire kansdichtheid, en wordt beschreven in appendix A. Deze laat toe om de selector een bias α te geven voor knopen met hogere graad, zodat het gedrag tussen beide extremen kan liggen, namelijk tussen altijd de hoogste graad uitkiezen ($\alpha = -2$), wat overeenkomt met de methode van Lewis et al., en altijd de laagste graad uitkiezen ($\alpha = 2$). $\alpha = 0$ komt overeen met een uniforme kansverdeling voor alle knopen.

In sectie 9.3 wordt een analyse gemaakt van het gedrag van beide selectiemethodes, om na te gaan of de selectiemethode van Lewis et al. invloed heeft op de kwaliteit van de oplossing, en of een bepaalde instelling van de bias α leidt tot eventueel betere resultaten.

6.2 Exact algoritme voor 2-reguliere grafen

Bij het evalueren van beide constructieve algoritmen op verschillende types grafen, bleek dat 2-reguliere grafen bijna nooit optimaal worden opgelost, ondanks dat dit eenvoudige problemen zijn. Aangezien de exacte solver ook heel snel de optimale oplossing vindt voor deze gevallen, leek het aangewezen om dit specifieke geval verder te onderzoeken.



Figuur 6.1: Voorbeeld van een 2-reguliere graaf. Vierkante knopen zijn voorgekleurd.

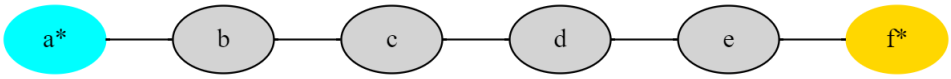
2-reguliere grafen, of in het algemeen geval d-reguliere grafen, zijn speciale grafen waarbij elke knoop dezelfde graad d heeft. Dit betekent dat 2-reguliere grafen bestaan uit circulaire kettingen van knopen, figuur 6.1 toont een voorbeeld. Een graaf bestaat dus uit één of meerdere kringen van knopen, die elk afzonderlijk kunnen behandeld worden.

6.2.1 Algoritme

Het algoritme dat volgt is gebaseerd op het labeling-systeem dat gebruikt wordt bij het Growth-MHV algoritme, maar gebruik maakt van groepen van knopen. Het belangrijkste concept gebruikt in het algoritme is dat de kleur van alle knopen tussen twee voorgekleurde knopen gelijk kan zijn. Er zijn namelijk twee mogelijkheden: ofwel hebben beide voorgekleurde knopen dezelfde kleur, en is de kleur van de knopen ertussen eenvoudig te bepalen, ofwel verschillen deze van kleur. In dat geval zal er altijd ergens in de volledige groep een overgang van de ene kleur naar de andere zijn. Een belangrijk inzicht is dat het niet uitmaakt waar deze grens ligt voor het aantal Happy knopen in de graaf, aangezien het altijd mogelijk is om hetzelfde aantal knopen Happy te maken.

Figuur 6.2 toont hoe zo een groep er zou kunnen uitzien. Voor elk van de vier knopen moet bepaald worden of ze de kleur van de linkse of de rechtse grens-knoop moeten krijgen. In deze groep is het altijd mogelijk om drie interne knopen Happy

te maken en alle vier Happy maken is onmogelijk. Dit geldt ook algemeen, een groep met n knopen en twee grens-knopen met een verschillende kleur bevat altijd $n-1$ Happy knopen. Dit betekent dat de Happiness van de totale graaf eigenlijk niet afhangt van de interne kleuring van deze groepen, aangezien het triviaal is om deze maximaal te kleuren. De Happiness van de voorgekleurde knopen, die elk grenzen aan twee groepen, zal de optimaliteit van de kleuring bepalen.



Figuur 6.2: Een groep van knopen in een 2-reguliere graaf, begrensd door 2 voorgekleurde knopen.

De eerste stap in het algoritme transformeert de graaf in een set van groepen van vrije knopen. Elk van deze groepen heeft een linkse en een rechtse grens-knoop, en deze bepalen of de groep de kleur van de linkse of de rechtse grens-knoop moet krijgen. Voor beide grens-knopen zijn er dan drie mogelijkheden, gebaseerd op de kleur van de knoop waarmee ze verbonden zijn buiten de groep. De eerste mogelijkheid is dat deze knoop nog geen kleur heeft, in dat geval is er nog geen beperking voor de grens-knoop. Als deze knoop wel al gekleurd is, dan heeft deze ofwel dezelfde kleur als de grens-knoop en kan de grens-knoop Happy gemaakt worden, ofwel verschillen beide kleuren, is de grens-knoop sowieso Unhappy.

Status linkse grens-knoop	**Status rechtse grens-knoop**	**Kleur van de groep**
Onbeperkt	Onbeperkt	?
Potentieel Happy	Onbeperkt	Links
Unhappy	Onbeperkt	Rechts
Onbeperkt	Potentieel Happy	Rechts
Potentieel Happy	Potentieel Happy	Links/Rechts
Unhappy	Potentieel Happy	Rechts
Onbeperkt	Unhappy	Links
Potentieel Happy	Unhappy	Links
Unhappy	Unhappy	Links/Rechts

Tabel 6.1: Beslissingstabel voor de verschillende mogelijke gevallen voor een groep.

Deze drie labels, Onbeperkt (O), Potentieel Happy (PH) en Unhappy (U), en het feit dat elke groep begrensd wordt door twee grens-knopen, zorgen voor in totaal negen mogelijke gevallen voor een groep. De volledige beslissingstabel die aangeeft wat de kleur moet zijn van de groep gebaseerd op de status van de grens-knopen is te zien in tabel 6.1. Deze tabel kan kort worden samengevat met de volgende regels: kies de kant met status PH, en als er een status U is, kies dan de andere kant. Dit betekent dat knopen die niet meer Happy kunnen zijn genegeerd worden, en dat knopen die Happy kunnen zijn het ook worden. In de gevallen dat beide statussen gelijk zijn, maakt de keuze niet uit voor PH - PH en U - U. Wat maakt dat dit

algoritme de optimale kleuring berekent, is dat het geen gokken maakt voor het O - O geval. In dit geval zijn er nog geen beperkingen en is het dus onmogelijk om de optimale beslissing te nemen.

Algoritme 6.2 geeft een overzicht van het volledige exact algoritme. Hier is te zien dat als beide grens-knopen van een groep status O hebben, deze groep voorlopig overgeslagen wordt. De *Last* variabele onthoudt wanneer er laatst een kleur is toegekend, om te vermijden dat het algoritme vast zou komen te zitten als alle overblijvende groepen in de O - O status zitten. In dat geval moet ergens een keuze gemaakt worden, wat gebeurt op lijn 9. Het enige mogelijke geval dat niet behandeld wordt in deze pseudocode is wanneer een groep geen grens-knopen heeft. Dit komt enkel voor als er geen enkele voorgekleurde knoop in een kring van knopen ligt. De optimale oplossing voor zo een groep is alle knopen dezelfde kleur te geven, en welke maakt niet uit.

Algoritme 6.2 Exact algoritme voor 2-reguliere grafen

```
1:  Groups ← makeGroups(G)
2:  Last ← null
3:  while #Groups ≠ 0 do
4:      g ← Groups.pop()
5:      if getLeftColour(g) = getRightColour(g) then
6:          Kleur g met deze kleur
7:      end if
8:      if g = Last then
9:          Kleur g met één van beide kleuren
10:         Last ← null
11:     else if getLeftState(g) = O and getRightState(g) = O then
12:         Groups.add(g)
13:         if Last = null then
14:             Last ← g
15:         end if
16:     else
17:         Last ← null
18:         Kleur g a.d.h.v. tabel 6.1
19:     end if
20: end while
```

6.2.2 Analyse van het algoritme

Het opstellen van alle groepen in een graaf, komt neer op een variant van BFS, en controleert voor elke knoop telkens beide verbindingen. Het aantal groepen dat gegenereerd wordt, is kleiner dan of gelijk aan het aantal voorgekleurde knopen p in de graaf. Dit kan eenvoudig begrepen worden door iteratief voorgekleurde knopen toe te voegen aan een graaf. Elke toevoeging verdeelt ofwel een bestaande groep in

twee, of komt juist op een grens tussen twee bestaande groepen te liggen en voegt dus geen groep toe.

In het slechtste geval, zijn er p groepen die allemaal status O - O hebben. Dit betekent dat alle groepen een eerste keer worden overlopen voordat de eerste kleur wordt toegekend in iteratie $p+1$. In het slechtste geval dan weer, komt het algoritme de twee aanliggende groepen die nu niet meer status O - O hebben pas helemaal op het einde tegen, en moet het dus $p-1$ groepen overlopen in totaal. Dit kan zich blijven herhalen, zodat er in elke iteratie slechts twee groepen minder, degene die juist een kleur gekregen hebben, moeten overlopen worden. Vergelijking 6.1 toont de berekening voor het totaal aantal groep-checks in het slechtste geval. In een meer gemiddeld geval daarentegen, is het eerder zeldzaam dat een groep meer dan 2 keer overlopen wordt. Samengevat kan men zeggen dat men van het algoritme normaal lineaire complexiteit verwacht in functie van het aantal voorgekleurde knopen, en in het slechtste geval wordt dit kwadratische complexiteit.

$$p + 1 + \sum_{i=0}^{\lfloor \frac{p-1}{2} \rfloor} p - 1 - i * 2 = \mathcal{O}(p^2) \tag{6.1}$$

Nu het exact algoritme uitgelegd is, is het ook mogelijk om toe te lichten waarom beide constructieve algoritmen er niet in slagen om deze eenvoudige problemen goed op te lossen. Het is vrij duidelijk waarom het Greedy-MHV algoritme slechte oplossingen oplevert. De oplossingen die dit algoritme genereert zijn namelijk oplossingen waarin alle vrije knopen dezelfde kleur hebben. Dit zorgt al voor een probleem in de stelling dat er in een groep altijd $n-1$ Happy knopen zijn, want de knopen zouden een kleur kunnen krijgen die verschilt van de kleur van beide grens-knopen.

Growth-MHV daarentegen gebruikt een techniek zeer gelijkaardig aan degene die gebruikt wordt in het exacte algoritme, maar gaat te snel kleuren kiezen. Een grens-knoop die status O heeft, komt overeen met een P-knoop in het Growth-MHV algoritme. Dit betekent dat deze knopen zo snel mogelijk Happy worden gemaakt, ook als deze in de uiteindelijke oplossing niet Happy zijn. Dit is een duidelijk voorbeeld van een greedy heuristiek die te snel 'goede' keuzes wil maken, en bijgevolg in een sub-optimale oplossing terecht komt. Dit wordt vermeden in het exacte algoritme door die groepen over te slaan, maar dit kan leiden tot een grotere complexiteit zoals hierboven aangegeven.

Hoofdstuk 7

Metaheuristisch algoritme

In dit hoofdstuk wordt een metaheuristisch algoritme voorgesteld om het MHV probleem op te lossen. In sectie 7.1 wordt het gebruikte framework, Simulated Annealing, algemeen uitgelegd. Om dit framework in de praktijk te kunnen gebruiken, moeten alle componenten van het algoritme ingevuld worden met eventueel probleemspecifieke operatoren. Alle in deze thesis gebruikte operatoren worden uiteengezet in sectie 7.2. De source code van deze thesis[21] voorziet een implementatie van het algoritme dat hier beschreven wordt, met alle beschreven operatoren voor elke component.

7.1 Simulated Annealing

Het metaheuristisch framework waarop het volledige algoritme is gebaseerd, is Simulated Annealing (SA). Dit is één van de oudste metaheuristische technieken om het globale optimum van een optimalisatieprobleem te berekenen. In 1983 hebben Kirkpatrick et al.[14] deze techniek gebruikt in een algoritme om het Travelling Salesman probleem op te lossen. Zij hebben deze techniek een naam gegeven, die geïnspireerd is op het annealing-proces dat gebruikt wordt om de kristalstructuur van materialen te bewerken.

SA is een iteratief proces, dat in elke stap de huidige energie van de oplossing probeert te verlagen door stochastische veranderingen aan te brengen in de oplossing. De pseudocode in algoritme 7.1 geeft een algemeen overzicht van het volledige proces. Dit proces begint met een initiële oplossing, die heel slecht kan zijn, en een begintemperatuur. Deze temperatuur bepaalt net zoals bij het fysische proces hoe groot mogelijke veranderingen zijn. Bij hoge temperatuur is veel energie beschikbaar om veranderingen te realiseren, en bij lage temperatuur zijn enkel kleine stappen mogelijk. Het iteratieve proces bestaat uit het zoeken naar een buur-oplossing van de huidige oplossing, die dan de nieuwe oplossing kan worden afhankelijk van de energie van beide oplossingen, en de huidige temperatuur.

Algoritme 7.1 Simulated Annealing

1: $s \leftarrow s_I$ {Een initiële oplossing}
2: $T \leftarrow T_I$
3: **while** stopconditie niet voldaan **do**
4: $s_{new} \leftarrow genereerBuur(s)$
5: **if** $kans(E(s), E(s_{new}), T) > random(0, 1)$ **then**
6: $s \leftarrow s_{new}$
7: **end if**
8: $T \leftarrow koelTemperatuur(T)$
9: **end while**
10: Resultaat: s

Dit is zoals eerder al vermeld slechts een framework voor metaheuristische algoritmen. Elk van de componenten die hierboven en in de pseudocode vermeld worden, zoals de buur-generatie en de acceptatie-kans, moeten voor het specifieke probleem op maat gemaakt worden. Deze componenten worden in de volgende sectie één voor één uitgebreid uitgelegd, samen met de gebruikte of ontworpen operatoren voor het MHV probleem.

7.2 Componenten

Deze sectie definieert voor elke abstracte component van algoritme 7.1 één of meerdere operatoren die samen het metaheuristisch algoritme ontworpen in deze thesis voorstellen. Voor de componenten waarvoor verschillende operatoren worden voorgesteld, wordt aangegeven of deze tegelijk worden gebruikt, of dat de keuze van de operator een extra parameter van het algoritme is.

7.2.1 Initiële oplossing

Het hele SA proces bestaat uit het zoeken naar buren van de huidige oplossing. Dit betekent dat er een initiële oplossing moet zijn waarvan het zoekproces vertrekt. Zolang het een volledige oplossing is, zonder niet-ingevulde variabelen, kan elke oplossing hiervoor gebruikt worden. Typisch wordt hiervoor ofwel een random oplossing gebruikt, ofwel de oplossing van een ander (snel) algoritme.

In totaal werden vier verschillende operatoren bekeken in deze thesis. De eerste is een volledig random oplossing, waarin voor elke vrije knoop random een kleur wordt gekozen. De volgende twee operatoren zijn de twee constructieve algoritmen van Li en Zhang., beschreven in sectie 6.1, Greedy-MHV en Growth-MHV. De laatste operator maakt ook gebruik van deze constructieve algoritmen, maar kiest de beste oplossing van de twee.

