Auteursrechterlijke overeenkomst

Opdat de Universiteit Hasselt uw eindverhandeling wereldwijd kan reproduceren, vertalen en distribueren is uw akkoord voor deze overeenkomst noodzakelijk. Gelieve de tijd te nemen om deze overeenkomst door te nemen, de gevraagde informatie in te vullen (en de overeenkomst te ondertekenen en af te geven).

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling met

Titel: Graafkleuren
Richting: master in de informatica - databases Jaar: 2008

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Ik ga akkoord,

BRUNINX, Geert

Datum: 5.11.2008

# *Graafkleuren*

**Geert Bruninx**

promotor :
dr. Dries VAN DYCK

Eindverhandeling voorgedragen tot het bekomen van de graad
master in de informatica databases

universiteit hasselt

# Samenvatting

Het kleuren van grafen is een van de klassieke NP-complete problemen uit de grafentheorie met vele varianten en al even veel praktische toepassingen. Bij knoopkleuren, bijvoorbeeld, is het de bedoeling elke knoop een kleur toe te kennen uit een zo kleine mogelijke verzameling kleuren zodat geen enkele boog twee knopen verbind met eenzelfde kleur. Bij boogkleuren tracht men de bogen te kleuren met zo weinig mogelijk kleuren zodat geen twee bogen met dezelfde kleur elkaar ontmoeten in een knoop. Men kan ook zowel de bogen als de knopen kleuren. Het minimaal aantal kleuren vereist om de knopen te kleuren noemt met het chromatisch getal en voor de bogen de chromatische index. Er zijn zowel bovengrenzen als ondergrenzen gekend voor beide grootheden en hun waarde is een zeer belangrijke eigenschap van de graaf die vaak terugkomt bij andere problemen.

Voor deze masterproef betekent dit concreet het volgende. De voorgedragen eindverhandeling draagt de titel “Graafkleuren”. Een inhoudsopgave, die als leidraad dienst doet, geeft een overzicht van de verschillende hoofdstukken die het onderwerp behandelen. Deze vindt u na het ”Woord vooraf”. We beginnen in hoofdstukken 1 en 2 met een inleiding en begrippen die we nodig hebben om het graafkleurenprobleem te bestuderen. In hoofdstukken 3 en 4 bestuderen we het graafkleurenprobleem zelf. Om daarna in hoofdstuk 5 enkele algoritmes voor graafkleuren te bespreken. De implementaties, die voor deze masterproef gemaakt werden, van deze algoritmes en hun resultaten op een verzameling problemen worden besproken in hoofdstukken 7 en 8. Afsluitend vindt u de conclusie en de bronvermeldingen.

# Dankwoord

Ik zou mijn dankbaarheid willen uitdrukken voor alle personen van wie de permanente steun mij in staat heeft gesteld het huidige eindwerk tot een goed einde te brengen. Ik houd eraan in het bijzonder dr. Dries Van Dyck te bedanken, voor zijn hulp, zijn steun en zijn raadgevingen bij het tot stand komen van deze masterproef. Ik wens eveneens alle andere professoren en assistenten te bedanken die mij tijdens de volledige studieperiode begeleid hebben in hun respectievelijke cursussen. Verder wens ik ook mijn ouders, mijn zus en mijn medestudenten te bedanken voor hun steun, hun aanmoedigingen en hun luisterende oor.

# Woord vooraf

Zoals de traditie het voorschrijft, krijgt iedere student die zijn Master-diploma wilt behalen eerst de moeilijke taak voorgeschoteld een thesis, oftewel wetenschappelijke eindverhandeling, te maken. Na een eerste “eindwerk voorgedragen tot het behalen van de graad van bachelor in de informatica” is het nu de beurt aan de afsluitende “masterproef tot het behalen van de graad van master in de informatica”.
In dergelijk document probeert de student te demonstreren dat hij in staat is om een onderwerp uit te diepen, neer te schrijven en om te zetten in een eindproduct. De student staat er evenwel niet alleen voor. Hij kan terecht bij een promotor. Deze heeft een belangrijke sturende en motiverende rol voor de student.
In een eerste fase dient het onderwerp afgebakend te worden, in samenspraak met de promotor. Vervolgens is het de beurt aan de literatuurstudie en het bestuderen van de bronnen.
Het is na het vergaren van de nodige informatie dat de student aan de slag gaat en probeert zijn onderwerp in eigen bewoordingen aan te pakken.
En bij een masterproef in de informatica hoort natuurlijk ook het implementeren van het onderzoeksproduct, alsook enkele experimenten op dit product.

# Inhoudsopgave

# Hoofdstuk 1

# Inleiding

Deze thesis behandelt het graafkleurenprobleem. Dit is een van de klassieke NP-complete problemen uit de grafentheorie. Het graafkleurenprobleem heeft vele varianten en al even veel praktische toepassingen. Enkele voorbeelden hiervan zijn het toekennen van frequenties aan radiostations of mobiele telefoons, het toewijzen van computerregisters, timetabling en scheduling, testen van printplaten (=printed circuit board testing ) en pattern matching. Ook het populaire spel Sudoku is een vorm van graafkleuren.

Voor deze thesis werden enkele documenten als leidraad gebruikt. Dit zijn vooral “Graph Theory” van Reinhard Diestel[1], “Graph Coloring Algorithms” van Walter Klotz[2] en ”Heuristics for graph coloring” van Dominique de Werra[3].

Na deze inleiding beginnen we in hoofdstuk 2 met het opfrissen van enkele begrippen en het introduceren van enkele nieuwe die we zeker nodig hebben om het onderwerp “Graafkleuren” te bestuderen. In hoofdstukken 3 en 4 bestuderen we het graafkleurenprobleem zelf. Om daarna in hoofdstuk 5 enkele algoritmes voor graafkleuren te bespreken. De implementaties, die voor deze masterproef gemaakt werden, van deze algoritmes en hun resultaten op een verzameling problemen worden besproken in hoofdstukken 7 en 8. Afsluitend vindt u de conclusie en de bronvermeldingen.

# Hoofdstuk 2

# Basisbegrippen

In dit hoofdstuk willen we enkele basisbegrippen opfrissen en andere introduceren. Verder geven we de notaties die we gebruiken voor deze begrippen. We introduceren enkele termen en definities en brengen enkele andere in herinnering.
Hierbij nemen we als uitgangspunt het niveau van een student uit de masteropleiding in de informatica. De meeste definities en notaties worden op dezelfde manier gehanteerd in de bijbel der grafentheorie, "Graph theory" van Reinhard Diestel[1].

Een *ongerichte graaf G = (V, E)* is een geordend paar bestaande uit een verzameling *V* en een verzameling *E* van ongeordende paren $\{v_1, v_2\}$ elementen uit *V*.
Een *gerichte graaf G = (V, E)* is een geordend paar van een verzameling *V* en een verzameling *E* van geordende paren $(v_1, v_2)$ elementen uit *V*. Voor het vervolg van deze masterproef geldt er dat indien we spreken over grafen, tenzij anders vermeld, we spreken over ongerichte grafen.
De elementen van *V* heten de *knopen* van de graaf *G* en de elementen van *E* de *bogen* van *G*.

Twee verschillende knopen *x, y* heten *adjacent* als $\{x, y\} \in E$. We zeggen dan dat knopen *x* en *y buren* zijn. Als $\{x, y\} \notin E$ daarentegen, dan noemen we de knopen *x* en *y onafhankelijk*.
Een knoop *x* is *incident* met een boog *e* als x ∈ *e*; *x* heet dan een *eindknoop* van boog *e*.
Twee verschillende bogen $e_1, e_2 \in E$ zijn *adjacent* als ze een eindknoop gemeen hebben.

Een *simpele* graaf $G_S = (V_S, E_S)$ is een graaf waarbij er tussen twee knopen slechts één boog loopt en een boog altijd tussen twee verschillende bogen loopt. M.a.w.,

$$E_S \subseteq \{\{v_1, v_2\} | v_1, v_2 \} \in V \wedge v_1 \neq v_2\}$$

De *graad* van een knoop *x* in een graaf *G = (V, E)* is het aantal buren van x en wordt genoteerd als *d(x)*: $d(x) = \left|\{y \in V \,|\, \{x, y\} \in E\}\right|$.
Het getal $\delta(G) = min\{d(v) | v \in V\}$ is de *minimum graad* van graaf *G*, terwijl
$\Delta(G) = max\{d(v) | v \in V\}$ de *maximum graad* van graaf *G* is.