Er zijn experimenten uitgevoerd met alle vier operatoren, en hieruit bleek dat de laatse operator, die de beste constructieve oplossing kiest, de beste resultaten

oplevert. Dit bleek ook het resultaat te zijn van de automatische tuner (zie sectie 8.3) die gebruikt is om het algoritme te tunen, aangezien de keuze van operator een parameter kan zijn in het hele algoritme.

7.2.2 Stop-conditie

De stop-conditie in het SA proces bepaalt wanneer het iteratieve zoek-proces moet stoppen. Dit kan zowel doordat de huidige oplossing goed genoeg is, alsook wanneer een bepaald budget is opgebruikt. Voor het MHV algoritme zijn twee verschillende stop-condities geïmplementeerd, die beide het proces limiteren tot een bepaald budget. Dit is ofwel een maximum aantal iteraties, ofwel een maximaal tijdsbudget.

7.2.3 Energie-functie

Het doel van SA is het vinden van de oplossing met de laagste energie. De term energie is overgenomen uit het fysische proces, waarbij het materiaal wordt afgekoeld tot een situatie met zo weinig mogelijk interne spanningen, en dus lage entropie. Het is de taak van de energie-functie om voor elke oplossing een getal te genereren dat de kwaliteit ervan voorstelt. Dit kan gewoon de doelfunctie van het optimalisatie-probleem zijn, of de negatie hiervan als de doelfunctie gemaximaliseerd moet worden.

De energie-functie die gebruikt wordt in dit metaheuristisch algoritme is gelijk aan het aantal Unhappy knopen in de huidige oplossing. Dit komt overeen met de doelfunctie van MHV, aangezien dit gelijk is aan het aantal knopen min het aantal Happy knopen.

7.2.4 Acceptatiekans

De acceptatiekans $P(e_1, e_2, T)$ bepaalt de kans dat een oplossing wordt aangenomen als de nieuwe oplossing in het SA proces. Deze functie gebruikt zowel de energie van beide oplossingen, als de huidige temperatuur. Eén van de belangrijkste concepten van SA is dat de kans dat een oplossing die slechter is dan de huidige wordt aangenomen groter dan 0 moet kunnen zijn. Dit laat toe dat het zoek-proces uit lokale optima weggeraakt, en zo het globale optimum kan vinden. De temperatuur T bepaalt hoeveel slechter e_2 kan zijn voordat de kans 0 wordt. Een andere belangrijke voorwaarde is dat als de temperatuur naar 0 gaat, het zoekgedrag verandert naar greedy-search, waarbij enkel betere oplossingen worden aangenomen. Dit zorgt ervoor dat op lage temperatuur enkel nog gezocht kan worden naar het lokale optimum.

De gebruikte functie wordt voorgesteld in vergelijking 7.1. Een betere oplossing wordt altijd aangenomen, en als de temperatuur (bijna) 0 is, wordt de deling door 0 vermeden door de tweede regel. De laatste regel definieert de kans aan de hand van een formule die zeer gelijkaardig is aan de Boltzmann-factor, die de verhouding

tussen de voorkomenskans van twee energie-statussen berekent aan de hand van hun energie en de temperatuur.

$$P(e_1, e_2, T) = \begin{cases} 1 & e_2 < e_1 \\ 0 & e_2 \geq e_1 \& T < 0.1 \\ \exp\left(-\frac{e_2 - e_1}{T}\right) & anders \end{cases} \tag{7.1}$$

7.2.5 Koelschema

De temperatuur in het SA proces bepaalt samen met de acceptatiekans het zoekgedrag van het totale algoritme. Typisch daalt de temperatuur tijdens het iteratieve proces, zodat in het begin grote stappen kunnen gezet worden in de zoekruimte, en later meer gefocust wordt op het zoeken van de beste lokale oplossing. Vaak houdt het koelschema in dat de temperatuur enkel daalt, vandaar ook de naam, maar dit is niet nodig om het algoritme te laten werken. Wat wel aangewezen is, is dat de temperatuur op het einde van het proces voor verschillende iteraties 0 is, zodat de greedy-search eigenschap van de acceptatiekans kan garanderen dat de finale oplossing een lokaal optimum is.

Het koelschema in dit metaheuristisch algoritme laat de temperatuur lineair dalen, zodat de laatste $\alpha * i_{max}$ van de iteraties temperatuur 0 hebben. Dit betekent dat de temperatuur kan geschreven worden in functie van ofwel het aantal iteraties, ofwel de reeds verlopen tijd, afhankelijk van het gebruikte stop-criterium. Deze functie wordt weergegeven in vergelijking 7.2. Hierin is i ofwel het aantal iteraties dat al uitgevoerd werd, ofwel de reeds verstreken tijd, en i_{max} komt overeen met de limiet hierop. α is een parameter van het finale algoritme, die bepaalt welk percentage van de iteraties zullen verlopen met temperatuur 0.

$$temperatuur(i) = \begin{cases} T_i * \left(1 - \frac{i}{(1-\alpha) * i_{max}}\right) & i \leq (1-\alpha) * i_{max} \\ 0 & i > (1-\alpha) * i_{max} \end{cases} \tag{7.2}$$

7.2.6 Buur-generatie

Het genereren van een buur van de huidige oplossing is de belangrijkste individuele component van het volledige algoritme. De buur van een oplossing, is een oplossing die bekomen wordt door een (kleine) verandering aan te brengen aan die oplossing. Het zoek-gedrag van het hele algoritme wordt grotendeels bepaald door hoe de buren gegenereerd worden.

Waar er bij de vorige componenten al een voor de hand liggende of standaard operator gebruikt is, was dit niet echt mogelijk voor de buur-generatie. Hiervoor is er geëxperimenteerd met verschillende speciaal voor MHV ontworpen operatoren. Het volledige algoritme kiest in elke iteratie één van de drie onderstaande operatoren,

de graad-gebaseerde, Merge- of Split-operator, volgens een verhouding en laat de gekozen operator de buur genereren. Deze verhouding ligt voorlopig vast tijdens het verloop van het algoritme, en wordt ingesteld als een parameter van het algoritme.

Graad-gebaseerde operator

De meest voor de hand liggende operator, genereert een buur door de kleur van een knoop te veranderen. Dit is een zeer eenvoudige operatie, maar het blijkt dat deze operator op zich in de praktijk niet sterk genoeg is om goede oplossingen te bekomen. Een oplossing heeft onder deze operator $(n-p)*(k-1)$ buren, elke vrije knoop (dit zijn $(n-p)$ knopen, p is het aantal voorgekleurde knopen) kan namelijk veranderen naar $k-1$ andere kleuren. De optimale oplossing kan bereikt worden met $n-p$ stappen vanuit elke initiële oplossing, door elke vrije knoop te veranderen naar zijn optimale kleur.

Deze operator alleen slaagt er niet in om goede oplossingen te bekomen, maar kan wel gebruikt worden als aanvulling op beide andere operatoren in deze sectie. De operator is ook verder verfijnd, om betere resultaten te bekomen. Een eerste aanpassing is het veranderen van kleur voor een knoop te beperken tot de kleur van één van zijn buren. Dit is interessant omdat als de knoop een kleur zou krijgen verschillend van al zijn buren, al die buren sowieso Unhappy zullen zijn. Aangezien het de bedoeling van deze operator is om kleine veranderingen aan te brengen in de huidige oplossing, gaat die enkel niet-aanliggende kleuren toewijzen wanneer de gekozen knoop alleen met knopen met éénzelfde kleur verbonden is.

Een tweede aanpassing is mogelijk bij het uitkiezen van de knoop die van kleur gaat veranderen. Knopen met hoge graad hebben meer impact op de kwaliteit van de oplossing, dus kan het interessant zijn om knopen met hoge graad een hogere kans te geven om van kleur te veranderen. Dit is geïmplementeerd aan de hand van een roulette-wiel selectie-proces dat gebruik maakt van de graad van elke knoop, zodat de selectiekans voor elke knoop evenredig is met zijn graad. Dit sluit direct ook knopen met graad 0 uit van de selectie, want die hebben geen enkele invloed op de kwaliteit van de oplossing.

Zoals reeds vermeld behaalt deze operator op zichzelf geen goede resultaten, waarschijnlijk omdat er niet genoeg rekening gehouden wordt met de huidige staat van Happiness in de graaf. Daarom wordt deze operator gebruikt als een aanvulling op de twee operatoren hieronder, die beide complexere operatoren zijn.

Merge-operator

De Merge-operator probeert de structuur in de huidige oplossing deels te behouden, door niet één knoop van kleur te veranderen, maar door een hele groep van kleur te veranderen. Deze operator redeneert niet over knopen, maar over gekleurde groepen,

een subset van geconnecteerde knopen van de graaf, die dezelfde kleur hebben.

De eerste stap voor deze operator stelt de gekleurde groepen op van een graaf. Dit wordt gedaan met een eenvoudig zoekalgoritme, weergegeven in algoritme 7.2. Dit algoritme berekent de set van gekleurde groepen die aan minstens één knoop met een andere kleur grenzen. Groepen zonder anders-gekleurde buren zijn niet relevant om te bekijken, omdat er geen betere oplossing mogelijk is voor die groep.

Algoritme 7.2 Opstellen gekleurde groepen

```
1:  Groups ← {}
2:  ∀v ∈ V : Done[v] = false
3:  for all v ∈ V do
4:    if !isPrecoloured(v) and !Done[v] then
5:      G ← {v}
6:      Colours ← {}
7:      Queue ← {v}
8:      while #Queue > 0 do
9:        v' ← Queue.pop()
10:       for all a ∈ Γ(v') do
11:         if colour(a)! = colour(v) then
12:           Colours ← Colours ∪ {colour(a)}
13:         else if isPrecoloured(a) then
14:           Done[a] = true
15:         else if !Done[a] then
16:           G ← G ∪ {a}
17:           Queue.add(a)
18:           Done[a] = true
19:         end if
20:       end for
21:     end while
22:     if #Colours > 0 then
23:       Groups ← Groups ∪ {(G, Colours)}
24:     end if
25:   else
26:     Done[v] = true
27:   end if
28: end for
29: Resultaat: Groups
```

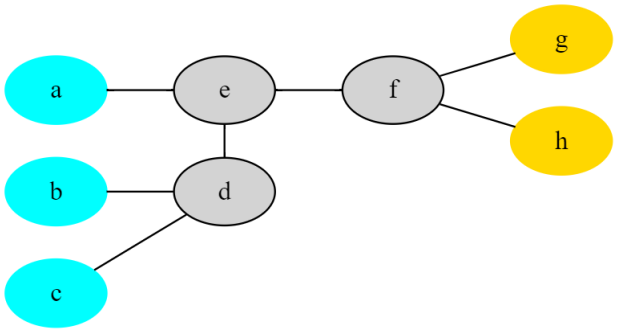
Algoritme 7.2 gebruikt een extra array om aan te duiden welke knopen al behandeld zijn, en deze wordt geïnitialiseerd op volledig $false$. Op lijn 3 begint de loop over alle knopen van de graaf, die itereert totdat een knoop gevonden wordt die niet voorgekleurd is, en nog niet behandeld is. Deze knoop is dan een deel van een nieuwe groep, die wordt aangemaakt op lijn 5. Lijn 6 en 7 maken de extra

sets klaar die nodig zijn voor het opbouwen van de nieuwe groep. Het volgende onderdeel van het algoritme werkt door in elke iteratie een knoop uit de huidige *Queue* te halen (lijn 9), en alle verbonden knopen te controleren. Als een verbonden knoop een andere kleur heeft dan de groep, dan wordt deze kleur toegevoegd aan de set van aangrenzende kleuren (lijn 12). Als de verbonden knoop nog niet behandeld is en dus dezelfde kleur heeft als de groep, dan wordt deze toegevoegd aan de groep en de *Queue*, zodat ook de aangrenzende knopen kunnen gecontroleerd worden.

De Merge-operator krijgt zijn naam omdat deze uniform een groep uitkiest uit deze lijst van groepen, en deze één van zijn aanliggende kleuren geeft. Dit zorgt ervoor dat deze groep eventueel samengaat met aanliggende groepen die deze kleur hebben. Op deze manier gaat het aantal groepen wanneer enkel deze operator gebruikt wordt ofwel constant blijven, ofwel dalen. Vandaar is het nuttig om deze operator naast de twee andere buur-operatoren te gebruiken, omdat deze beide het aantal groepen kunnen verhogen.

Split-operator

De split-operator is ontworpen als aanvulling op de Merge-operator, en behaalt op zichzelf geen goede resultaten. Net zoals de Merge-operator gebruikt deze operator gekleurde groepen, maar in plaats van groepen samen te voegen, splitst deze operator een groep in twee. Dit kan nodig zijn in verschillende situaties, bijvoorbeeld wanneer de Merge-operator te veel knopen heeft samengenomen in dezelfde groep.



Figuur 7.1: Voorbeeld van een gekleurde groep in een graaf die opgesplitst kan worden.

Figuur 7.1 toont een situatie waarbij het interessant kan zijn om de groep (de niet-gekleurde knopen d, e en f) te splitsen in twee delen. De situatie waarbij e en d blauw zijn en f geel is, zorgt waarschijnlijk voor een hoger aantal Happy knopen in de volledige graaf. Dit is niet noodzakelijk zo, aangezien de Happiness van de rand-knopen ook door de rest van de graaf wordt beïnvloed, wat in de figuur niet wordt weergegeven.

De optimale splitsing berekenen is algemeen gezien niet triviaal, aangezien dit bijna overeenkomt met het originele MHV probleem maar dan in de kleinere groep. In het algemeen komt dit overeen met een versie van het Multiterminal Cut probleem[5], waarbij gezocht wordt naar de zo klein mogelijke groep van bogen die er voor zorgt dat er geen twee terminal-knopen in dezelfde component zitten. De terminal-knopen in dit geval zijn de rand-knopen van een groep die worden samengenomen tot één knoop per kleur. Het Multiterminal Cut probleem is in het algemeen NP-hard zolang er drie of meer terminals zijn, wat maakt dat het splitsingsprobleem hier zeker niet eenvoudig op te lossen is.

Het gebruikte algoritme om een groep op te splitsen is bijgevolg een benadering van het optimale geval, enerzijds omdat de optimale splitsing niet eenvoudig te berekenen is, maar ook omdat deze optimale splitsing geen enkele garantie geeft over de Happiness van deze groep, en bij uitbreiding de hele graaf. Ook om deze redenen gebruikt het algoritme random elementen bij het opstellen van de splitsing.

De eerste stap in het algoritme, weergegeven in algoritme 7.3 a.d.h.v. set-notatie, kiest twee knopen uit de groep, die elk verbonden zijn met een knoop die een andere kleur heeft dan de groep. In het algoritme worden deze de linkse en de rechtse seed-knoop genoemd (lijn 1 en 2). Deze knopen worden als de twee startpunten van het splits-algoritme gebruikt. Dit verdeelt alle knopen van de groep in twee groepen, en is gebaseerd op de werking van Krager's algoritme[13], een random algoritme dat de minimum Cut van een graaf benadert. In het algoritme worden twee extra sets bijgehouden. Deze sets houden bij welke knopen uit de originele groep direct verbonden zijn met de knopen in de twee nieuwe groepen die worden opgebouwd. Deze sets worden geïnitialiseerd op lijn 3 en 4 in het algoritme met alle knopen die met de seed-knopen verbonden zijn, en die in de groep liggen.

Zolang er nog knopen zijn die in geen van beide groepen zitten, worden er iteratief knopen toegevoegd aan beide groepen. Als er geen niet-toegewezen knopen meer zijn die verbonden zijn met de linkse groep, worden alle nog niet toegewezen knopen aan de rechtse groep toegewezen (lijn 6-8). Hetzelfde geldt analoog voor het omgekeerde geval, waar dat er geen niet-toegewezen knopen meer verbonden zijn met de rechtse groep (lijn 9-11).

Als beide extra sets daarentegen niet leeg zijn, wordt er uniform een knoop gekozen uit de unie van deze sets (lijn 13). De gekozen knoop wordt toegevoegd aan de groep die overeenkomt met de extra set waar deze momenteel in zit (lijn 15/19). Nu dat een extra knoop is toegewezen, moet de set van verbonden knopen van die groep worden bijgewerkt. Dit gebeurt door alle knopen verbonden met de gekozen knoop, die in de groep zitten, toe te voegen aan de set, en alle reeds toegewezen knopen hieruit te verwijderen (lijn 16/20). De set van verbonden knopen van de andere groep moet ook worden bijgewerkt, door de gekozen knoop hieruit te verwijderen (lijn 17/21).

Algoritme 7.3 Splits-algoritme

1: $LeftGroup \leftarrow \{leftSeed\}$
2: $RightGroup \leftarrow \{rightSeed\}$
3: $LeftAdj \leftarrow \Gamma(leftSeed) \cap Group$
4: $RightAdj \leftarrow \Gamma(rightSeed) \cap Group$
5: **while** $\#(LeftAdj \cup RightAdj) > 0$ **do**
6: **if** $\#LeftAdj = 0$ **then**
7: $RightGroup \leftarrow Group \backslash LeftGroup$
8: $RightAdj \leftarrow \{\}$
9: **else if** $\#RightAdj = 0$ **then**
10: $LeftGroup \leftarrow Group \backslash RightGroup$
11: $LeftAdj \leftarrow \{\}$
12: **else**
13: $picked \leftarrow$ kies uniform een knoop uit $LeftAdj \cup RightAdj$
14: **if** $Picked \in LeftAdj$ **then**
15: $LeftGroup \leftarrow LeftGroup \cup \{picked\}$
16: $LeftAdj \leftarrow (LeftAdj \cup \Gamma(picked) \cap Group) \backslash (LeftGroup \cup RightGroup)$
17: $RightAdj \leftarrow RightAdj \backslash \{picked\}$
18: **else**
19: $RightGroup \leftarrow RightGroup \cup \{picked\}$
20: $RightAdj \leftarrow (RightAdj \cup \Gamma(picked) \cap Group) \backslash (LeftGroup \cup RightGroup)$
21: $LeftAdj \leftarrow LeftAdj \backslash \{picked\}$
22: **end if**
23: **end if**
24: **end while**
25: Resultaat: $LeftGroup$, $RightGroup$

Nu de groep opgesplitst is in twee kleinere groepen, moeten beide groepen een verschillende kleur krijgen. Dit wordt bekomen door voor beide groepen random een kleur te kiezen uit de kleuren waarmee ze verbonden zijn.

7.3 Besluit

Het metaheuristisch algoritme voorgesteld in dit hoofdstuk maakt gebruik van het Simulated Annealing framework, een techniek om het globale optimum van een optimalisatie-probleem te benaderen. Dit gebeurt door in elke stap een buur van de huidige oplossing te berekenen, aan de hand van één van de drie in deze thesis ontwikkelde buur-operatoren. Deze buur-oplossing kan dan de nieuwe oplossing worden met een kans gerelateerd aan de energie ervan, zijnde het aantal Unhappy knopen. Deze relatie is temperatuurafhankelijk, wat betekent dat er een temperatuur wordt bijgehouden in het proces die bepaalt hoe groot de kans is dat een oplossing met hogere energie toch wordt aangenomen. De temperatuur daalt lineair tijdens het

zoek-proces, wat ervoor zorgt dat het zoek-gedrag evolueert van (zeer) exploratief naar een strikte greedy local search.

De belangrijkste bijdrage van dit werk in het kader van het oplossen van het MHV probleem, bestaat uit de drie probleemspecifieke operatoren die buren genereren van oplossingen. Elk van deze drie operatoren zijn speciaal voor het MHV probleem ontwikkeld, en proberen aan de hand van de structuur in de huidige oplossing een buur-oplossing met gelijkaardige structuur te bouwen.

Het volledige algoritme heeft verschillende parameters die moeten afgesteld worden, weergegeven in tabel 7.1. Dit zijn zowel numerieke parameters, zoals de initiële temperatuur en de verhouding tussen de verschillende buur-operatoren, alsook de keuze tussen verschillende operatoren voor de initiële oplossing. Hoe deze getuned zijn voor de verschillende experimenten wordt uitgelegd in sectie 8.3, samen met de gebruikte waarden voor de parameters. Hoe goed de prestaties van dit algoritme zijn, en hoe het algoritme zich verhoudt tot het Tabu-search algoritme beschreven in [25], wordt uiteengezet in sectie 9.4.

Parameter	**Beschrijving**	**Mogelijke waarden**
$init$	Initiële oplossing	random, greedy, growth, best
i_{max}	Maximaal aantal iteraties/tijdsbuget	$\mathbb{N}$
T_i	Initiële temperatuur	$\mathbb{R}_0^+$
α	Deel van iteraties met temperatuur 0	$[0, 1]$
r_{swap} r_{merge} r_{split}	Verhouding van buur-operatoren	$[0, 1]$ $r_{swap} + r_{merge} + r_{split} = 1$

Tabel 7.1: Alle parameters van het metaheuristisch algorimte.

Hoofdstuk 8

Opstelling van de experimenten

Dit hoofdstuk beschrijft hoe alle experimenten die leiden tot de resultaten in hoofdstuk 9 opgesteld zijn. Sectie 8.1 beschrijft hoe de probleeminstanties worden gegenereerd. Hierin worden eerst de drie graaf generatoren die ook gebruikt worden in de paper van Lewis et al.[19] beschreven, en vervolgens ook een nieuwe generator die gebruik maakt van de lineaire kansverdeling beschreven in appendix A.

Sectie 8.2 beschrijft welke delen van de instantie-ruimte worden beschouwd in deze thesis, en hoe deze beslissingen genomen zijn. In sectie 8.3 wordt uiteengezet hoe en waarom een automatische tuner gebruikt is om het metaheuristisch algoritme af te regelen. Tot slot geeft tabel 8.3 een overzicht van de waarden gebruikt voor alle parameters van het metaheuristisch algoritme voor alle experimenten.

8.1 Aanmaken van MHV-instanties

Het aanmaken van MHV-instanties bestaat uit twee delen, eerst moet een graaf aangemaakt worden, en daarna moeten knopen op deze graaf worden voorgekleurd. Het aanmaken van random grafen is al uitgebreid beschreven in literatuur, aangezien grafen de basis vormen van verschillende uiteenliggende problemen. Het is mogelijk om elk soort graaf te gebruiken voor MHV-instanties, hier worden vier verschillende graafgeneratoren besproken.

8.1.1 Generatoren van Lewis et al.

In de paper van Lewis et al.[19] worden twee verschillende manieren beschreven om grafen te genereren: random grafen en Scale-free grafen. Random grafen worden aangemaakt aan de hand van het Erdös-Rényi model[7]. Dit model maakt een graaf door in een set van n niet-verbonden knopen een boog te plaatsen met een vaste onafhankelijke kans q tussen elk paar knopen. Dit betekent dat gemiddeld $\frac{n(n-1)}{2}q$ bogen in de graaf aanwezig zijn, en dat de graden van de knopen in de graaf binomiaal verdeeld zijn met gemiddelde $(n-1)q$ en variantie $(n-1)q(1-q)$. Wat interessant

is om op te merken is dat de grafen gegenereerd aan de hand van deze methode niet noodzakelijk geconnecteerd zijn. Het is dus mogelijk dat er geïsoleerde knopen zijn in de graaf, en/of dat er meerdere componenten zijn in de graaf, wat mogelijk maakt om de uiteindelijke instantie te reduceren.

Scale-free grafen worden aangemaakt aan de hand van het Barabási-Albert model[2], dat iteratief knopen toevoegt aan de graaf totdat er n knopen zijn in totaal. Elk van deze toegevoegde knopen wordt met s andere knopen verbonden, waarbij er een grotere kans is om te verbinden met knopen met een hoge graad. De implementatie van Lewis et al.[17] begint met een volledig geconnecteerde graaf met s knopen, en hier worden dan iteratief nog $n - q$ knopen aan toegevoegd. Om de s bogen die de nieuwe knoop verbinden aan te maken, worden s knopen gekozen uit de aanwezige knopen. De kans om knoop u te kiezen en deze te verbinden met v wordt gegeven in vergelijking 8.1. Dit komt overeen met een uniforme kansverdeling die gewogen wordt met de graad van de knopen. De finale graaf heeft in totaal $\frac{q(q-1)}{2} + q(n-q)$ bogen, en de graden van de knopen zijn verdeeld volgens een machtsfunctie.

$$P(u,v) = \begin{cases} \dfrac{graad(u)}{\sum_{w \in V \setminus \Gamma(v)} graad(w)} & \{u,v\} \notin E \\ 0 & anders \end{cases} \tag{8.1}$$

De source code van Lewis et al. bevat ook een implementatie van een generator die d-reguliere grafen genereert. Dit zijn grafen waarbij alle knopen exact dezelfde graad d hebben. Deze grafen worden aangemaakt aan de hand van de Steger en Wormald methode[24]. Deze methode begint met nd punten in n groepen die initieel allemaal ongepaard zijn. Zolang een geschikt paar kan gevormd worden, waarbij geschikt betekent dat punten uit dezelfde groep niet gepaard mogen worden en er slechts één paar mag bestaan tussen twee groepen, worden twee random punten i en j gekozen uit de ongepaarde knopen. Als deze een geschikt paar vormen, wordt dit toegevoegd aan de gegenereerde paren, en worden deze beide punten verwijderd uit de ongepaarde punten. Dit wordt herhaald totdat alle punten in paren zijn opgedeeld. Aan de hand van deze paren wordt een graaf opgesteld door knopen x en y te verbinden als er een paar gevormd is dat punten bevat uit groep x en groep y. Er is een kans dat deze methode faalt om een geschikte graaf te genereren, dus dit wordt iteratief herhaald totdat een geschikte graaf wordt gegenereerd.

Het toevoegen van voorgekleurde knopen gebeurt onafhankelijk van de gebruikte techniek om de graaf aan te maken. In totaal worden p knopen voorgekleurd door eerst k random uitgekozen knopen toe te kennen aan de k verschillende kleuren zodat elke kleur minstens één keer voorkomt in de instantie. De overige $p-k$ voorgekleurde knopen worden ook random uitgekozen uit de graaf, en worden toegekend aan een random kleur. Alle random keuzes gebruiken een uniforme verdeling.

8.1.2 Lineaire verdeling generator

Uit verschillende experimenten bleek dat de gemiddelde graad van de knopen een belangrijke factor is in de moeilijkheid van een instantie, dit wordt verder uitgewerkt in de volgende sectie. De motivatie achter het maken van deze graaf-generator was dat er verschillende verdelingen van graden mogelijk zijn die dezelfde gemiddelde graad hebben. Er is bijvoorbeeld een groot verschil tussen een d-reguliere graaf waarbij alle knopen dezelfde graad d hebben, en een Scale-free graaf die ook een gemiddelde graad heeft van d.

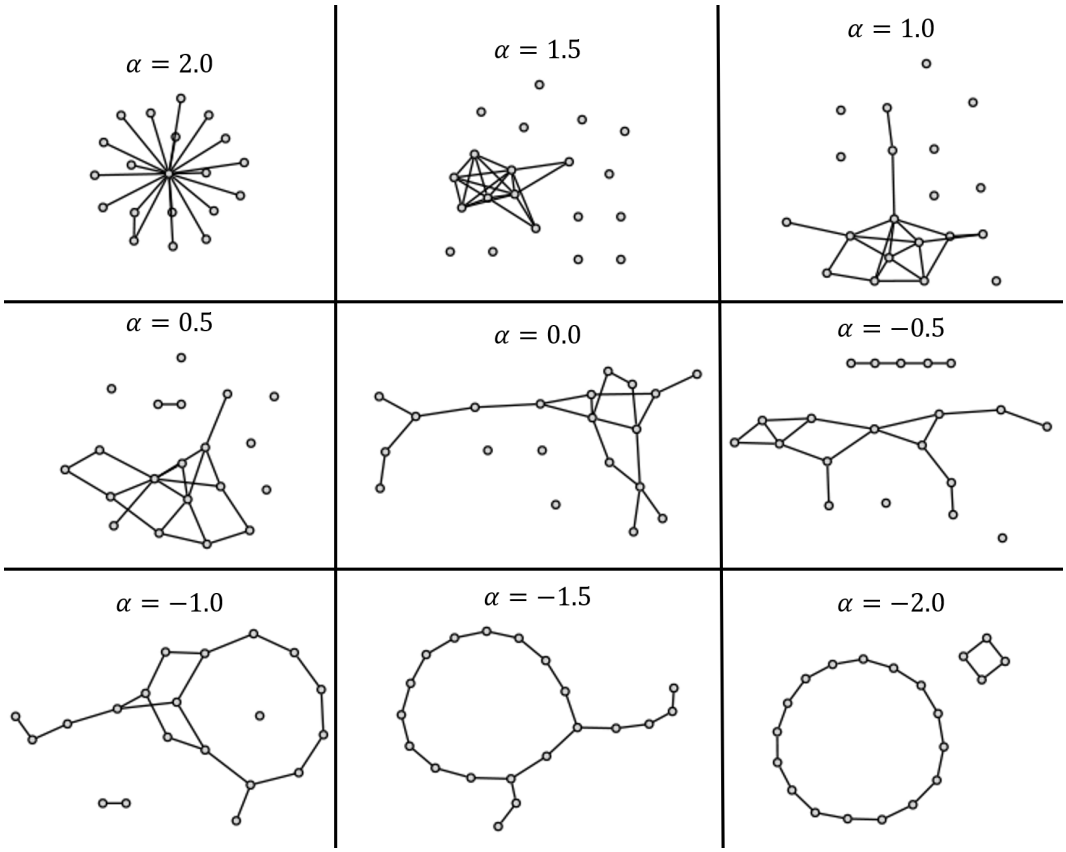
De lineaire verdeling generator maakt verschillende grafen met n knopen en gemiddelde graad d aan de hand van een extra parameter α. Deze parameter wordt gebruikt om knopen uit te kiezen gesorteerd volgens graad met de lineaire kansverdeling beschreven in appendix A.