Een *wandeling* in een graaf *G = (V, E)* is een geordende rij knopen $x_0, x_1, x_2, \ldots, x_n$ zodat elk tweetal opeenvolgende knopen verbonden is: $x_0, x_1, \ldots x_n \in V$, niet noodzakelijk allemaal verschillend én $\{x_1, x_2\}, \{x_2, x_3\}, \ldots, \{x_{n-1}, x_n\} \in E$. We noemen $x_0$ en $x_n$, respectievelijk de begin- en eindknoop van de wandeling.
Een wandeling heet *gesloten* als begin- en eindknoop samenvallen. Anders heet de wandeling *open*.
De *lengte* van een wandeling is het aantal achtereenvolgende bogen.
Een wandeling heet een *pad* als de knopen in de wandeling allemaal verschillend zijn.
Een gesloten pad heet een *cykel*.

Een graaf G heet *samenhangend* als elk tweetal verschillende knopen x en y in G begin- en eindpunt zijn van een pad in G.

Een *boom* is een samenhangende graaf zonder cykels.
De knopen met graad 1 in een boom vormen de *bladeren*.

Een graaf heet *compleet* of *volledig* als elk tweetal verschillende knopen verbonden is.
$K_n$ is de notatie voor de volledige graaf met $n$ knopen.
Een graaf $G'= (V', E')$ is een *deelgraaf* van graaf $G = (V, E)$, genoteerd $G' \subseteq G$ als $V' \subseteq V$ én $E' \subseteq E$.
Het grootste getal $r$ zodat $K_r \subseteq G$ is het *kliekgetal* $\omega(G)$ van graaf $G$ en $K_r$ noemen we een *r-kliek* in G.

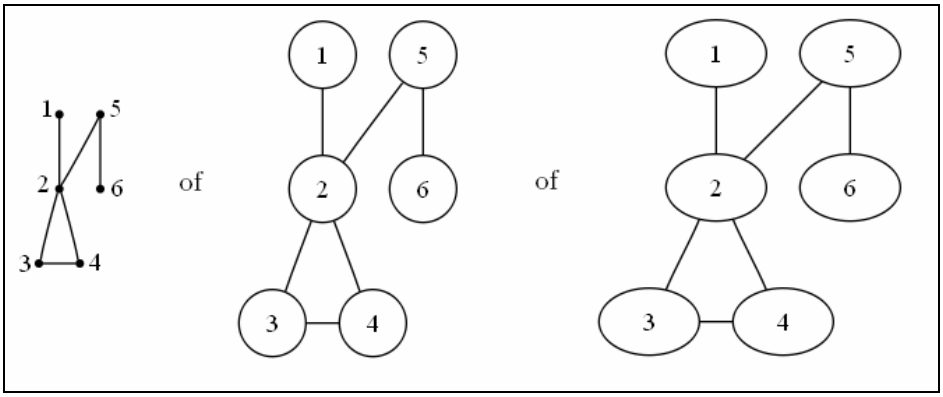
Een graaf $G = (V,E)$ heet *bipartiet* als $V$ een opsplitsing heeft in twee niet-lege disjuncte deelverzamelingen $V_1$ en $V_2$, zodat er enkel bogen bestaan tussen de knopen van de twee deelverzamelingen. Er zijn dus geen bogen tussen knopen van $V_1$ onderling of tussen knopen van $V_2$ onderling: $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$, $\forall \{x, y\} \in E : (x \in V_1 \wedge y \in V_2) \vee (x \in V_2 \wedge y \in V_1)$

De *lijngraaf L(G)* van graaf G is de graaf met een knoop voor iedere boog uit $G$. Twee knopen van L(G) zijn adjacent als en slechts als de corresponderende bogen in G adjacent waren, m.a.w. $L(G) = (\{v_e | e \in E\}, \{\{v_{e_1}, v_{e_2}\} | \{e_1, e_2\} \in E \wedge e_1 \cap e_2 \neq \emptyset\})$.

Een *merge*, genoteerd met $G / x, y$, van onafhankelijke knopen $x$ en $y$ is een operatie op een graaf $G$. Het resultaat van de merge $G / x, y$ is een nieuwe graaf waarin knopen $x$ en $y$ samengesmolten, geïndentificeerd zijn tot een knoop $v_{x,y}$. De buren van $v_{x,y}$ zijn dan de buren van $x$ en de buren van $y$. M.a.w.,

$$G / x, y = (V \setminus \{x, y\} \cup \{v_{x,y}\}, E \setminus \{\{v, w\} \in E \mid \{v, w\} \cap \{x, y\} \neq \emptyset\} \cup \{\{v_{x,y}, w\} | \{x, w\} \in E\} \setminus \{\{y, w\} \in E)$$

Het is ook uiterst intuïtief om van een graaf een voorstelling te maken. Dit gebeurt gewoonlijk door voor iedere knoop een punt, cirkel of ellips te tekenen. Twee punten worden met elkaar verbonden met een lijn als de twee corresponderende knopen een boog vormen.




**Figuur 2.1: Graaf G=({1, 2, 3, 4, 5, 6}, {(1, 2), (2, 3), (2, 4), (2, 5), (3, 4), (5,6)})**

Een graaf G heet *planair* als we de graaf kunnen tekenen in het vlak zodat de bogen elkaar niet snijden of raken, behalve in een gemeenschappelijk eindpunt.

# Hoofdstuk 3

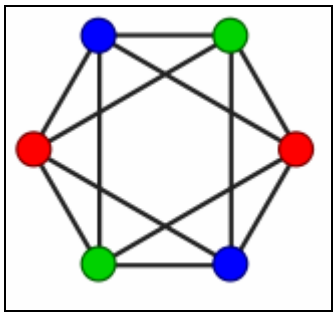
# Graafkleuren

## 3.1 Wat is graafkleuren?

Het kleuren van grafen is een van de klassieke NP-complete problemen uit de grafentheorie met vele varianten en al even veel praktische toepassingen. Informeel betekent het kleuren van een graaf: het kleuren van elementen van de graaf zodanig dat aangrenzende elementen met verschillende kleuren worden gekleurd. We schrijven bewust elementen van de graaf, want het is mogelijk om zowel de knopen van de graaf, als de bogen van de graaf, of zelfs de combinatie van de twee te kleuren. Met als handleiding de lessen van Alexander Schrijver[4], leggen we precies uit wat knoopkleuren is.

## 3.2 Knoopkleuren

Knoopkleuren is het toekennen van kleuren aan de knopen van de graaf, zodanig dat adjacente knopen met verschillende kleuren worden gekleurd.



**Figuur 3.1: Graaf met 3 kleuren**
**Bron: Wikimedia Commons**

Bij de bestudering van een kleuring, blijkt echter ook dat het niet uitmaakt welke kleuren er precies worden gebruikt. Het is alleen van belang dat buren niet dezelfde kleur krijgen. Dus of onze kleurverzameling nu {rood, groen, geel, blauw} is, of {oranje, paars, bruin, grijs}, dat maakt wiskundig voor het probleem geen verschil. We kunnen de kleuren dus ook gewoon een nummer geven. Daarvoor gebruiken we een abstracte verzameling K van kleuren.
$K = \{1, 2, 3, \ldots, k\}$

Een geldige knoopkleuring van een graaf $G = (V, E)$ is een functie $F: V \rightarrow K$ met volgende eigenschap

$$als\ \{v, w\} \in E\ dan\ F(v) \neq F(w)$$

Hierbij wordt niet geëist dat alle kleuren uit $K$ ook echt gebruikt worden. Als je voldoende kleuren tot je beschikking hebt, zou je bijvoorbeeld alle punten een verschillende kleur kunnen geven. Een graaf $G$ met $n$ knopen is dus altijd met $n$ kleuren te kleuren. De uitdaging schuilt erin om zo weinig mogelijk kleuren te gebruiken.
Het minimaal aantal kleuren nodig om een graaf geldig te kleuren, is een intrinsieke eigenschap van de graaf en wordt het *chromatisch getal* genoemd. Het chromatisch getal van graaf $G$ wordt genoteerd met $\chi(G)$. Het knoopkleurenprobleem is dus een combinatorisch

optimalistatieprobleem dat erin bestaat een kleuring te vinden die een minimaal aantal kleuren gebruikt.

Een graaf G heet *k-kleurbaar* als $\chi(G) \leq k$ , en heet *k-chromatisch* als $\chi(G) = k$ .

Wanneer een graaf *G* k-kleurbaar is, betekent dit dat de verzameling knopen *V* te verdelen is in disjuncte deelverzamelingen van onafhankelijke knopen, $P_1, P_2, P_3, \ldots, P_k$, ook partities genoemd Deze partities worden *kleur-klassen* genoemd. Een kleur-klasse van een kleuring *F* bevat knopen die met eenzelfde kleur gekleurd zijn.