Het proces begint met n niet-geconnecteerde knopen, en voegt iteratief $\frac{dn}{2}$ bogen toe om de gemiddelde graad d te bekomen. In elke iteratie worden twee knopen uitgekozen gebruikmakend van de lineaire kansverdeling over de knopen die nog bogen kunnen krijgen gesorteerd volgens graad. Als deze al verbonden zijn, wordt de tweede knoop v vervangen door de eerstgevonden knoop die nog niet met de eerste verbonden is. Het zoeken gebeurt door de nog niet gekozen knopen te verdelen in twee groepen, die met hogere graad dan v en die met lagere graad. De groep met lagere graad wordt overlopen van hoge graad naar lage, en de groep met hogere graad wordt overlopen van lage graad naar hoge. Afhankelijk van de waarde van α wordt bepaald welke van deze groepen eerst wordt overlopen, als $\alpha > 0$ wordt de hogere graad eerst verwerkt en als $\alpha \leq 0$ wordt de lagere graad eerst verwerkt.

Dit leidt tot grafen zoals weergegeven in figuur 8.1. Hier is te zien dat $\alpha = -2$ leidt tot d-reguliere grafen, zolang d een geheel getal is. Dit gebeurt omdat er in elke stap de twee knopen met de laagste graad met elkaar verbonden worden. Het andere extreme geval, $\alpha = 2$, zorgt voor grafen waarin alle knopen verbonden zijn met een aantal knopen met maximale graad, aangezien in elke stap de twee knopen met hoogste graad die nog niet verbonden zijn worden verbonden. De waarden van α tussenin proberen een overgang tussen beide extremen weer te geven.

8.2 Beschouwde deel van de instantie-ruimte

Om de experimenten die uitgevoerd zijn voor deze thesis te kaderen wordt in deze sectie uiteengezet welke instanties bekeken zijn. Zoals aangegeven wordt in de paper van Thiruvady et al.[25], is er een zeer groot deel van de instantie-ruimte dat bestaat uit triviale instanties, waarin geen enkele knoop Happy kan gemaakt worden omdat de gemiddelde graad te groot is, of omdat er te veel voorgekleurde knopen zijn. Omdat deze instanties niet interessant zijn om de prestaties van verschillende algoritmen te evalueren, worden deze grafen hier ook uitgesloten van de test-set.



Figuur 8.1: Grafen met $n = 20$ knopen en met een gemiddelde graad van $d = 2$ gegenereerd met de lineaire verdeling generator met verschillende waarden van α.

De ruimte met alle grafen kan beschreven worden aan de hand van verschillende kenmerken van grafen, zoals aantal knopen, densiteit, clustering... Er is een beperkt onderzoek gevoerd om te bepalen welke van deze kenmerken het best gebruikt worden om de gebruikte grafen te beschrijven. Het is mogelijk om hiernaar een uitgebreid onderzoek te doen, zoals bijvoorbeeld in [22] wordt uitgevoerd voor het Graph Colouring probleem, maar dit valt buiten het bereik van dit werk. Het graaf-kenmerk dat het best illustreert waar er verschillen in prestaties liggen in de instantie-ruimte is de gemiddelde graad van de knopen. Dit kenmerk wordt ook gebruikt in de resultaten van Thiruvady et al.[25], maar zij geven geen verklaring waarom zij dit kenmerk gebruiken.

Het interessantste gebied in de instantie-ruimte bevindt zich rond grafen met relatief lage gemiddelde graad. Dit werd ook bevestigd in eerdere experimenten. Voor de experimenten hier worden bijgevolg grafen gebruikt met gemiddelde graad tussen 0 en 25. Uit eerdere experimenten bleek ook dat het aantal kleuren k geen grote invloed heeft op de prestaties van verschillende algoritmen alsook de algemene moeilijkheid van de instantie. Hetzelfde geldt voor het aantal voorgekleurde knopen, waarbij weinig verschil op te merken valt zolang niet meer dan de helft van de

knopen wordt voorgekleurd. Bijgevolg worden de instanties aangemaakt met 10 of 50 verschillende kleuren, en tussen 5% en 25% voorgekleurde knopen.

De grafen worden aangemaakt aan de hand van de graaf-generatoren beschreven in de vorige sectie, buiten de d-reguliere generator, omdat deze grafen ook gemaakt worden door de lineaire verdeling generator. Voor de random grafen wordt een kans q gebruikt tussen 0 en $\frac{25}{n-1}$, wat overeenkomt met de gemiddelde graad gedefinieerd in de vorige paragraaf. Voor de Scale-free grafen is het minder eenvoudig om het domein van s vast te leggen, dit gebeurt aan de hand van vergelijking 8.2. Deze vergelijking volgt uit het aantal bogen dat wordt aangemaakt door de generator samen met de formule voor de gemiddelde graad van een graaf.

$$0 \leq \frac{2s(\frac{s-1}{2} + n - s)}{n} \leq 25, s \in \mathbb{N} \tag{8.2}$$

Voor een vast aantal knopen n wordt hiermee bepaald welke waarden van s gebruikt kunnen worden. Voor de lineaire verdeling generator wordt een gemiddelde graad tussen 0 en 25 gebruikt, en alle waarden van α zijn bruikbaar. Het is ook aangewezen $\alpha = -2$ op te nemen in de gebruikte configuraties, aangezien dit de d-reguliere grafen genereert.

In de experimenten worden grafen gebruikt met 1000 knopen. Dit leidt tot het het aantal configuraties weergegeven in tabel 8.1. Hier wordt meegerekend dat per graaf $2 * 3$ verschillende voorkleuringen worden gemaakt. Om random effecten uit te middelen, worden per configuratie 20 verschillende grafen gemaakt, wat ervoor zorgt dat er in totaal $3000 + 1440 + 27000 = 31440$ grafen in de test-set opgenomen zijn.

Generator	**Parameter 1**	**Parameter 2**	**# configuraties**
Random	$q \in \{1, 2 \dots 25\}$	/	150
Scale-free ($n = 1000$)	$s \in \{1, 2 \dots 12\}$	/	72
Lineaire verdeling	$d \in \{1, 2, \dots 25\}$	$\alpha \in \{x/2 \mid x \in \mathbb{Z} \cup [-4, 4]\}$	1350

Tabel 8.1: Overzicht van de gebruikte parameters voor de instantie-generatoren.

8.3 Tuning van algoritme-parameters

Omdat het afstellen van alle parameters van een (metaheuristisch) algoritme vaak even moeilijk is als het originele probleem, is het metaheuristisch algoritme beschreven in hoofdstuk 7 geconfigureerd met een automatische tuner. Het zoeken naar optimale parameter-configuraties is namelijk ook een optimalisatieprobleem, waarbij gezocht wordt in de parameter-ruimte en de te optimaliseren functie de uitvoering van een ander algoritme is.

De automatische tuner die hier gebruikt is, is *irace*[20]. Dit software-pakket maakt gebruik van de iterated racing techniek om verschillende configuraties met elkaar te vergelijken en zo de optimale configuratie te bekomen. *irace* wordt gebruikt met de standaardinstellingen en een tuning budget van 5000 algoritme-uitvoeringen om alle parameters van het algoritme, weergegeven in tabel 8.2, af te regelen. Om het algorimte te tunen is een tuning-set nodig, die best andere grafen bevat dan de test-set. De tuning-set is opgebouwd uit 150 random grafen, 1 graaf per configuratie die verschilt van de random grafen uit de test-set.

In een eerste fase werd ook de keuze voor de reductietechniek mee opgenomen in de te tunen parameters, maar nadat hieruit bleek dat de articulatie-methode altijd de beste resultaten geeft, zoals verwacht, werd besloten om deze keuze vast te leggen. Het tijdsbudget of het maximaal aantal iteraties tunen heeft weinig zin, aangezien het te verwachten is dat de tuner altijd zal kiezen voor het hoogst mogelijke budget. Daarom wordt het computationele budget vastgelegd. De gekozen waarde is 5000 iteraties, aangezien dit voor grafen met 1000 knopen altijd genoeg tijd bleek te zijn om tot een goede oplossing te komen. Het deel van de iteraties dat met temperatuur 0 verloopt, α, is vastgelegd op 5%. Dit zou ook kunnen opgenomen worden in de te tunen parameters, maar het bleek dat *irace* moeilijkheden had als deze parameter werd toegevoegd. Dit zou kunnen verklaard worden door het feit dat voor een groot deel van de probleeminstanties de metaheuristiek er niet in slaagt een verbetering te vinden op de initiële greedy oplossing. Voor elk van deze instanties is de waarde van α niet relevant, en dus kan het moeilijk zijn voor de automatische tuner om het globale optimum te vinden. De waarde van 5% is gekozen omdat uit experimenten bleek dat dit in alle bekeken gevallen genoeg iteraties voorziet om te convergeren naar het lokale optimum, en toch genoeg iteraties overlaat voor de rest van het SA proces.

Parameter	**Beschouwde waarden**
$init$	random, greedy, growth, best
T_i	$[1, 500]$
r_{swap} r_{merge} r_{split}	$[0, 1]$ $r_{swap} + r_{merge} + r_{split} = 1$

Tabel 8.2: Beschouwde waarden voor alle parameters van het metaheuristisch algorimte.

Het beschouwde domein van de overige parameters wordt weergegeven in tabel 8.2. De waarden die *irace* voor elk van deze parameters heeft bepaald, samen met de handmatig afgestelde parameters, worden weergegeven in tabel 8.3. Hierin is te zien dat een combinatie van de verschillende buur-operatoren inderdaad een beter resultaat geeft, en dat de gestructureerde operatoren, Merge en Split, belangrijker zijn dan de eerder eenvoudige swap operator bij het bekomen van een goede oplossing.

Parameter	**Getunede waarde**
reductietechniek	articulatie
$init$	best
i_{max}	5000 iteraties
T_i	211
α	0.05
r_{swap}	0.05
r_{merge}	0.42
r_{split}	0.43

Tabel 8.3: Gebruikte waarden voor alle parameters van het metaheuristisch algoritme.

Hoofdstuk 9

Resultaten

Dit hoofdstuk beschrijft alle resultaten van de verschillende experimenten die werden uitgevoerd. De algemene opstelling van de experimenten wordt uitgebreid besproken in het vorige hoofdstuk. In totaal zijn vier experimenten opgezet, om verschillende algoritmen en technieken experimenteel te analyseren en te vergelijken.

Sectie 9.1 beschrijft hoe de vergelijking gemaakt wordt tussen de reductiemethode van Thiruvady et al. en de nieuwe methodes beschreven in hoofdstuk 4. In sectie 9.2 wordt een experimentele vergelijking gemaakt tussen de twee IP-modellen voorgesteld in hoofdstuk 5. Sectie 9.3 behandelt de invloed van de selectiemethode in het Growth-MHV algoritme uit sectie 6.1 op de prestaties ervan. Vervolgens worden in sectie 9.4 de prestaties van het metaheuristisch algoritme beschreven in hoofdstuk 7 geanalyseerd. Daar wordt ook de vergelijking gemaakt met het Tabu-search algoritme ontworpen door Thiruvady et al.[25]. Tot slot worden een aantal conclusies op algemeen niveau getrokken in sectie 9.5.

Alle experimenten maken gebruik van een C++ implementatie van de verschillende algoritmen[21]. De gebruikte implementatie van beide constructieve algoritmen is grotendeels overgenomen uit de source code van Lewis et al.[16] met beperkte aanpassingen. Voor de vergelijking met het Tabu-search algoritme van Thiruvady et al. wordt de door hun ontwikkelde source code[18] gebruikt.

Alle experimenten waarvoor de uitvoeringstijd een belangrijk aspect is, zijn uitgevoerd op een PC met een Quad-core (8 logische cores)[1] Intel i7-6700K @ 4.0 GHz en 16 GB DDR4 ram. Dit heeft enkel belang voor de experimenten met de exacte solver en de vergelijking met het Tabu-search algoritme. De implementatie van Tabu-search voorziet namelijk een tijdslimiet, en deze wordt gebruikt om de vergelijking te maken met de metaheuristiek ontworpen in deze thesis.

[1] Alle implementaties gebruiken slechts één thread, buiten de exacte solver. Deze wordt gelimiteerd tot 6 threads om interferentie van andere processen te elimineren.

9.1 Vergelijking reductie-methodes

In hoofdstuk 4 worden drie verschillende methodes voorgesteld om knopen, waarvan de kleur triviaal bepaald kan worden, te verwijderen uit de probleeminstantie. De eerste methode is de reductiemethode van Thiruvady et al.[25], waarbij knopen worden verwijderd die slechts met één kleur verbonden zijn, of die enkel verbonden zijn met voorgekleurde knopen met verschillende kleuren.

De andere twee methodes zijn nieuw ontworpen voor deze thesis, en vormen een uitbreiding op de methode van Thiruvady et al. Deze maken beide gebruik van het concept van referenties, zodat als de kleur van knopen triviaal kan verbonden worden met de kleur van een andere knoop, deze knoop ook kan verwijderd worden. De eerste variant reduceert dezelfde knopen als de methode van Thiruvady en kettingen van vrije knopen. De tweede variant vertrekt van articulatie-knopen, en reduceert de componenten aan de hand van een aantal regels, en dit wordt iteratief herhaald.

Het doel van de experimenten in deze sectie bestaat uit twee delen. Het eerste deel bestaat uit een analyse van het aantal extra knopen dat wordt gereduceerd door de twee nieuwe methodes in vergelijking met de methode van Thiruvady et al. Het tweede deel analyseert het gedrag van de reductie-methodes in de instantie-ruimte, omdat dit ook een idee kan geven van de moeilijkheid van de originele instanties.

De data gebruikt voor onderstaande resultaten is verzameld door elk van de drie reductiemethodes uit te voeren op elk van de 31440 grafen beschreven in sectie 8.2 en het aantal gereduceerde knopen te noteren. De resultaten hieronder analyseren het verschil in aantal gereduceerde knopen met de methode van Thiruvady et al. Er wordt verder gesproken over types grafen, hiermee wordt bedoeld dat het gaat over grafen die zijn gegenereerd met de respectievelijke generator (zie sectie 8.1). Om plaats te besparen zullen de grafen gegenereerd met de lineaire verdeling generator aangeduid worden met het graaf-type Lineair.

9.1.1 Vergelijking verschillende methodes

Zoals te zien is in de resultaten in tabel 9.1, leiden beide methodes tot meer gereduceerde knopen. De articulatie-methode zorgt voor slechts een kleine verbetering ten opzichte van de eenvoudige methode, onafhankelijk van het type graaf en het aantal kleuren. Verder is ook te zien dat de andere parameters hier geen invloed op hebben.