## 3.3 Begrenzingen van het chromatisch getal

Omdat het vinden van het chromatisch getal van een graaf een zeer moeilijk probleem is, zoals we verder in hoofdstuk 4 bespreken, zou het handig zijn om toch enige begrenzingen te kennen voor de waarde van dit getal $\chi(G)$. Er zijn zowel bovengrenzen als ondergrenzen van $\chi(G)$ gekend:

**Stelling 3.1:** Voor iedere graaf *G* met *m* bogen geldt er dat:

$$\chi(G) \leq \frac{1}{2} + \sqrt{2m + \frac{1}{4}}$$

**Bewijs:** Veronderstel een kleuring *F* van graaf *G* met k = $\chi(G)$ kleuren. Dan heeft *G* ten minste één boog tussen elke twee kleur-klassen. Als hij dat niet heeft, zou dezelfde kleur gebruikt kunnen worden voor beide kleur-klassen. Daarom geldt, $m \geq \frac{1}{2}k(k-1)$. Het oplossen of uitwerken van deze vergelijking naar variabele *k* levert de beweerde begrenzing.□

**Stelling 3.2:** $\chi(G) \leq \Delta(G) + 1$

Rangschik de knopen van de graaf in een willekeurige orde $v_1, v_2, \ldots, v_n$
Kleur nu in volgorde iedere knoop $v_i$ met de eerste beschikbare kleur, d.w.z. met het kleinste positieve getal dat nog niet gebruikt werd voor de reeds gekleurde buren $v_1, \ldots, v_{i-1}$ van knoop $v_i$ . Op deze manier gebruiken we nooit méér dan $\Delta(G)+1$ kleuren, zelfs voor de meest ongunstige keuze van orde $v_1, v_2, \ldots, v_n$ □

**Stelling 3.3:** $\chi(G) \geq \omega(G)$

**Bewijs:** In paragraaf 3.6 behandelen we enkele speciale grafen. Hieronder ook de complete graaf. We weten dat we voor het kleuren van zo'n complete graaf met *n* knopen, ook *n* kleuren nodig hebben. Het kliekgetal $\omega(G)$ is daarom een ondergrens voor het chromatisch getal, omdat alle knopen van de kliek een verschillende kleur moeten krijgen in een geldige kleuring. □

De volgende stelling geven we zonder bewijs, omdat het te technisch is en buiten het kader van dit proefschrift valt. We verwijzen voor het bewijs naar Diestel[1].

**Stelling 3.4:** De stelling van Brooks(1941) : Voor een samenhangende graaf G geldt, tenzij G een complete graaf is of een oneven cykel heeft:

$$\chi(G) \leq \Delta(G)$$

## 3.4 Boogkleuren

Net zoals bij knoopkleuren, moeten we kleuren toekennen, maar dit keer, zoals de naam al aangeeft, aan de bogen van de graaf. Daarbij mogen bogen, die elkaar ontmoeten in een knoop, niet eenzelfde kleur hebben.

Een geldige boogkleuring van een graaf $G = (V, E)$ is een functie $f: E \rightarrow K$ met volgende eigenschap:

$$\begin{aligned} &\text{als } e_1 = \{u, v\}; \quad e_2 = \{u, w\}; \quad e_1, e_2 \in E \\ &\text{dan } f(e_1) \neq f(e_2) \\ &\text{met } K = \{1, 2, 3, \ldots, k\} \end{aligned}$$

Het minimale aantal kleuren dat nodig is om de bogen van G te kleuren wordt de *chromatische index* van graaf *G* genoemd en wordt genoteerd met $\chi'(G)$.

De chromatische index van een graaf *G* is gelijk aan het chromatische getal van zijn lijngraaf *L(G):*

$$\chi'(G) = \chi(L(G))$$

Ook voor de chromatische index zijn enkele begrenzingen gekend:
**Stelling 3.5:** Voor alle grafen geldt:

$$\chi'(G) \geq \Delta(G)$$

**Stelling 3.6:** Stelling van Vizing[5]: voor simpele grafen *G* met maximum graad $\Delta(G)$ geldt:

$$\Delta(G) \leq \chi'(G) \leq \Delta(G) + 1$$

**Stelling 3.7:** Stelling van Shannon[6]: $\Delta(G) \leq \chi'(G) \leq \frac{3}{2}\Delta(G)$

**Stelling 3.8:** Stelling van König[7]: voor bipartiete grafen G geldt:

$$\chi'(G) = \Delta(G)$$

**Stelling 3.9:** Complete grafen voldoen aan[8]:

$$\chi'(K_n) = \begin{cases} n-1 & \text{als n even} \\ n & \text{als n oneven} \end{cases}$$

## *3.5 Andere varianten*

Er bestaan nog vele andere varianten van graafkleuren. De volgende zijn de meest gekende en meest bestudeerde varianten:

Lijst-knoopkleuren:

iedere knoop heeft een lijst met kleurmogelijkheden waaruit gekozen kan worden voor de kleurtoekenning

Lijst-boogkleuren:

iedere boog heeft een lijst met kleurmogelijkheden waaruit gekozen kan worden voor de kleurtoekenning

Totaal kleuren:

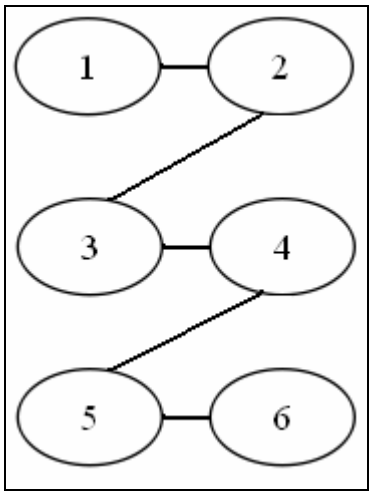
zowel de knopen als de bogen worden gekleurd, grenzende knoop-knoop, boog-boog en knoop-boog combinaties krijgen een verschillende kleur

## *3.6 Speciale grafen*

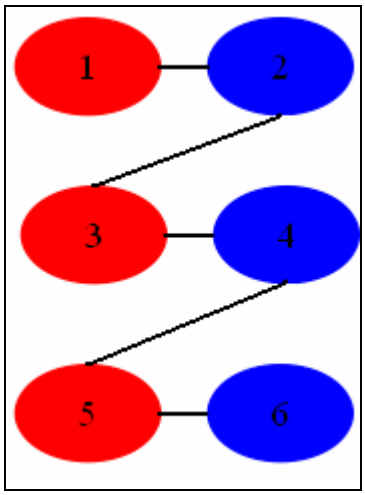
### 3.6.1 Bipartiete graaf / Boom

Alle bipartiete grafen, en bijgevolg alle bomen zijn 2-kleurbaar. Ook geldt dat alle grafen die 2-kleurbaar zijn, bipartiet zijn. (zie figuren 3.2 t/m 3.5)

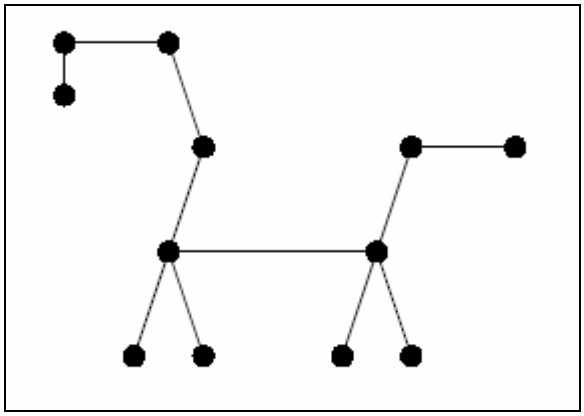
*Graaf G is bipartiet* $\Leftrightarrow$ *G is 2-kleurbaar*



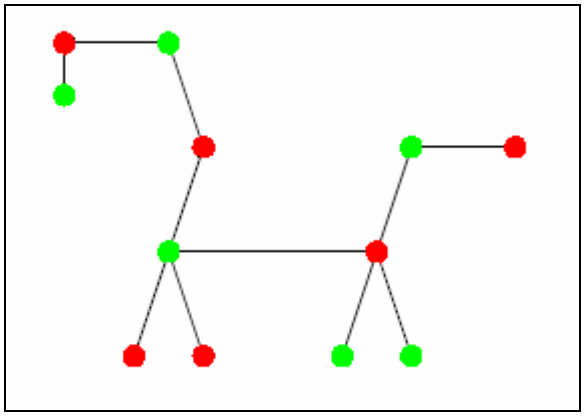

**Figuur 3.2: Bipartiete graaf**




**Figuur 3.3: Bipartiete graaf ingekleurd**



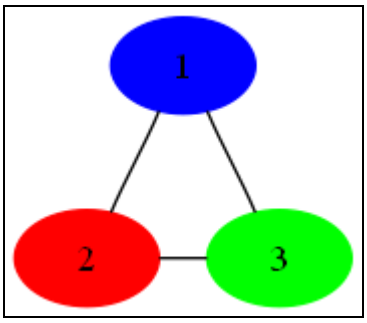
**Figuur 3.4: Boom**



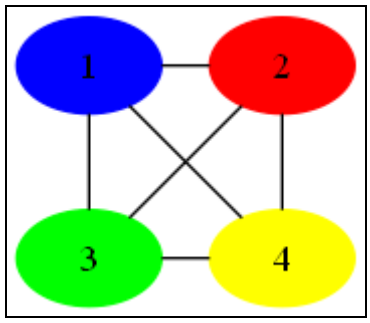
**Figuur 3.5: Boom uit figuur 3.3 ingekleurd**