In de tabel is te zien dat het aantal kleuren bijna geen invloed heeft. Dit wordt verklaard door het feit dat het aantal kleuren geen invloed heeft op de graaf, en de reductiemethodes gebruiken voornamelijk de structuur van de graaf. De Thiruvady knopen van type 2, knopen enkel verbonden met Unhappy voorgekleurde knopen met verschillende kleuren, komen vaker voor als er meer kleuren zijn, aangezien er minder kans is dat een knoop verbonden is met twee voorgekleurde knopen met

Graaf-type	**# kleuren**	**Eenvoudig**	**Articulatie**
Random	10	21.61	24.54
	50	21.72	24.59
Scale-free	10	44.34	49.73
	50	44.52	49.95
Lineair	10	33.71	36.57
	50	33.77	36.62

Tabel 9.1: Gemiddeld aantal knopen meer gereduceerd door beide nieuwe reductie-methodes in vergelijking met de methode van Thiruvady et al. in functie van het type graaf en het aantal kleuren.

dezelfde kleur. Dit verklaart het kleine verschil.

Het verschil tussen de verschillende types grafen is daarentegen wel duidelijk te zien. Hoewel het gemiddelde van de Scale-free grafen een stuk hoger lijkt, is er algemeen weinig verschil. Er is namelijk enkel een verschil tussen de methodes voor Scale-free grafen met $s = 1$, en voor alle andere waarden van s is er geen verschil voor beide methodes. Elke knoop in een Scale-free graaf heeft minstens s bogen. Er zijn dus geen knopen met graad 0 of 1 als s groter wordt. Aangezien de enige verbetering die de eenvoudige methode aanbrengt op het algoritme van Thiruvady et al. deze knopen nodig heeft, is het logisch dat er geen verschil is tussen deze methodes voor grotere waarden van s. Dit is ook de reden dat de articulatiemethode geen extra knopen reduceert, er zijn namelijk geen articulatieknopen in Scale-free grafen als s groter is dan 1.

Graaf-type	α	**Eenvoudig**	**Articulatie**
Lineair	-2	0	4.8
	-1.5	5.56	10.25
	-1	10.16	14.29
	-0.5	14.19	17.70
	0	21.86	24.77
	0.5	34.77	37.15
	1	63.43	65.83
	1.5	93.93	94.76
	2	59.78	59.82

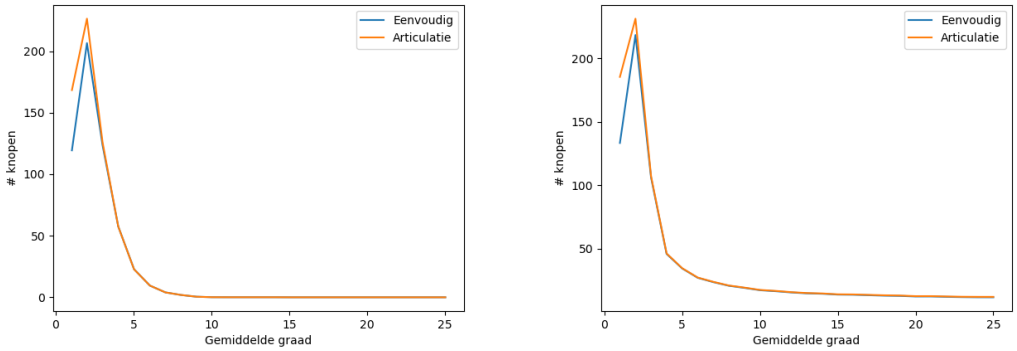
Tabel 9.2: Gemiddeld aantal knopen extra gereduceerd in grafen aangemaakt met de lineaire verdeling generator door beide nieuwe reductiemethodes in vergelijking met de methode van Thiruvady et al. in functie van α.

Tabel 9.2 toont de invloed van de parameter α van de lineaire verdeling generator op de verschillen tussen de reductiemethodes. Hier is te zien dat als de graden van de knopen dichter bij elkaar liggen (voor kleine waarden van α) er minder knopen

kunnen gereduceerd worden. De knopen die beide nieuwe methodes extra reduceren, hebben meestal een lage graad, en als er zo minder zijn, is het normaal dat het verschil tussen de verschillende methodes kleiner is. In het extreme geval met $\alpha = -2$, wat overeenkomt met d-reguliere grafen, kan de eenvoudige methode geen knopen extra reduceren, omdat knopen met graad 1 nooit zullen voorkomen.

De articulatie-methode lijkt hier wel een verbetering op te realiseren, maar dit komt enkel doordat er extra knopen worden gereduceerd als de gemiddelde graad ≤ 1. De originele methode reduceert alle niet-voorgekleurde knopen, en lost eigenlijk het probleem op. De articulatiemethode controleert hierna nog eens op knopen zonder bogen, en dit houdt alle overblijvende voorgekleurde knopen in die niet verbonden zijn met een andere voorgekleurde knoop. Hoewel dit tot meer gereduceerde knopen leidt, is deze niet relevant aangezien het probleem reeds triviaal is na de eerste fase.

In de tabel is ook te zien dat het geval met $\alpha = 0$ overeenkomt met de resultaten van de random grafen in tabel 9.1, wat te verwachten is aangezien dit ook uniforme random grafen zijn. Het geval met $\alpha = 2$ toont een groot verschil met het geval met $\alpha = 1.5$, wat eenvoudig te verklaren is aan de hand van figuur 8.1 waarin voorbeeldgrafen te zien zijn. Als $\alpha = 2$, is elke knoop verbonden met één of meer knopen met maximale graad, waar er bij $\alpha = 1.5$ een cluster is die bijna volledig geconnecteerd is en een groot aantal knopen met graad 0 bevat. Deze knopen met graad 0 worden eenvoudig gereduceerd, en verklaren het grote verschil tussen beide gevallen.



(a) Random grafen

(b) Lineaire grafen

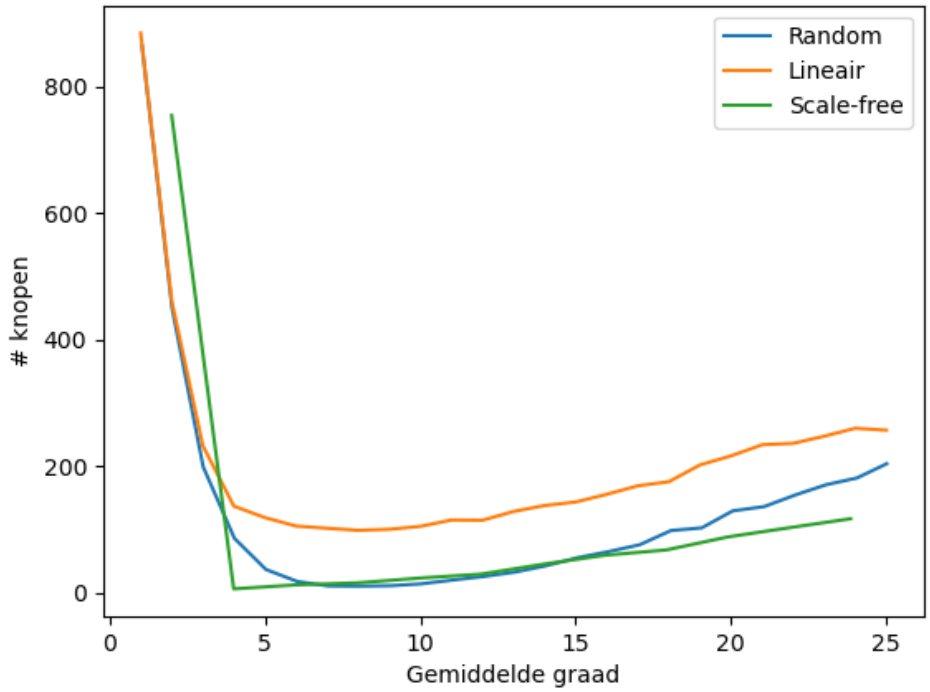
Figuur 9.1: Gemiddeld aantal knopen extra gereduceerd door beide nieuwe reductiemethodes in vergelijking met de methode van Thiruvady et al. in functie van de gemiddelde graad.

Figuur 9.1 toont hoe het verschil tussen de nieuwe methodes verloopt in functie van de gemiddelde graad van de graaf. Hierin is te zien dat de meeste reducties gebeuren bij grafen met zeer lage gemiddelde graad. Aangezien beide technieken

vooral knopen met zeer lage graad reduceren, is dit het verloop dat te verwachten is. Het verschil tussen beide nieuwe methodes is enkel merkbaar voor zeer lage graad, maar ook daar is slechts een beperkte verbetering op te merken.

9.1.2 Analyse reductiemethodes in instantie-ruimte

Figuur 9.2 toont hoe het totaal aantal knopen gereduceerd door de articulatiemethode verloopt in functie van de gemiddelde graad van de graaf. Hier is te zien dat het algemene verloop voor de drie types grafen identiek is. Grafen met zeer lage graad worden bijna volledig gereduceerd, zodat de enige knopen die overblijven bijna allemaal voorgekleurd zijn. Dit is te verwachten, omdat er veel knopen zijn met graad 0 of 1 die eenvoudig gereduceerd worden.



Figuur 9.2: Aantal knopen gereduceerd door de articulatiemethode in functie van de gemiddelde graad van de graaf.

In grafen met hoge gemiddelde graad is het een stuk moeilijker om knopen te reduceren, aangezien er minder knopen zijn met lage graad. De stijging in het aantal gereduceerde knopen voor hogere graden wordt volledig verklaard doordat er meer knopen van het Thiruvady type 2 zijn, die gegarandeerd Unhappy zijn.

Wat nog kan opgemerkt worden is dat de grafen gegenereerd met de lineaire verdeling generator meer gereduceerde knopen hebben dan de twee andere types grafen. Dit wordt opnieuw verklaard door het feit dat er veel knopen zijn met graad 0 in deze grafen, aangezien de resultaten op de grafiek uitgemiddeld zijn voor alle waarden van α.

In het algemeen wordt het kleinst aantal knopen gereduceerd rond het gebied tussen een gemiddelde graad van 5 en 10. Dit betekent dat hier de meeste knopen te vinden zijn die niet met de eenvoudige regels van de reductiemethodes kunnen 'opgelost' worden. Dit komt overeen met de bevindingen van Thiruvady et al.[25], die aangeven dat dit in hun onderzoek ook het moeilijke gebied bleek te zijn. Voor lagere graad blijkt het zeer eenvoudig om kleuren toe te kennen, omdat er zeer weinig beperkingen zijn door het lage aantal bogen. Voor hogere graad daarentegen wordt het 'eenvoudiger' omdat er juist veel meer bogen zijn, en het voor meer knopen onmogelijk wordt om Happy te zijn.

9.2 Resultaten exacte solver

In hoofdstuk 5 worden twee verschillende IP-modellen uiteengezet die beide door Lewis et al.[19] zijn voorgesteld. Tabel 5.1 geeft een overzicht van de grootte van beide modellen. Het eerste model is compacter en heeft minder beperkingen nodig om het probleem voor te stellen, het tweede model heeft meer variabelen en beperkingen nodig maar het domein van de variabelen is kleiner.

Het doel van de experimenten beschreven in deze sectie bestaat uit twee delen. Hoofdzakelijk was het de bedoeling om beide modellen met elkaar te vergelijken aan de hand van een implementatie in CPLEX. Door beide modellen eenzelfde probleem te laten oplossen en de uitvoeringstijden te vergelijken, is het mogelijk om een uitspraak te doen over experimentele verschillen tussen beide. Aan de hand van deze uitvoeringstijden is het ook mogelijk om een uitspraak te doen over de moeilijkheid van verschillende instanties, wat een tweede doel was van de experimenten. Beide aspecten worden samen besproken hieronder.

Het experiment gebruikt andere testinstanties dan diegene besproken in sectie 8.2, aangezien de exacte solver te veel tijd nodig heeft om grafen met $n = 1000$ knopen tot optimaliteit op te lossen. Voor een 'gemiddelde' instantie heeft de solver op de gebruikte hardware vaak minstens een uur nodig om de optimale oplossing te bekomen. Het is mogelijk om een tijdslimiet te zetten op de uitvoering van de solver, en een analyse te maken aan de hand van de voorlopige resultaten. Hier werd gekozen voor een andere optie, waarbij kleinere grafen worden bekeken, die wel op aanvaardbare tijd kunnen opgelost worden. De gebruikte grafen hebben slechts $n = 100$ knopen, en de exacte solver heeft gemiddeld slechts een aantal seconden nodig om deze instanties tot optimaliteit op te lossen. Het aantal kleuren k is bijgevolg ook aangepast van 10 en 50 naar 5 en 10, omdat minstens k knopen moeten

worden voorgekleurd. Voor de rest worden dezelfde instellingen gebruikt als voor de andere testinstanties.[2]

Type graaf	Eerste model	Tweede model
Random	0.50	2.99
Scale-free	0.38	1.96
Lineair	0.48	2.27

Tabel 9.3: Gemiddelde uitvoeringstijd van beide IP-modellen voor verschillende types grafen.

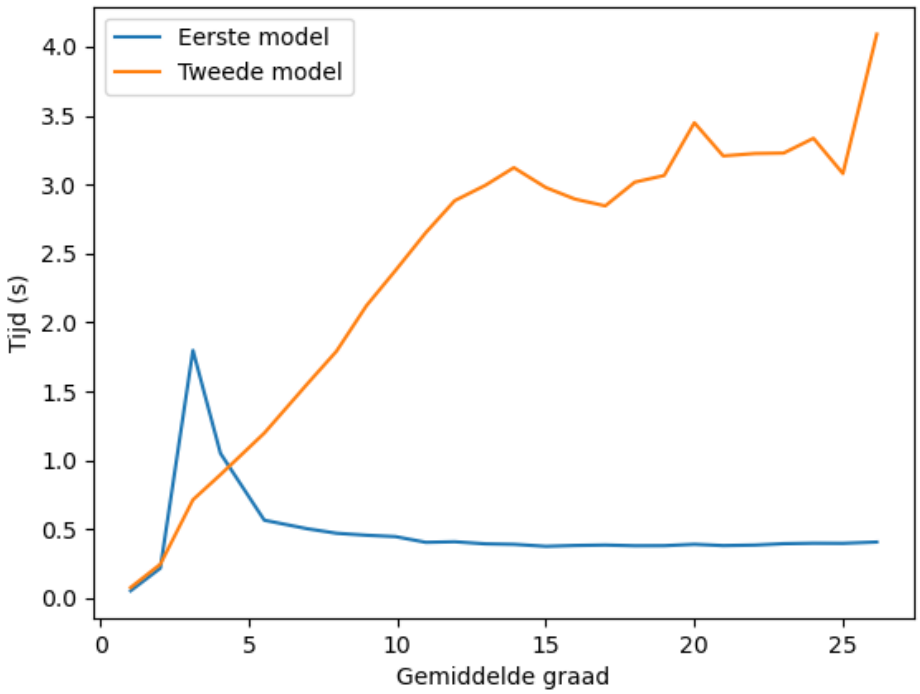
Tabel 9.3 toont de uitvoeringstijd van beide modellen voor de verschillende types grafen. Hierin is te zien dat het tweede model een stuk trager is dan het eerste, wat de bevindingen van Lewis et al. bevestigt. Zij hadden namelijk ook bekomen dat het tweede model trager tot een oplossing komt, door gebruik te maken van een implementatie in Gurobi. Het blijkt ook dat Scale-free grafen eenvoudiger op te lossen zijn dan de andere twee types in beide modellen. Een mogelijke verklaring hiervoor is dat Scale-free grafen altijd uit exact één component bestaan, wat niet gegarandeerd het geval is bij de andere twee types. Dit verkleint de zoekruimte omdat alle knopen met elkaar verbonden zijn, en er dus meer mogelijke oplossingen sneller kunnen uitgesloten worden.