### 3.6.2 Complete graaf

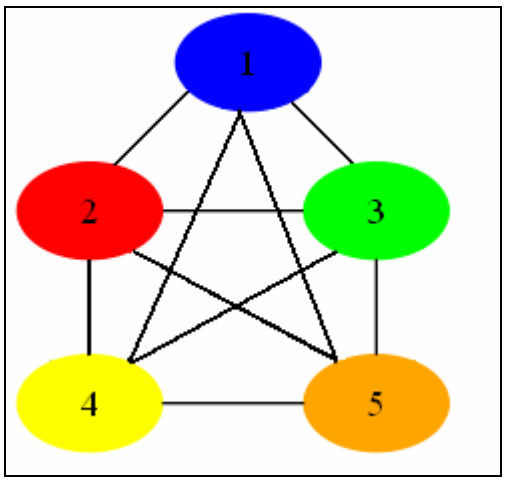
Een complete graaf met *k* knopen is *k*-chromatisch. (zie figuren 3.6 t/m 3.8)




**Figuur 3.6: Complete graaf met 3 knopen**




**Figuur 3.7: Complete graaf met 4 knopen**

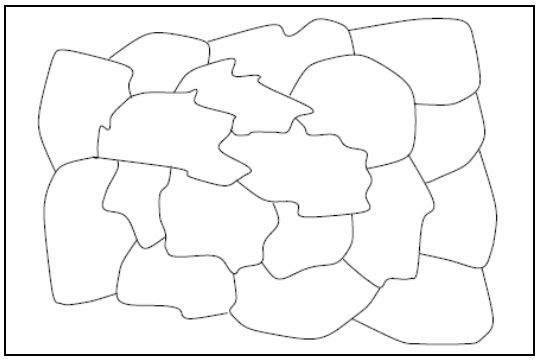



**Figuur 3.8: Complete graaf met 5 knopen**

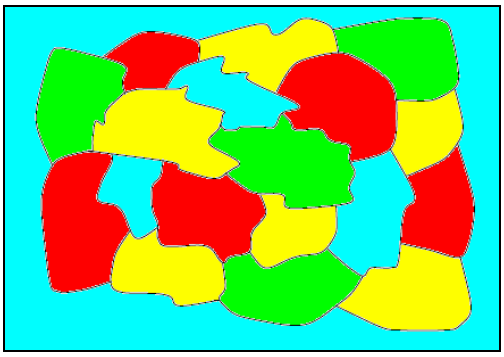
### 3.6.3 Planaire graaf

De vierkleurenstelling[9], één van de bekendste stellingen in de wiskunde, toont aan dat het mogelijk is elke willekeurige landkaart (mits enkele voorwaarden), met behulp van slechts vier kleuren zo in te kleuren dat geen twee aangrenzende landen dezelfde kleur krijgen (zie figuren 3.9 en 3.10). We kunnen zo'n kaartkleuring echter ook bestuderen met behulp van grafen. Laten we elk land voorstellen door een knoop, en de knopen van twee aaneengrenzende landen met elkaar verbinden door een boog (zie figuren 3.11 en 3.12). Deze graaf is bovendien steeds planair.
Het kleuren van de landkaart correspondeert dan met het kleuren van de knopen van de bekomen graaf. Bijgevolg stellen we dat planaire grafen 4-kleurbaar zijn.

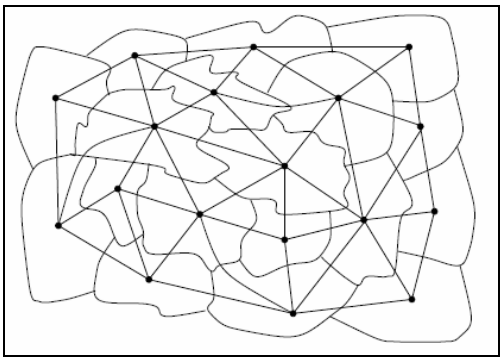
$$\textit{Graaf } G \textit{ is planair} \Rightarrow \chi(G) \leq 4$$



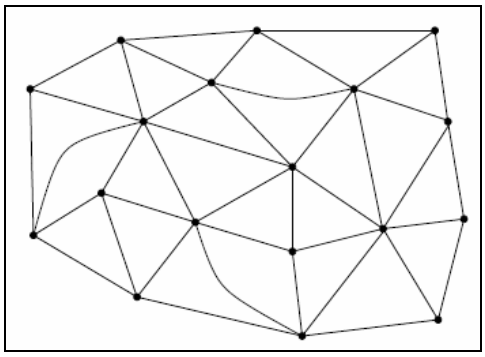
**Figuur 3.9: Landkaart [4]**



**Figuur 3.10: Landkaart ingekleurd met 4 kleuren**



**Figuur 3.11: Van landkaart naar graaf [4]**



**Figuur 3.12: Corresponderende graaf [4]**

# Hoofdstuk 4

# Algoritmes voor graafkleuren

## 4.1 Complexiteit

De complexiteit van een probleem wordt typisch uitgedrukt in termen van een beslissingsprobleem: Beslis of "iets" $\in$ van een taal $L$. In ons geval is dus $L = \{<G, k> \mid G$ is een k-kleurbare graaf$\}$

Het beslissingsprobleem Graafkleuring($G$, $k$) vraagt of er een kleuring van graaf $G$ bestaat die hoogstens $k$ kleuren gebuikt. In 1972 bewees Richard M. Karp de NP-compleetheid van dit probleem in zijn gerenommeerd paper "Reducibility Among Combinatorial Problems"[10]. Het graafkleurenprobleem stond onder de naam "chromatic number", de Engelse vertaling van chromatisch getal, in zijn lijstje van 21 problemen waarvan hij de NP-compleetheid aantoonde. (zie figuur 4.1) Het graafkleurenprobleem beperkt zich echter niet tot enkel het beslissingsprobleem. We moeten ook de parameter $k$ optimaliseren, aangezien we op zoek zijn naar de kleinste $k$ waarvoor dit geldt. Het graafkleurenprobleem is dus het combinatorisch optimalisatieprobleem Graafkleuring(G, k) $\wedge$ $\neg$Graafkleuring(G, k-1). Het linkerdeel van het probleem is NP-compleet, hetgeen betekent dat het rechterdeel coNP-compleet is.

Het vinden van het chromatische getal van een graaf G is een NP-hard probleem. Dit betekent dat het graafkleurenprobleem behoort tot die klasse van problemen die minstens zo moeilijk op te lossen zijn als de moeilijkste problemen in NP.

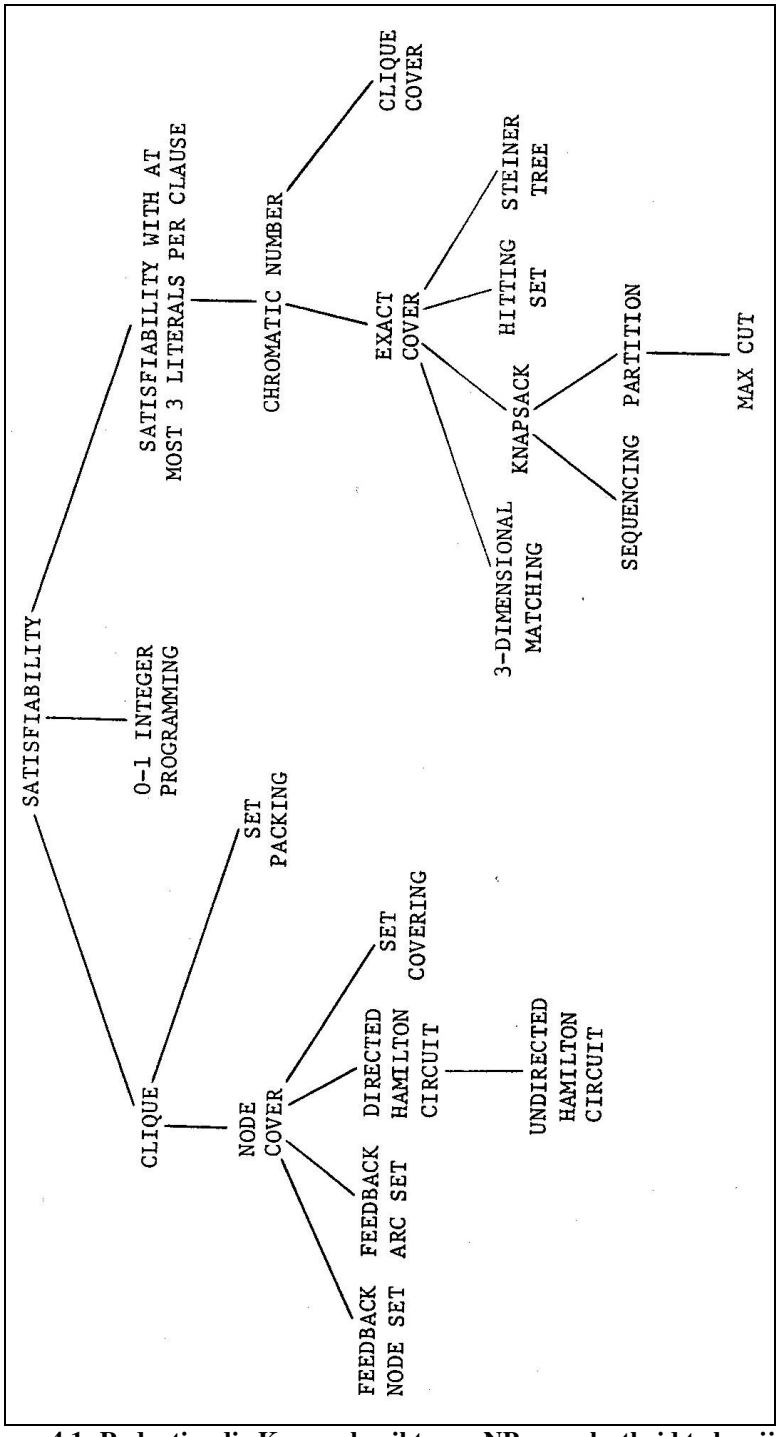
Dat het graafkleurenprobleem behoort tot de klasse NP-hard, vertelt ons dat het (in vele gevallen) zeer tijdrovend is om de optimale oplossing te vinden. We moeten ons daarom behelpen met heuristieken. Heuristieken geven niet altijd de beste oplossing. Maar ze berekenen wel snel een benadering, dat hopelijk in de buurt van de optimale oplossing ligt.

## 4.2 Heuristieken

In hoofdstuk 5 en 6 zullen we enkele heuristische algoritmes bespreken, merge-algoritmes in hoofdstuk 5 en sequetiële algoritmes in hoofdstuk 6. Het grote verschil tussen deze twee soorten is dat de eerste groep niet onmiddellijk een kleur gaat toekennen aan knopen, maar wel bepaald welke knopen dezelfde kleur mogen krijgen. De tweede groep gaat tijdens de uitvoering wel onmiddellijk kleuren toekennen aan de knopen.

Met heuristieken is het zo dat de aandacht vaak gericht wordt op de snelheid. Het gebeurt dan ook vaak dat de criteria die de algoritmes toepassen, een niet-deterministische uitvoering van het algoritme veroorzaken. We kunnen van deze feiten gebruik maken om meerdere uitvoeringen van het algoritme te doen op eenzelfde probleeminstantie en het beste resultaat van alle runs te onthouden. Dit kan op twee manieren gebeuren: Ofwel maken we een willekeurige keuze als er zich meerdere evenwaardige kandidaten zijn die voldoen aan de

criteria. Ofwel passen we vóór iedere run van het algoritme de labeling/nummering van de knopen op willekeurige manier aan. Deze twee manieren verhelpen het probleem dat zou kunnen ontstaan als het algoritme steeds, slechts op basis van de labeling van de knoop, dezelfde knoop zou kiezen bij meerdere evenwaardige kandidaten.




**Figuur 4.1: Reducties die Karp gebruikte om NP-compleetheid te bewijzen**

# Hoofdstuk 5

# Merge-algoritmes (Zykov-algoritmes)

## 5.1 Oorsprong

De idee achter de merge-algoritmes is afkomstig uit een stelling van A. A. Zykov [11]. Deze formuleerde in 1952 het volgende:

$\chi(G) = \min\{\chi(G/x, y), \chi(G + xy)\}$ voor onafhankelijke knopen $x$ en $y$ van graaf $G$.

**Bewijs***:* Neem *C(G)* als de verzameling van alle geldige kleuringen van de graaf $G$ en $|K|$ het aantal (verschillende) kleuren gebruikt door één zo'n kleuring $K \in C(G)$

Er geldt dat
$$\begin{aligned}\chi(G) &= \min\{|K| : K \in C(G)\} \\ &= \min\{\min\{|K| : K(x) = K(y)\}, \min\{|K| : K(x) \neq K(y)\}\} \\ &= \min\{\chi(G/x, y), \chi(G + xy)\} \quad \square\end{aligned}$$

Door het meermaals toepassen van deze stelling, construeren we een Zykov-boom. Een Zykov-boom is een binaire boom van grafen. De wortel van de Zykov-boom is de originele graaf $G$. Het linkerkind is steeds de graaf $G/x,y$ en het rechterkind is de graaf $G + xy$. De constructie van een Zykov-boom is volledig wanneer er geen verdere reductie volgens de stelling van Zykov mogelijk is. De bladeren van de Zykov-boom bevatten dan complete grafen $G_i=(V_i, E_i)$. De stelling van Zykov betekent dus dat $\chi(G) = min|V_i|$.

Een Zykov-boom wordt niet uniek bepaald, maar hangt af van de volgorde waarin onafhankelijke knoopparen $x,y$ worden gekozen. Wel heeft elke Zykov-boom precies één tak die uitsluitend door merges (samentrekkingen) wordt geproduceerd. Het uiteinde (=blad) van deze tak bevat een complete graaf $G_C$. Het aantal knopen van $G_C$ vormt een bovengrens voor $\chi(G)$.

Hoe langer de tak is die graaf $G_C$ bekomt, en bijgevolg hoe meer merges (*samentrekkingen*) er gebeuren, des te beter deze begrenzing zal zijn. Daarom lijkt het een goede strategie om eerst die onafhankelijke knopen $x,y$ te mergen, zodat er zoveel mogelijk onafhankelijke knoopparen overblijven. Dit is ook de opzet van het eerste merge-algoritme dat we bespreken, het Dutton & Brigham-algoritme[12].

## 5.2 Dutton & Brigham algoritme

We weten dus dat we door zo veel mogelijk merges te doen, een betere begrenzing van het chromatische getal krijgen. In het Dutton & Brigham-algoritme, genoemd naar zijn bedenkers, proberen we dit te doen door steeds de twee onafhankelijke knopen met het hoogste aantal gemeenschappelijke buren te mergen.

### 5.2.1 Algoritme

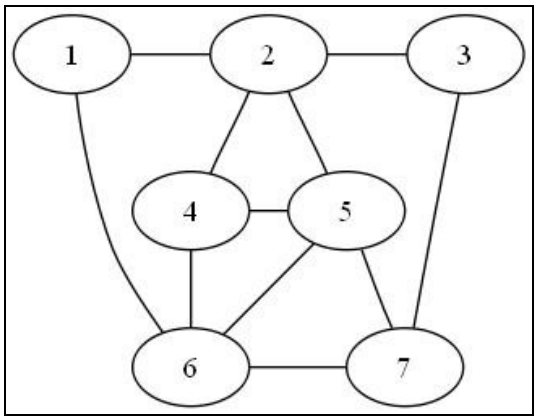
Zolang er onafhankelijke knopen zijn in graaf $G$, worden de volgende stappen herhaald:

a) bereken voor ieder paar onafhankelijke knopen $v_i$, $v_j$ het aantal gemeenschappelijke buren $c_{ij}$
b) bepaal het paar $v_r$, $v_s$ waarvoor $c_{rs}$ maximaal is
c) merge $v_r$ en $v_s$

Op die manier vinden we in iedere stap een paar knopen $v_r$ en $v_s$, dat dezelfde kleur krijgt in de kleuring die we construeren. Wanneer er geen onafhankelijke knopen meer zijn, bekomen we een complete graaf $K_p$. Deze complete graaf is gemakkelijk te kleuren. Iedere knoop moet namelijk een andere kleur krijgen. We weten ook welke kleuren de andere knopen van de originele graaf krijgen, namelijk de kleur van de knoop waarmee ze gemergd werden. We hebben daarmee een p-kleuring voor de originele graaf $G$ geconstrueerd.