# kleuren	# voorgekleurd	Eerste model	Tweede model
5	5	0.41	2.35
	15	0.37	1.41
	25	0.26	0.70
10	15	0.88	5.15
	25	0.49	2.02

Tabel 9.4: Gemiddelde uitvoeringstijd van beide IP-modellen voor verschillende voorkleuringen.

In tabel 9.4 is te zien wat de invloed is van verschillende voorkleuringen. Opvallend is dat de uitvoeringstijd daalt voor een groter aantal voorgekleurde knopen. Dit wordt op gelijkaardige manier verklaard als hierboven, aangezien elke voorgekleurde knoop een knoop minder is waarvoor een waarde gezocht moet worden. Uit het relatieve verschil tussen $k = 5$ en $k = 10$ is op te merken dat het tweede model meer extra tijd nodig heeft als er meer kleuren zijn. Dit kan veroorzaakt worden door de verschillende aanpak van beide modellen. Bij het eerste model wordt enkel het domein van de variabelen groter, terwijl er bij het tweede model variabelen bijkomen.

[2]Merk op dat instanties met $k = 10$ kleuren en $5\%(0.05 * 100 = 5)$ voorgekleurde knopen onmogelijk zijn, omdat minstens k knopen moeten worden voorgekleurd. Dit zorgt ervoor dat er slechts 5 verschillende kleurenconfiguraties zijn per graaf in plaats van 6 zoals in de andere dataset.



Figuur 9.3: Gemiddelde uitvoeringstijd van beide IP-modellen in functie van de gemiddelde graad van de graaf.

Figuur 9.3 toont hoe de uitvoeringstijd varieert in functie van de gemiddelde graad van de graaf. Hierin is te zien dat hoewel de gemiddelde uitvoeringstijd van het tweede model groter is dan die van het eerste, er een gebied is waarin het tweede model beter presteert. Voor grafen met een gemiddelde graad onder 5, blijkt dat het tweede model sneller tot de optimale oplossing komt. Dit effect werd ook opgemerkt door Lewis et al., zij geven aan dat het tweede model voor sommige kleine grafen soms beter presteerde, maar dat ze er geen patroon in konden herkennen.

Door enkel de resultaten van het eerste model te bekijken, zou men kunnen concluderen dat het gebied rond een gemiddelde graad van 4 een moeilijk gebied is in de instantie-ruimte. Omgekeerd kan men uit de resultaten van het tweede model besluiten dat instanties moeilijker worden als de gemiddelde graad oploopt. Deze conclusies hebben echter weinig betekenis, want deze gelden enkel met zekerheid voor deze implementatie van deze IP-modellen, en er is geen enkele garantie dat ze ook gelden in het algemeen.

Door ook te kijken naar figuur 9.1, lijkt het waarschijnlijk dat de moeilijkheid van een instantie een combinatie is van beide effecten. In grafen met lage gemiddelde graad zijn er veel knopen die triviaal kunnen opgelost worden, en bij hoge gemiddelde graad zijn er veel knopen die gegarandeerd Unhappy zijn en die geen invloed hebben op de rest van de graaf.

9.3 Analyse gedrag selector in Growth-MHV

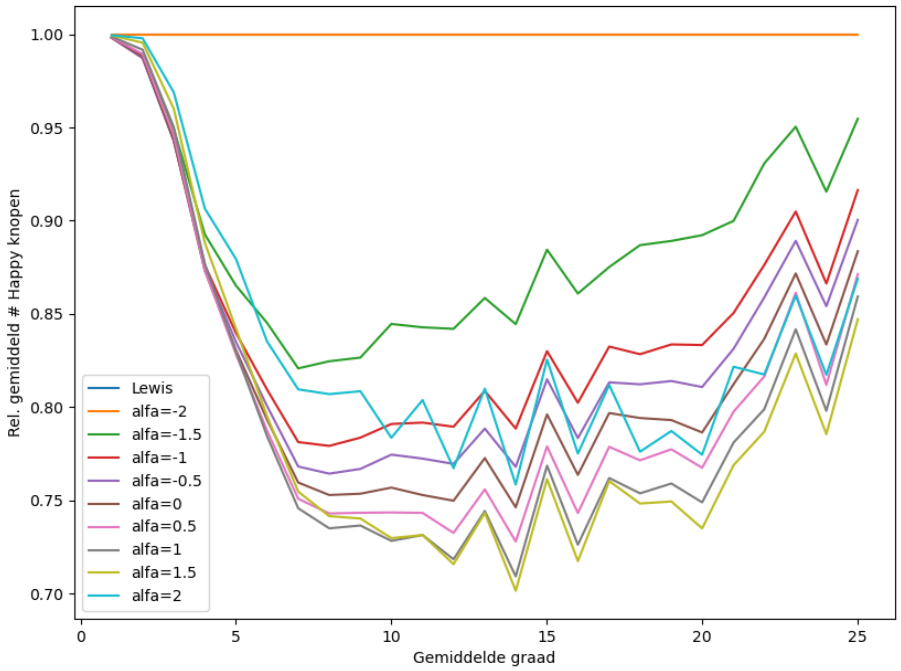
Het Growth-MHV algoritme moet op verschillende plaatsen in het beslissingsproces een keuze maken tussen verschillende mogelijke knopen. In de originele paper van Li en Zhang[28] wordt niet vermeld hoe deze keuze best gemaakt wordt. Lewis et al. stellen in hun paper[19] en source code[16] voor om altijd de knoop met hoogst mogelijke graad uit te kiezen. De argumentering die ze hierbij geven is dat het algoritme dan sneller is. In deze thesis is een alternatieve selectiemethode geïmplementeerd, die gebruik maakt van een lineaire verdeling om variabele bias te kunnen geven aan het random kiezen van knopen.

In deze sectie wordt de selector van Lewis et al. vergeleken met de lineaire verdeling selector. De resultaten hieronder zijn bekomen door het Growth algoritme met verschillende selectoren uit te voeren op de 31440 instanties beschreven in 8.2. Voor de lineaire verdeling selector zijn 9 verschillende waarden gebruikt voor de bias α_{bias}[3], $\alpha_{bias} \in \{x/2 | x \in \mathbb{Z} \cup [-4, 4]\}$. Aangezien deze selector random gedrag vertoont, wordt het algoritme met de lineaire verdeling selector voor elke graaf 20 keer uitgevoerd voor elke waarde van α_{bias}. In totaal geeft dit $31440 * (1 + 9 * 20) = 5.69$ miljoen uitvoeringen van het algoritme.

Figuur 9.4 toont het gemiddeld aantal Happy knopen voor de verschillende selectiemethodes in functie van de gemiddelde graad van de graaf. De grafiek geeft de relatieve verschillen ten opzichte van het grootste gemiddelde weer $\left(\frac{x_i}{max(x_i)}\right)$, aangezien het aantal Happy knopen sterk verschilt tussen grafen met verschillende gemiddelde graad. Een eerste feit dat vastgesteld kan worden, is dat de gemiddeldes van de selector van Lewis et al. en $\alpha_{bias} = -2$ volledig samenvallen. Dit is verwacht, aangezien deze beide exact hetzelfde gedrag vertonen, zoals uitgelegd wordt op het einde van sectie 6.1.

Verder is te zien dat de selector van Lewis et al. voor zo goed als alle instanties ruimschoots de beste resultaten behaalt. Beide lijnen zijn volledig horizontaal in de grafiek, wat betekent dat deze voor elke gemiddelde graad het beste resultaat bekomen. Dit betekent dat de deterministische selectiemethode van Lewis et al. consistent goede resultaten behaalt. Aangezien deze selector ook tot een kortere uitvoeringstijd leidt, is het duidelijk dat deze selector superieur is aan alle random selectiemethodes die hier bekeken werden.

[3] Merk op dat deze parameter verschilt van de parameter van de lineaire verdeling generator.



Figuur 9.4: Gemiddeld aantal Happy knopen relatief ten opzichte van het maximum aantal Happy knopen voor verschillende selectiemethodes.

Wat ook opgemerkt kan worden is dat de invloed van de bias duidelijk te zien is in de grafiek. Kleinere waarden voor α_{bias}, wat betekent dat er meer voorkeur gegeven wordt aan knopen met hoge graad, zorgen voor betere prestaties dan uniforme keuzes ($\alpha_{bias} = 0$), en een grotere bias zorgt voor slechtere prestaties. Een uitzondering hierop is $\alpha_{bias} = 2$, deze bias zorgt voor prestaties die niet in lijn liggen met de rest. Voor deze instelling van de bias gedraagt het algoritme zich deterministisch, het kiest namelijk altijd de knoop met de laagste graad. Hoewel de prestaties gemiddeld gezien redelijk goed zijn, zal deze instelling nooit een oplossing bekomen die beter is dan de andere deterministische setting (de selector van Lewis et al.).

De beste selector met random gedrag gebruikte een bias van $\alpha = -1.5$. Hoewel de gemiddelde prestaties van deze selector slechter zijn dan die van de selector van Lewis et al., blijkt dat voor 68% van de grafen het beste resultaat dat deze selector behaalde in de 20 uitvoeringen wel beter is dan het resultaat behaald door de Lewis selector. Dit betekent dat als het mogelijk is om het Growth algoritme meerdere keren uit te voeren, het interessant kan zijn om een aantal keer de random selector met een kleine bias ($\alpha = -1.5$ of kleiner) uit te voeren, om eventueel een beter resultaat te bekomen.

9.4 Prestaties metaheuristiek

In sectie 9.4.1 wordt het metaheuristisch algoritme ontworpen in het kader van deze thesis, beschreven in hoofdstuk 7, geanalyseerd. Een eerste luik van deze analyse vergelijkt de prestaties van de metaheuristiek met de twee constructieve algoritmen, aangezien deze de enige beschikbare oplossingsmethodes waren op het moment dat de metaheuristiek ontworpen is. Hiernaast worden de prestaties ook vergeleken met die van het Tabu-search algoritme gepubliceerd in het begin van 2020 door Thiruvady et al.[25].

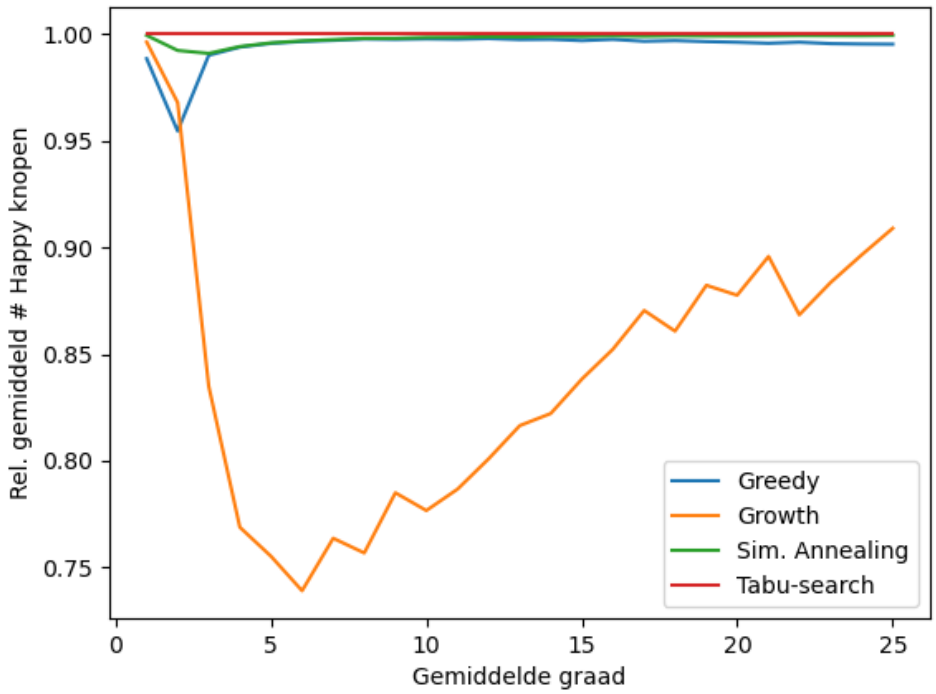
Tot slot wordt in sectie 9.4.2 dieper ingegaan op de gemaakte experimentele vergelijking tussen beide algoritmen. Hier wordt het uitgevoerde experiment kritisch geëvalueerd op een hoger niveau.

9.4.1 Experimentele resultaten

Het experiment bestaat uit de uitvoering van vier algoritmen (Greedy-MHV, Growth-MHV, Simulated Annealing en Tabu-search) op alle testinstanties beschreven in sectie 8.2. Voor de metaheuristiek worden de optimale parameters gebruikt, die beschreven worden in sectie 8.3. Het computationele budget bestaat uit 5000 iteraties, wat op de gebruikte hardware overeenkomt met ongeveer 5 seconden uitvoeringstijd voor het Simulated Annealing proces. Het Tabu-search algoritme krijgt bijgevolg ook 5 seconden uitvoeringstijd ter beschikking, en wordt uitgevoerd met de default parameters.

Figuur 9.5 toont de gemiddelde prestaties van de vier algoritmen, relatief ten opzichte van de beste prestatie. Hier is te zien dat hoewel het Greedy algoritme voor grafen met zeer lage graad slechter presteert dan het Growth algoritme, het voor de rest van de instanties duidelijk veel beter presteert. Merk hierbij op dat de originele instantie niet enkel met het Greedy algoritme wordt opgelost. Voor elk algoritme buiten Tabu-search wordt de instantie eerst gereduceerd met de articulatie-methode vooraleer het algoritme erop wordt toegepast. Dit betekent dat voor de meeste grafen de combinatie van articulatiereductie en het Greedy algoritme een snelle oplossingsmethode is die toch leidt tot relatief goede resultaten.

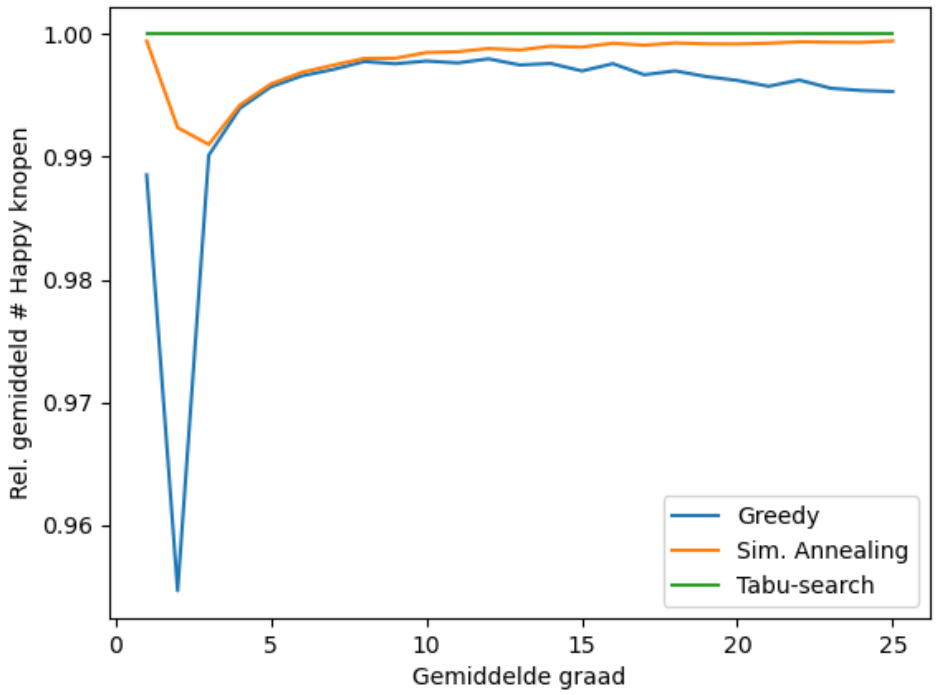
Het Simulated Annealing algoritme vertrekt van de beste van de twee constructieve algoritmen, dus het is logisch dat het altijd een oplossing vindt die minstens even goed is als deze. In de grafiek is te zien dat er enkel een groot verschil is tussen het beste van de twee en Simulated Annealing in het gebied waarin beide constructieve algoritmen ongeveer even goed presteren. Dit gebied komt overeen met het moeilijke gebied beschreven in de vorige secties. Hier wordt het kleinste aantal knopen gereduceerd, en moet er dus een grotere instantie opgelost worden door het effectieve algoritme.



Figuur 9.5: Relatieve gemiddelde prestaties van de vier algoritmen in functie van de gemiddelde graad van de graaf.

In figuur 9.6 worden dezelfde resultaten getoond als in figuur 9.5, maar zonder de resultaten van het Growth algoritme, omdat deze een stuk slechter zijn dan die van de andere drie. Hier is duidelijker te zien dat het Tabu-search algoritme algemeen de beste resultaten bekomt. Voor slechts 307 van de 31440 testinstanties slaagde het Simulated Annealing algoritme erin om een betere oplossing te bekomen dan het Tabu-search algoritme, en deze hadden allemaal een gemiddelde graad tussen 1 en 2. Wat ook opgemerkt kan worden, is dat hoewel het verschil tussen beide metaheuristieken consistent is, de gemiddelde prestaties van beide dicht bij elkaar liggen. Het grootste relatieve verschil tussen beide gemiddeldes is kleiner dan 1%, wat voor de gebruikte testinstanties neerkomt op een paar knopen.

Op deze grafiek is ook duidelijk te zien dat het Simulated Annealing algoritme vertrekt van de Greedy oplossing vanaf een gemiddelde graad van ongeveer 4. Het is opvallend dat de prestatie van beide algoritmen eerst bijna identiek is, maar dat met stijgende gemiddelde graad het verschil tussen beide groeit en Simulated Annealing dichter bij het resultaat van Tabu-search komt.

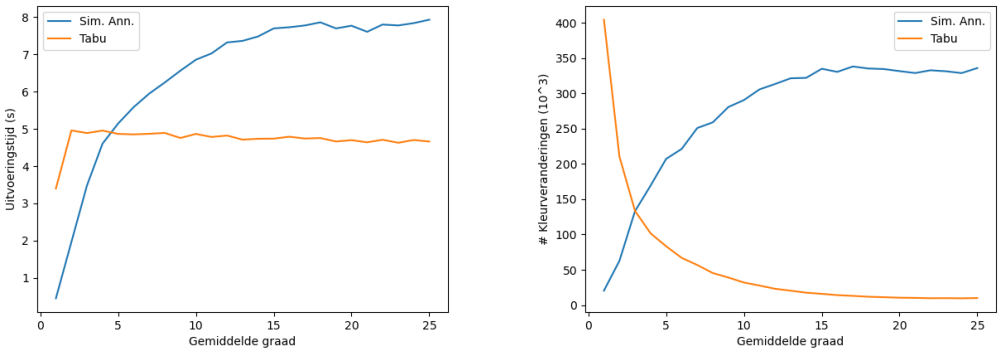


Figuur 9.6: Relatieve gemiddelde prestaties van Greedy, Simulated Annealing en Tabu-search in functie van de gemiddelde graad van de graaf.

9.4.2 Analyse van het experiment

In deze sectie wordt op een hoger niveau met een kritische blik gekeken naar het uitgevoerde experiment. Aangezien de implementaties van beide algoritmen verschillen in efficiëntie, en de source code van Tabu-search[18] enkel een tijdslimiet om het computationele budget te beperken voorziet, is het niet eenvoudig om een eerlijke vergelijking tussen beide op te stellen.

De gebruikte code voor het Tabu-search algoritme is bijzonder goed ontworpen, en gebruikt een zeer efficiënte manier om de zoekruimte te overlopen. De gebruikte implementatie van het Simulated Annealing algoritme daarentegen kan op verschillende plaatsen nog geoptimaliseerd worden. In elke iteratie moet berekend worden welke gekleurde groepen in de graaf aanwezig zijn, en dit wordt momenteel in elke iteratie herhaald. Het is mogelijk om dit te implementeren a.d.h.v. een Δ-operator, die voor elke buur-operator berekent welke verschillen dit introduceert in de huidige groepen. Dit zou een groot verschil kunnen maken in de snelheid van de implementatie, aangezien uit analyse van de uitvoering van de code blijkt dat hierbij een groot deel van de tijd verloren gaat.



(a) Uitvoeringstijd (b) Kleurveranderingen

Figuur 9.7: Uitvoeringstijd en aantal kleurveranderingen in beide algoritmen in functie van de gemiddelde graad van de instantie.

Figuur 9.7a toont hoe de uitvoeringstijd van beide algoritmen verloopt in functie van de gemiddelde graad van de instantie. Hier is te zien dat het Tabu-search algoritme de limiet van 5 seconden volgt. Het Simulated Annealing algoritme heeft geen tijdslimiet gekregen in de experimenten, maar een limiet van 5000 iteraties. Op de grafiek is te zien dat de tijd benodigd voor deze iteraties varieert voor verschillende grafen. Als de gemiddelde graad hoger wordt is er meer tijd nodig, omdat bij het opbouwen van de gekleurde groepen de bogen van alle knopen moeten worden gecontroleerd.

Figuur 9.7b toont het aantal knopen dat van kleur verandert in het zoekproces voor beide algoritmen. Voor het Tabu-search algoritme komt dit overeen met het aantal iteraties, aangezien in elke iteratie slechts één knoop van kleur verandert. Bij het Simulated Annealing algoritme kunnen meerdere knopen van kleur veranderen in elke iteratie, aangezien de buur-operatoren een hele groep knopen van kleur veranderen. Uit de resultaten is af te leiden dat het aantal iteraties bij het Tabu-search algoritme sterk daalt in functie van de gemiddelde graad. Dit komt door het feit dat het algoritme alle buren van een knoop afgaat in elke iteratie, en dus meer tijd nodig heeft als de gemiddelde graad hoger wordt. Het Simulated Annealing algoritme daarentegen verandert meerdere knopen per iteratie als de gemiddelde graad groter wordt. Dit gebeurt omdat de grootte van de gekleurde groepen groter wordt als de gemiddelde graad groter wordt, en er dus veel knopen van kleur veranderen als die groep een andere kleur krijgt. Voor lage graad daarentegen zijn de groepen veel kleiner, en worden dus minder knopen per iteratie van kleur veranderd.

Hoewel de curve van Simulated Annealing in figuur 9.7b aangeeft dat er veel meer knopen van kleur veranderen dan in het Tabu-search algoritme, worden in elke uitvoering in dit experiment slechts 5000 verschillende oplossingen gegenereerd, namelijk

één per iteratie. Dit geeft een idee van de grootte van het tijdens het zoekproces bekeken deel van de oplossingsruimte. Bij het Tabu-search algoritme daarentegen komt de curve op de grafiek wel overeen met het aantal bekeken oplossingen, dat vooral voor lage gemiddelde graad een stuk hoger is dan de 5000 van Simulated Annealing. Dit zou een mogelijke verklaring kunnen zijn voor het verschil tussen de prestaties van beide, dat het grootst is in dit gebied.

De twee grafieken in figuur 9.7 geven samen aan wat het effect is van de verschillende beperkingen op het computationele budget. Als de optimalisatie die hierboven wordt voorgesteld zou toegevoegd worden aan de code van Simulated Annealing, verandert er niets aan de grafiek met het aantal kleurveranderingen, maar de benodigde tijd zou wel een stuk kunnen dalen, wat eventueel zou kunnen toelaten om het Simulated Annealing algoritme een groter computationeel budget te geven, zonder dat het meer tijd inneemt dan het Tabu-search algoritme. Voor deze eventueel meer eerlijke vergelijking tussen beide algoritmen was er geen tijd meer binnen het tijdsbudget van deze thesis, maar dit zou eventueel bekeken kunnen worden in verder onderzoek.

9.5 Conclusies

De individuele resultaten van de vier verschillende experimenten in dit hoofdstuk worden in hun respectievelijke sectie al besproken, maar er zijn ook conclusies die op een meer algemeen niveau kunnen getrokken worden. In de experimenten wordt bevestigd dat de articulatie-methode de reductiemethode is die het grootste aantal knopen uit de instantie reduceert. Samen met het Greedy algoritme vormt dit een oplossingsmethode die aanzienlijk sneller is dan beide metaheuristieken die hier geanalyseerd werden, en ook een goede oplossing bekomt.

Hoewel er in sectie 9.3 uitgebreid gekeken wordt naar verschillende selectoren voor het Growth algoritme, blijkt dat in de meeste gevallen het Greedy algoritme toch betere oplossingen bekomt. Dit was ook de reden dat er niet gekeken is naar de invloed van de selectiemethode op de initiële oplossing van het metaheuristisch algoritme. Hoewel het mogelijk is om een betere initiële oplossing te bekomen door eventueel het Growth algoritme verschillende keren uit te voeren met de random selector, is de invloed hiervan waarschijnlijk niet merkbaar.

Het Simulated Annealing algoritme ontworpen in deze thesis (in combinatie met de articulatie-reductiemethode) blijkt een verbetering te zijn op de twee constructieve algoritmen, maar slaagt er niet in om de prestaties van het Tabu-search algoritme te evenaren. Relatief gezien liggen de prestaties minder dan 1% onder die van het Tabu-search algoritme. Een mogelijke reden hiervoor is dat de gebruikte implementaties van beide algoritmen niet even efficiënt zijn, wat het moeilijk maakt om volledig eerlijke testen op te zetten.

Uit deze experimenten kan niet afgeleid worden of de efficiëntie van de gebruikte code de enige reden is waarom de resultaten slechter zijn dan die van het Tabu-search algoritme, of dat er algoritmische beperkingen zijn in de gebruikte aanpak. Dit zou kunnen bekeken worden in een volgend onderzoek, aangezien er binnen het voorziene tijdsbudget geen tijd meer was voor de verdere optimalisatie van de gebruikte implementatie.

Hoofdstuk 10

Besluit

In deze thesis werden verschillende technieken besproken en geanalyseerd om het Maximum Happy Vertices probleem op te lossen. Een eerste stap in het oplossen van het probleem is de toepassing van de reductiemethodes beschreven in hoofdstuk 4. Beide nieuwe algoritmen die hier voorgesteld worden vormen een uitbreiding op de techniek beschreven door Thiruvady et al.[25], en slagen erin de probleeminstantie aanzienlijk te verkleinen in de meeste gevallen. Het gebied waar het kleinste aantal knopen wordt gereduceerd, rond grafen met een gemiddelde graad tussen 5 en 10, komt overeen met het moeilijke gebied dat gevonden is in vorige papers, en met het moeilijke gebied dat wordt gevonden in de andere experimenten.

Vervolgens werd gekeken naar de constructieve algoritmen van Li en Zhang[28], Greedy-MHV en Growth-MHV. Uit de resultaten blijkt dat de combinatie van articulatiereductie en het Greedy algoritme op zeer beperkte tijd (minder dan 1s) zeer goede resultaten behaalt. Het Growth algoritme daarentegen behaalt ondanks de grotere complexiteit significant slechtere resultaten, en komt overduidelijk uit de resultaten als het slechtste algoritme voor de bekeken instanties. Lewis et al. maken in hun implementatie van het Growth algoritme een keuze voor een specifieke selectiemethode, en ze geven aan dat dit enkel gekozen is om snelheidsredenen. De uitgebreide analyse van deze en een alternatieve selectiemethode geeft aan dat hun selector niet alleen sneller is, maar ook de beste resultaten behaalt.

Beide Integer Programming modellen die worden voorgesteld door Lewis et al.[19] werden zowel theoretisch als experimenteel vergeleken. Lewis et al. gaven aan dat het compact model bij hen betere resultaten bekwam, zonder dit te onderbouwen met resultaten. Dit wordt wel bevestigd in de uitgebreide experimenten in deze thesis.

Tot slot werd er in deze thesis een nieuw metaheuristisch algoritme voorgesteld dat gebruik maakt van het Simulated Annealing framework. Dit algoritme maakt gebruik van drie verschillende probleem-specifieke buur-operatoren, ontwikkeld in deze thesis. Dit algoritme slaagt er niet in om de resultaten van het Tabu-search algoritme van Thiruvady et al.[25] te evenaren in de opgestelde experimenten. De gemiddelde

prestaties van dit algoritme blijven wel binnen een relatief verschil van minder dan 1%.

Er zijn verschillende pistes voor verder onderzoek rond het Maximum Happy Vertices probleem. De referentie-gebaseerde reductiemethode geïntroduceerd in deze thesis kan eventueel uitgebreid worden met andere selectiemethodes naast degene die hier voorgesteld werden. Verder is het mogelijk om de voorgestelde verbeteringen te implementeren voor het Simulated Annealing algoritme. Deze verbeteringen zouden toelaten om een meer eerlijke vergelijking te maken tussen beide algoritmen, waar geen tijd meer voor was binnen deze thesis. Hierbij kan ook de reductiemethode worden toegevoegd aan het Tabu-search algoritme, en kan bekeken worden of dit de prestaties ervan verbetert.

Het onderzoek naar de moeilijkheid van de instanties in deze thesis was zeer beperkt. Het is mogelijk dit onderzoek uit te bereiden naar een volledig onderzoek van de instantie-ruimte, en eventueel te bekijken of er gebieden zijn waar bepaalde algoritmen beter presteren dan andere, zoals in [23] wordt uitgevoerd voor het Graph Colouring Probleem.