### 5.2.2 Voorbeeld

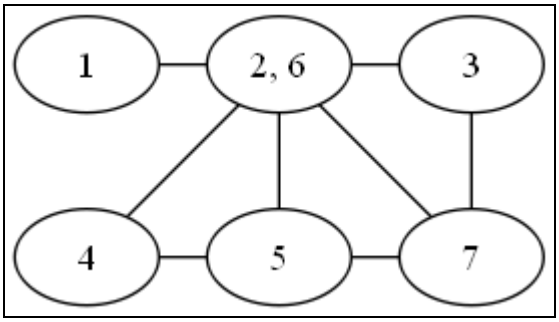
We overlopen de acties van het Dutton & Brigham algoritme op de graaf uit figuur 5.1.
We tonen het aantal gemeenschappelijke buren van de onafhankelijke knopen in de tabellen 5.1 t/m 5.4. De gemarkeerde waarde is de hoogste, of degene die we kozen in geval van meerdere paren met hetzelfde grootste aantal. De figuren 5.2 t/m 5.5 tonen de grafen na iedere merge.
Zoals we zien in tabel 5.2, hebben de knopen 2 en 6 het grootste aantal gemeenschappelijke buren. Deze knopen worden daarom als eerste gekozen om te mergen. Na het mergen van knoop 2 en knoop 6 bekomen we de graaf in figuur 5.2. Het zoeken naar het volgende paar (zie tabel 5.2) levert meerdere knoopparen op die in aanmerking komen om als volgende te mergen. We kiezen willekeurig voor knopen 3 en 5, en bekomen de graaf in figuur 5.3. Wanneer we deze stappen blijven herhalen (zie figuren 5.3 en 5.4, tabellen 5.4 en 5.5), bekomen we uiteindelijk een complete graaf $K_3$, te zien in figuur 5.5. Deze is makkelijk in te kleuren met 3 kleuren. Stap voor stap kunnen we dan de originele graaf kleuren, door 2 knopen die we zonet mergden dezelfde kleur te geven. Het hele proces wordt nog even snel samengevat in figuren 5.6 en 5.7.
We merk ook nog even op dat de bekomen kleuring optimaal is, aangezien het kliek-getal van de graaf drie is, en $\chi(G) \geq \omega(G)$ (cfr. Stelling 3.3).




**Figuur 5.1: Graaf G**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | - | - | 1 | 2 | 2 | - | 1 |
| 2 | - | - | - | - | - | 3 | 2 |
| 3 | 1 | - | - | 2 | 2 | 1 | - |
| 4 | 2 | - | 2 | - | - | - | 2 |
| 5 | 2 | - | 2 | - | - | - | - |
| 6 | - | 3 | 1 | - | - | - | - |
| 7 | 1 | 2 | - | 2 | - | - | - |

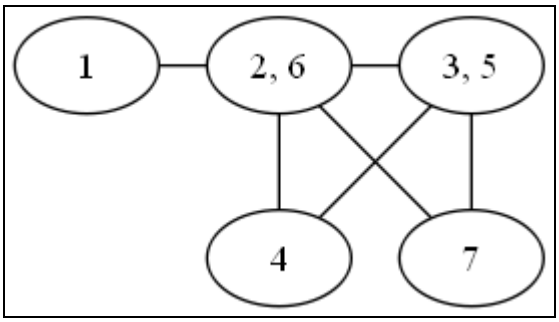
**Tabel 5.1: Gemeenschappelijke buren van de onafhankelijke knopen uit graaf G van figuur 5.1**




**Figuur 5.2: Nieuwe graaf G = G/ 2,6**

| | 1 | 2, 6 | 3 | 4 | 5 | 7 |
|---|---|---|---|---|---|---|
| 1 | - | - | 1 | 1 | 1 | 1 |
| 2, 6 | - | - | - | - | - | - |
| 3 | 1 | - | - | 1 | 2 | - |
| 4 | 1 | - | 1 | - | - | 2 |
| 5 | 1 | - | 2 | - | - | - |
| 7 | 1 | - | - | 2 | - | - |

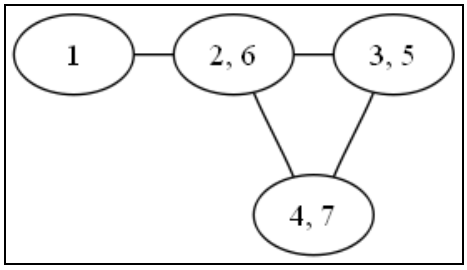
**Tabel 5.2: Gemeenschappelijke buren van de onafhankelijke knopen uit graaf G van figuur 5.2**




**Figuur 5.3: Nieuwe graaf G = G / 3, 5**

| | 1 | 2, 6 | 3, 5 | 4 | 7 |
|---|---|---|---|---|---|
| 1 | - | - | 1 | 1 | 1 |
| 2, 6 | - | - | - | - | - |
| 3, 5 | 1 | - | - | - | - |
| 4 | 1 | - | - | - | 2 |
| 7 | 1 | - | - | 2 | - |

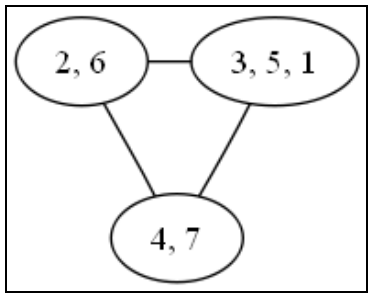
**Tabel 5.3: Gemeenschappelijke buren van de onafhankelijke knopen uit graaf G van figuur 5.3**



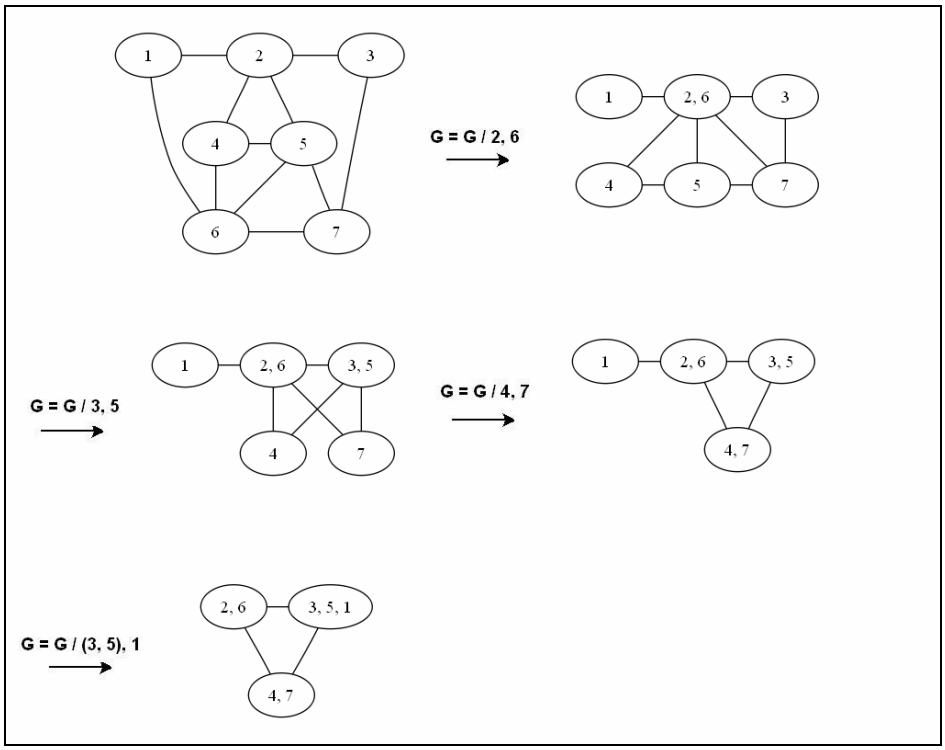

**Figuur 5.4: Nieuwe graaf G = G / 4, 7**

| | 1 | 2, 6 | 3, 5 | 4, 7 |
|---|---|---|---|---|
| 1 | - | - | 1 | 1 |
| 2, 6 | - | - | - | - |
| 3, 5 | 1 | - | - | - |
| 4, 7 | 1 | - | - | - |