Bijlagen

Bijlage A

Lineaire kansverdeling

In bepaalde gevallen kan het handig zijn om een kansverdeling te hebben die aan de hand van een parameter kan ingesteld worden op een bepaalde bias. In deze thesis is hiervoor een lineaire kansverdeling opgesteld, die zoals de naam aangeeft een lineaire kansverdelingsfunctie heeft.

A.1 Eenvoudige lineaire kansverdeling

De basis van de kansverdeling is geïnspireerd op een antwoord op een vraag op StackExchange [1] [26], waarbij een lineaire verdeling wordt voorgesteld, weergegeven in vergelijking A.1.

$$pdf(x) = \frac{1 + \alpha x}{2}, x \in [-1, 1], \alpha \in [-1, 1] \tag{A.1}$$

Om random getallen te genereren volgens deze verdeling, wordt de Inverse Transform Sampling[8] techniek gebruikt. Deze techniek transformeert uniform verdeelde getallen naar eender welke verdeling aan de hand van de inverse van de cumulatieve kansverdeling. Dit kan men verklaren door het feit dat als $\mathbf{X}$ een continue random variabele is met cumulatieve kansverdeling $\mathbf{F_x}$, de variabele $\mathbf{Y} = \mathbf{F_x}(\mathbf{X})$ uniform verdeeld is. Als men aan beide kanten van deze vergelijking de inverse van cumulatieve kansverdeling $\mathbf{F_x}^{-1}$ toevoegt, verkrijgt men de gebruikte techniek.

[1] `stackexchange.com`

De eerste stap bestaat uit het berekenen van de cumulatieve kansverdeling, weergegeven in vergelijking A.2. De volgende stap is het berekenen van de inverse van deze functie, weergegeven in A.3

$$\begin{aligned} cdf(x) &= \int_{-\infty}^{x} f(t)dt \\ &= \int_{-1}^{x} \frac{1+\alpha t}{2} dt \\ &= \frac{t}{2} + \frac{\alpha t^2}{4}\Big|_{-1}^{x} \\ &= \frac{\alpha}{4}x^2 + \frac{1}{2}x + \frac{2-\alpha}{4} \end{aligned} \tag{A.2}$$

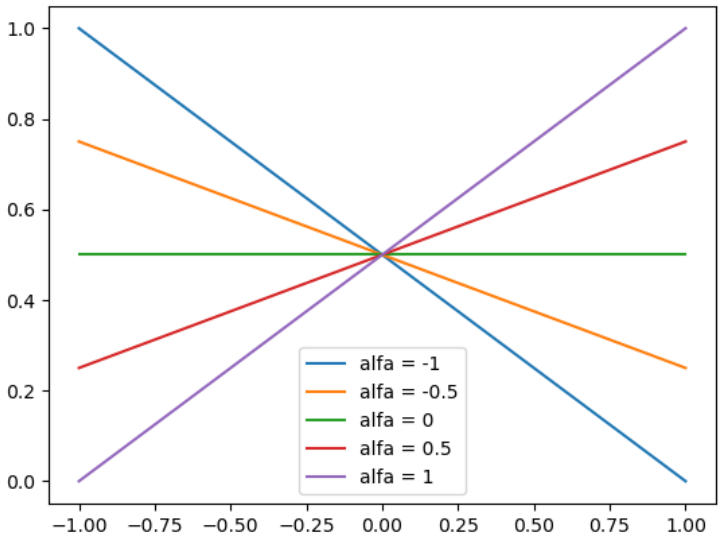
$$\begin{aligned} y &= \frac{\alpha}{4}x^2 + \frac{1}{2}x + \frac{2-\alpha}{4} \\ 4y &= \alpha x^2 + 2x + 2 - \alpha \\ x &= \frac{-2 \pm \sqrt{16\alpha y + 4\alpha^2 - 8\alpha + 4}}{2\alpha} \\ cdf^{-1}(y) &= \frac{\sqrt{4\alpha y + \alpha^2 - 2\alpha + 1} - 1}{\alpha} \end{aligned} \tag{A.3}$$

Deze vergelijking kan dan gebruikt worden om random getallen te genereren tussen -1 en 1 aan de hand van een uniform verdeelde variabele zoals weergegeven in vergelijking A.4. Deze verdeling wordt verder gebruikt als **Lineair**(α).

$$x = \frac{\sqrt{4\alpha u + \alpha^2 - 2\alpha + 1} - 1}{\alpha}, u \sim Uniform(0,1), \alpha \in [-1,1] \tag{A.4}$$

A.2 Uitbreiding voor grotere bias

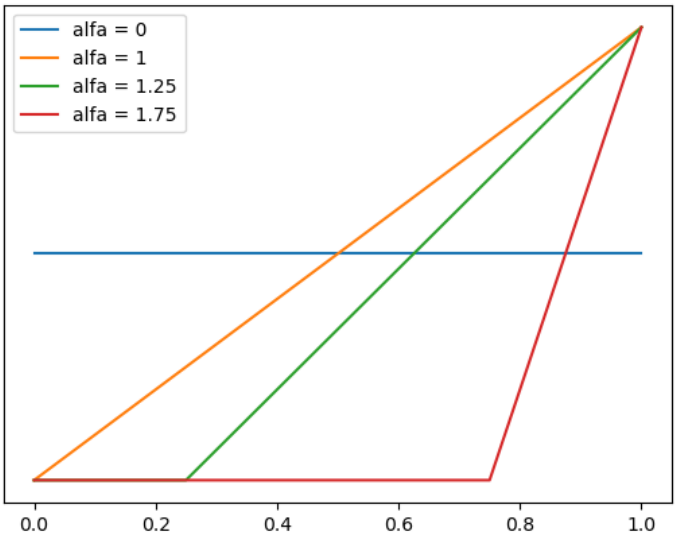
De verdeling beschreven hierboven laat al toe om verschillende soorten bias aan een verdeling te geven. De verdeling voor een aantal waarden van α wordt weergegeven in figuur A.1. Het is ook nog mogelijk deze verdeling te gebruiken om een andere verdeling op te stellen, die een nog sterkere bias toelaat.



Figuur A.1: **Lineair**(α) voor verschillende waarden van α.

De uitgebreide verdeling **Lin**(α) laat een groter domein toe voor α, $[-2, 2]$. De volledige verdeling wordt weergegeven in vergelijking A.5, en een grafische voorstelling wordt gegeven in A.2. Dit genereert getallen tussen 0 en 1, en volgens dezelfde verdeling als **Lineair**(α) zolang α tussen -1 en 1 ligt..

$$\mathbf{Lin}(\alpha) = \begin{cases} \dfrac{(\mathbf{Lineair}(1)+1)(2-\alpha)+2\alpha-2}{2} & \alpha > 1 \\ \dfrac{\mathbf{Lineair}(\alpha)+1}{2} & -1 \leq \alpha \leq 1 \\ \dfrac{(\mathbf{Lineair}(-1)+1)(2+\alpha)}{2} & \alpha < -1 \end{cases} \tag{A.5}$$



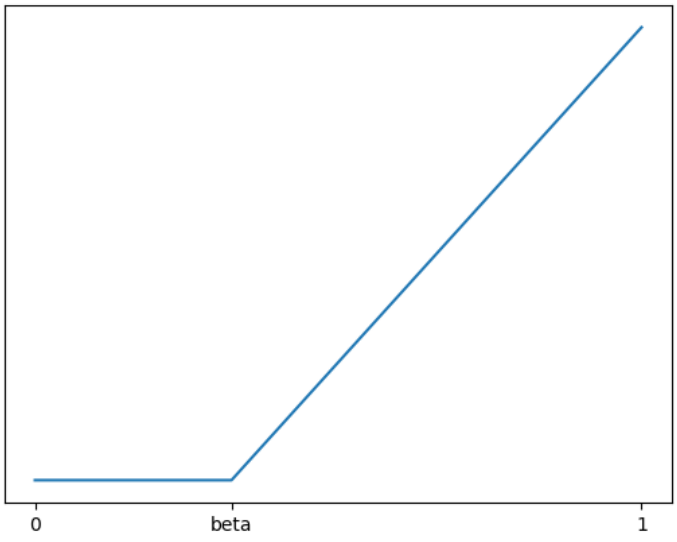
Figuur A.2: **Lin**(α) voor verschillende waarden van $\alpha > 0$. Merk op dat dit slechts een overzicht schetst van het verloop van de verdeling voor verschillende waarden, de curves voor $\alpha = 1.25$ en $\alpha = 1.75$ zijn geschaald zodat hun maximum samenvalt met die voor $\alpha = 1$, wat geen geldige verdeling meer is aangezien de oppervlakte onder de curve niet gelijk is aan 1.

De eerste stap in de afleiding van de drie formules weergegeven in vergelijking A.5 is de transformatie van het domein van de originele verdeling **Lineair**(α). Deze wordt bekomen door de functie te verschuiven en te schalen met deze formule: $\mathbf{Y} = \dfrac{\mathbf{X}+1}{2}$. Dit transformeert het domein van $[-1, 1]$ naar $[0, 1]$. Dit is de enige bewerking nodig voor het tweede geval.

In het bovenste geval, met $\alpha > 1$, willen we een verdeling bekomen die eruitziet zoals figuur A.3. *beta* bepaalt hoe sterk de bias is, als $beta = 0$ komt de verdeling overeen met **Lineair**(1) en als $beta = 1$ dan komt de verdeling overeen met een deterministische verdeling waar altijd exact 1 gekozen wordt. Deze *beta* kan gelijkgesteld worden aan $\alpha - 1$, zodat het mogelijke domein van α wordt uitgebreid van 1 tot 2.

Het opstellen van de volledige vergelijking komt neer op een extra transformatie van de reeds getransformeerde verdeling naar $[beta, 1]$. Dit wordt afgeleid in vergelijking A.6. De afleiding van het derde geval, met $\alpha < 1$, verloopt analoog, met $beta = \alpha + 2$.

$$
\begin{aligned}
\mathbf{X} &= \mathbf{Lineair}(1) \\
\mathbf{Y} &= \frac{\mathbf{X}+1}{2} \\
\mathbf{Z} &= \mathbf{Y}(1-beta)+beta \\
&= \frac{\mathbf{X}+1}{2}(1-beta)+beta \\
&= \frac{\mathbf{X}+1}{2}(2-\alpha)+\alpha-1 \\
&= \frac{(\mathbf{Lineair}(1)+1)(2-\alpha)+2\alpha-2}{2}
\end{aligned}
\tag{A.6}
$$



Figuur A.3: Eenvoudige afbeelding van de uitgebreide lineaire verdeling voor een waarde van $\alpha > 1$, met $beta = \alpha - 1$

Bibliografie

[1] A. Agrawal. On the parameterized complexity of happy vertex coloring. In L. Brankovic, J. Ryan, and W. F. Smyth, editors, *Combinatorial Algorithms*, pages 103–115, Cham, 2018. Springer International Publishing.

[2] R. Albert and A. lászló Barabási. *Statistical mechanics of complex networks*, volume 74, chapter VII. Scale-Free Networks, pages 71–76. American Physical Society (APS), 2002.

[3] N. R. Aravind, S. Kalyanasundaram, and A. S. Kare. Linear time algorithms for happy vertex coloring problems for trees. In V. Mäkinen, S. J. Puglisi, and L. Salmela, editors, *Combinatorial Algorithms*, pages 281–292, Cham, 2016. Springer International Publishing.

[4] N. R. Aravind, S. Kalyanasundaram, A. S. Kare, and J. Lauri. Algorithms and hardness results for happy coloring problems. *ArXiv*, 2017.

[5] W. Cunningham. The optimal multiterminal cut problem. *Discrete Mathematics & Theoretical Computer Science*, 5, 1991.

[6] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, 2nd edition, 2015.

[7] P. Erdös and A. Rényi. On random graphs I. *Publicationes Mathematicae Debrecen*, 6:290, 1959.

[8] S. I. Gass and M. C. Fu. Inverse transform method. In *Encyclopedia of Operations Research and Management Science*, pages 815–815, Boston, MA, 2013. Springer US.

[9] F. Glover and M. Laguna. Tabu search. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization: Volume1-3*, pages 2093–2229, Boston, MA, 1998. Springer US.

[10] B. Granger, M. Yu, and K. Zhou. Optimization with absolute values. URL: `www.optimization.mccormick.northwestern.edu/index.php/Optimization_with_absolute_values`, 2014. Laatst nagekeken Mei 2020.

[11] K. L. Hoffman, M. Padberg, and G. Rinaldi. Traveling salesman problem. In S. I. Gass and M. C. Fu, editors, *Encyclopedia of Operations Research and Management Science*, pages 1573–1578, Boston, MA, 2013. Springer US.

[12] J. Hopcroft and R. Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.

[13] D. Karger. Global min-cuts in *rnc* and other ramifications of a simple mincut algorithm. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 21–30, 1993.

[14] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[15] B. Korte and J. Vygen. The knapsack problem. In *Combinatorial Optimization: Theory and Algorithms*, pages 439–448, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[16] R. Lewis. Algorithm source code. URL: `www.rhydlewis.eu/resources/happyalgs.zip`, 2019. Laatst nagekeken Mei 2020.

[17] R. Lewis. Problem instance generator. URL: `www.rhydlewis.eu/resources/happyGen.zip`, 2019. Laatst nagekeken Mei 2020.

[18] R. Lewis. Tabu search source code. URL: `www.rhydlewis.eu/resources/happytabu.zip`, 2020. Laatst nagekeken Mei 2020.

[19] R. Lewis, D. Thiruvady, and K. Morgan. Finding happiness: An analysis of the maximum happy vertices problem. *Computers & Operations Research*, 103:265 – 276, 2019.

[20] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43 – 58, 2016.

[21] F. Peeters. Thesis source code. URL: `www.github.com/FPeeters/MaximumHappyVertices`, 2020. Laatst nagekeken Mei 2020.

[22] K. Smith-Miles, D. Baatar, B. Wreford, and R. Lewis. Towards objective measures of algorithm performance across instance space. *Computers & Operations Research*, 45:12–24, 2014.

[23] K. Smith-Miles and T. T. Tan. Measuring algorithm footprints in instance space. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–8, 2012.

[24] A. Steger and N. C. Wormald. Generating random regular graphs quickly. *Combinatorics, Probability and Computing*, 8(4):377–396, 1999.

[25] D. Thiruvady, R. Lewis, and K. Morgan. Tackling the maximum happy vertices problem in large networks. *4OR*, 2020.

[26] wolfies (www.stats.stackexchange.com/users/24905/wolfies). Generate random numbers with linear distribution. URL:www.stats.stackexchange.com/q/171631. Laaste nagekeken: Maart 2020.

[27] J. Xue. Graph coloring. In C. A. Floudas and P. M. Pardalos, editors, *Encyclopedia of Optimization*, pages 1444–1448, Boston, MA, 2009. Springer US.

[28] P. Zhang and A. Li. Algorithmic aspects of homophyly of networks. *Theoretical Computer Science*, 593:117 – 131, 2015.