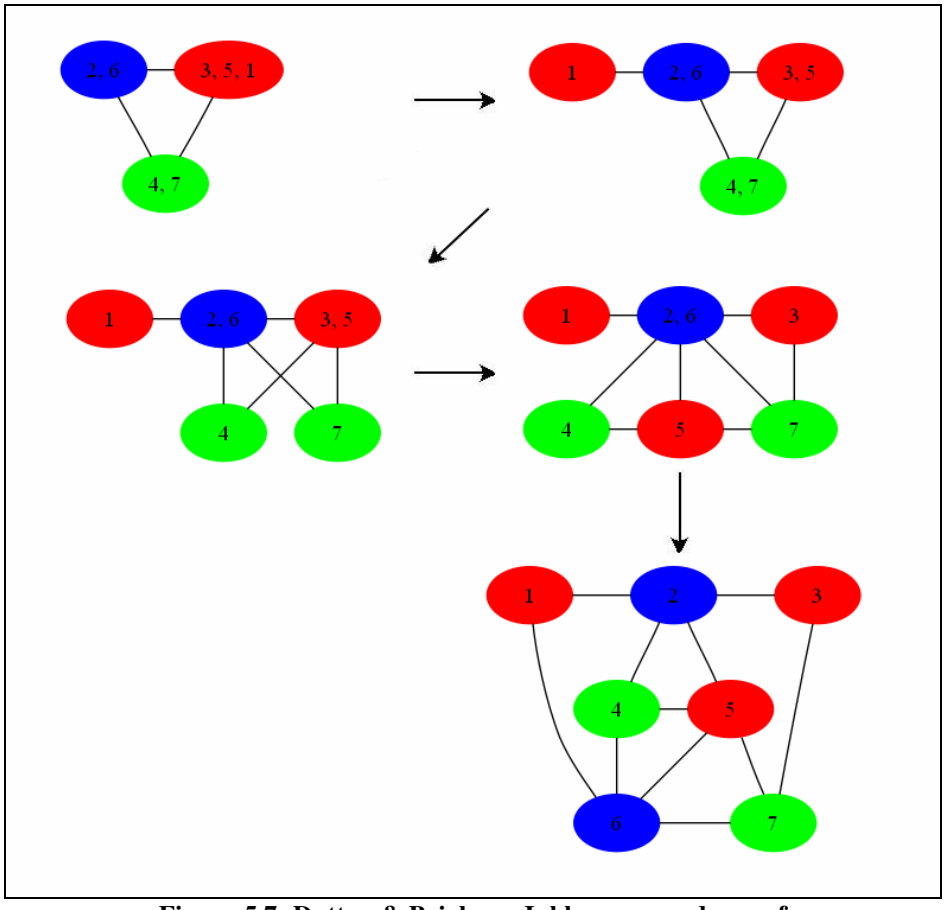
**Tabel 5.4: Gemeenschappelijke buren van de onafhankelijke knopen uit graaf G van figuur 5.4**




**Figuur 5.5: Nieuwe Graaf G = G / (3, 5), 1**




**Figuur 5.6: Overzicht merges van Dutton & Brigham-algoritme op graaf G uit figuur 5.1**




**Figuur 5.7: Dutton & Brigham: Inkleuren van de graaf**

### 5.2.3 Pseudocode

De implementatie van het Dutton & Brigham-algoritme kunnen we op informele wijze beschijven met de pseudocode uit tabel 5.5.

### 5.2.4 Tijdscomplexiteit

Het Dutton & Brigham algoritme kan geïmplementeerd worden in $O(n^3)$ waarbij *n* het aantal knopen is. Dit is namelijk de tijd die we nodig hebben om voor ieder paar knopen(=$O(n^2)$) het aantal gemeenschappelijke buren(=O(n)) te berekenen. Wanneer we een tabel bijhouden met het aantal gemeenschappelijke buren van ieder paar onafhankelijke knopen, dan neemt de hoofdlus van het Dutton & Brigham algoritme ook $O(n^3)$ in beslag. We overlopen namelijk in worst-case de while-lus *n* keer. In die lus duurt het zoeken naar het paar met het grootste aantal gemeenschappelijke buren $O(n^2)$, de merge-operatie O(n) en het updaten van de tabel $O(n^2)$.

| Dutton & Brigham-algoritme |
|---|
| *Invoer*: graaf G  *Uitvoer*: een kleuring van G<br><br>Initialiseer:<br>  tabel T met voor ieder paar onafhankelijke knopen het aantal gemeenschappelijke buren<br>  stack S = Ø  om gemergde knopen bij te houden<br><br>*while* G ≠ compleet<br>  zoek het paar onafhankelijke knopen $v_r$ , $v_s$ met het grootste aantal gemeenschappelijke buren<br>  G = G / $v_r$ , $v_s$<br>  push koppel $v_r$ , $v_s$ op stack S<br>  update tabel T<br><br>Geef iedere knoop in de complete graaf een verschillende kleur<br><br>while stack S ≠ Ø<br>  pop koppel x, y van de stack S<br>  geef knoop y dezelfde kleur als knoop x<br><br>*return* kleuring K |

**Tabel 5.5: Pseudocode Dutton & Brigham**

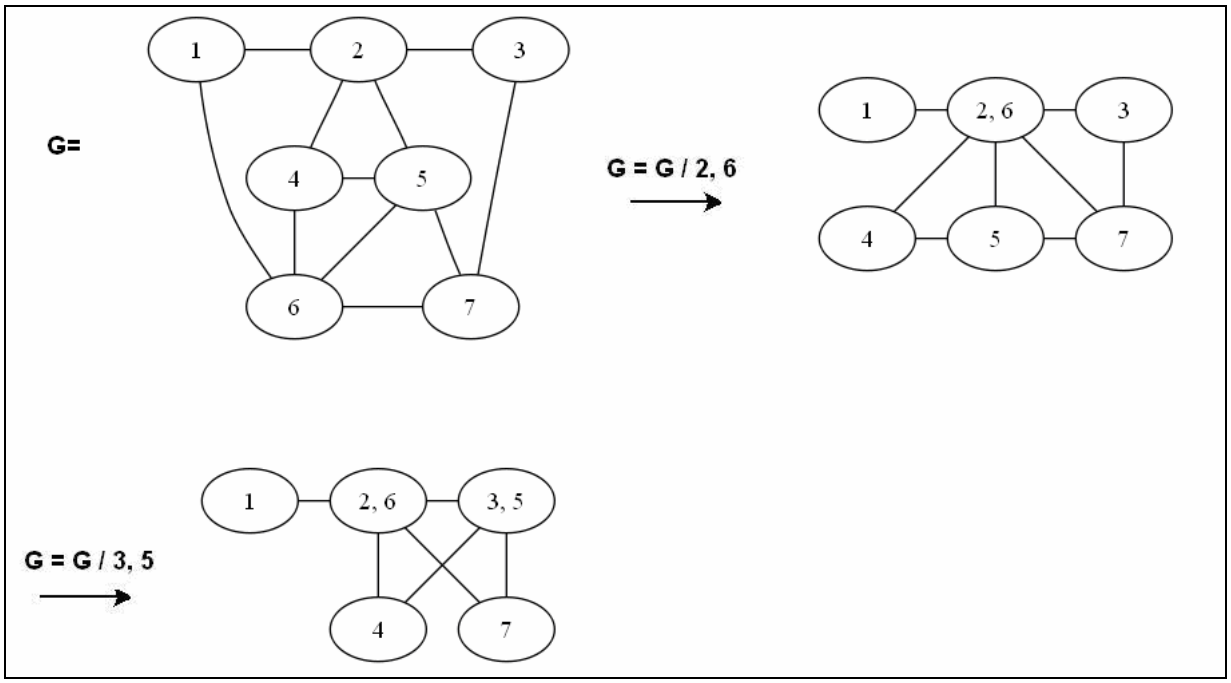
## 5.3 Cosine-algoritme

### 5.3.1 Algoritme en tijdscomplexiteit

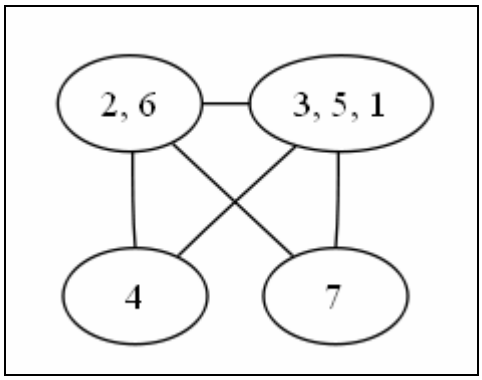
Een variatie op het Dutton & Brigham algoritme werd geïntroduceerd door Alain Hertz[13]. Hij kwam met de idee om zo lang mogelijk dezelfde knoop $v_r$ te gebruiken in de merge-acties. De opzet is om steeds de knoop $v_r$ met zijn onafhankelijk knopen te mergen totdat $v_r$ adjacent is aan alle knopen. In vele gevallen kan het proces zo (aanzienlijk) versneld worden. Dit is omdat we in de lusdoorgang niet altijd op zoek gaan naar een paar onafhankelijke knopen met het grootste aantal gemeenschappelijke buren, hetgeen in $O(n^2)$ gebeurt. Maar we zoeken, zo vaak als mogelijk is, naar de knoop $v_s$ die het meeste gemeenschappelijke buren heeft met $v_r$, onder alle onafhankelijke knopen van $v_r$; wat in O(n) kan. De complexiteit van het Cosine-algoritme is, zoals het Dutton & Brigham algoritme, $O(n^3)$ en is dus theoretisch gezien gelijk. In de praktijk zal het Cosine-algoritme echter sneller presteren dan zijn voorganger van Dutton & Brigham. Deze bewering zullen we in hoofdstuk 8 ook aantonen met de resultaten van de experimenten. Ook zullen we in de resultaten opletten of het Cosine-algoritme 'slechtere' kleuringen construeert, d.w.z. kleuringen waarin we meer kleuren nodig hebben. Hetgeen wel in de verwachtingen ligt omdat we niet steeds de twee knopen met het hoogste aantal gemeenschappelijke buren in de graaf mergen.

### 5.3.2 Voorbeeld

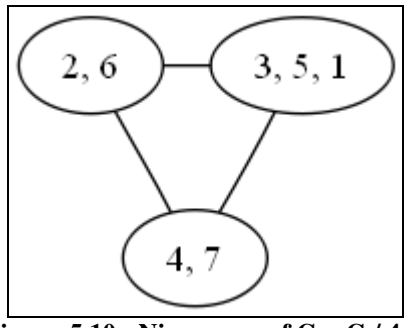
We gebruiken hier dezelfde graaf (zie figuur 5.1). als in het voorbeeld van het Dutton & Brigham-algoritme. We willen vooral laten zien waar het Cosine-algoritme sneller zal presteren als het Dutton & Brigham-algoritme.
In figuur 5.8 zien we de eerste stappen van het Cosine-algoritme, die dezelfde zijn als het Dutton & Brigham-algoritme. Dit komt omdat na de eerste merge, de knoop "2, 6" reeds adjacent is met alle andere knopen van de graaf. Het is pas na de merge van knopen 3 en 5 dat we kunnen zien waar de twee algoritmes verschillen. De knoop "3, 5" is namelijk niet adjacent met knoop 1 en wordt dus hiermee gemergd (zie figuur 5.9). In het Dutton & Brigham-algoritme moesten we eerst zoeken naar een onafhankelijk paar knopen. In dit voorbeeld lijkt dit verschil miniem, maar in grote grafen gaat dit kleine verschil al snel grotere gevolgen hebben, zoals zal blijken in hoofdstuk 8. Tenslotte zien we in figuuren 5.10 de complete graaf nadat de laatste merge gebeurde en in figuur 5.11 een kleuring met drie kleuren van de originele graaf. We hebben voor dit voorbeeld dus een kleuring kunnen construeren die hetzelfde aantal kleuren gebruikt als het Dutton & Brigham-algoritme. Maar we hebben gezien dat het Cosine-algoritme zijn kleuring met minder bewerkingen kan bekomen.



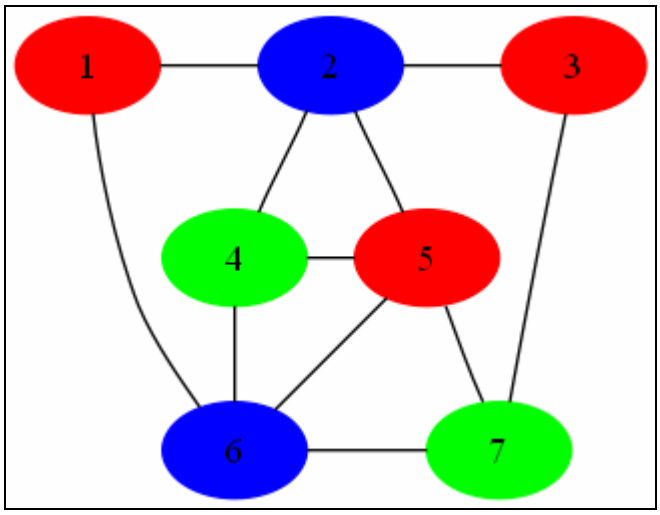

**Figuur 5.8: Eerste stappen van het Cosine-algoritme op invoer van graaf G.**




**Figuur 5.9: Nieuwe graaf G = G / (3,5) , 1**




**Figuur 5.10: Nieuwe graaf G = G / 4, 7**



Figuur 5.11: Ingekleurde graaf G

### 5.3.3 Pseudocode

De implementatie van het Cosine-algoritme kunnen we op informele wijze beschijven met de pseudocode uit tabel 5.6.

| COSINE-algoritme |
|---|
| *Invoer*: graaf G  *Uitvoer*: een kleuring van G<br><br>Initialiseer:<br>  tabel T met voor ieder paar onafhankelijke knopen het aantal gemeenschappelijke buren<br><br>*while* G ≠ compleet<br>  zoek het paar onafhankelijke knopen $v_r$ , $v_s$ met het grootste aantal gemeenschappelijke buren<br>  G = G / $v_r$ , $v_s$<br>  update tabel T<br>  *while* G heeft knopen onafhankelijk aan $v_r$<br>  zoek knoop $v_s$, onafhankelijk aan $v_r$, met grootste aantal gemeenschappelijke buren met $v_r$<br>  G = G / $v_r$ , $v_s$<br>  update tabel T<br><br>Maak kleuring K door gemergde knopen dezelfde kleur te geven<br><br>*return* kleuring K |

**Tabel 5.6: Pseudocode Cosine-algoritme**

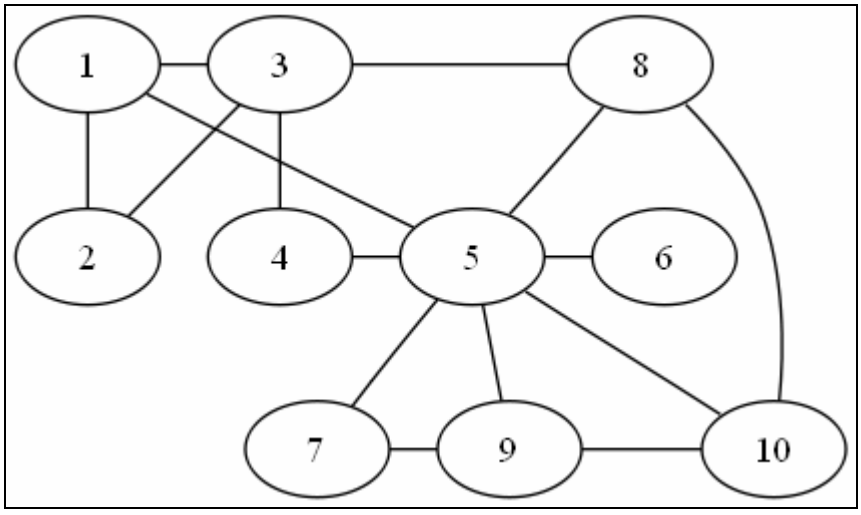
## 5.4 Recursive-Largest-First (RLF) algoritme

### 5.4.1 Algoritme

Een efficiënter algoritme, dat volgens Walter Klotz[2] ook betere resultaten behaalt dan de vorige algoritmes, is het Recursive-Largest-First-algoritme van Leighton[14], ook kortweg RLF genoemd. Nog steeds gaan we op zoek naar een paar knopen om te mergen, met als doel een zo goed mogelijke kleuring te construeren. Net zoals in het Cosine-algoritme gaan we eerst één knoop *x* vast kiezen. We veranderen de knoop *x* niet totdat knoop *x* adjacent is met alle knopen. Om het knopenpaar te vervolledigen kiezen we een knoop *y* die onafhankelijk is aan *x*. Het criterium voor de keuze van knoop *x* is de knoop met de maximum graad in de graaf. Dit criterium lijkt ook een goede strategie omdat knopen met een hoge graad een grote beperkende rol spelen in het toekennen van kleuren. De knoop *y* die we dan gaan mergen met *x* is die knoop die het grootste aantal gemeenschappelijke buren heeft met *x*. Wanneer knoop *x* adjacent is met alle andere knopen, verwijderen we de knoop uit de graaf en herhalen we het proces op de overgebleven graaf. De knoop *x* vormt samen met de knopen die we ermee gemergd hebben, één kleurklasse. In hoofdstuk 8 zullen we in de resultaten tonen hoe RLF presteert t.o.v. de vorige twee algoritmes.

### 5.4.2 Voorbeeld

We nemen als invoer voor het RLF-algoritme de graaf uit figuur 5.12. We zoeken eerst de knoop *x* met de hoogste graad in graaf *G* (zie tabel 5.7). Dit is knoop 5. Daarna zoeken we de onafhankelijke knoop *y* met het hoogste aantal gemeenschappelijke buren met knoop 5 (zie tabel 5.8). Er zijn 2 onafhankelijke knopen, nl. knopen 2 en 3. Knoop 3 heeft het hoogste aantal gemeenschappelijke buren met knoop 5. Daarom gaan we als eerste deze twee knopen, 5 en 3, mergen. Dit levert ons de graaf uit figuur 5.13 op. We zien dat knoop "5, 3" nu met alle andere knopen adjacent is. En we verwijderen deze knoop uit de graaf en bekomen de graaf uit figuur 5.14. In figuur 5.15 zien we wat de verdere stappen in het RLF-algoritme zijn. Knoop 9 is de knoop met de hoogste graad en we mergen hem achtereenvolgens met knopen 8, 4, 6 en 2. Na het verwijderen van deze knoop, omdat hij adjacent is met alle andere knopen van de graaf, behouden we 3 losse knopen die we ook mergen. Dit levert een kleuring met 3 kleuren zoals we er een zien in figuur 5.16.



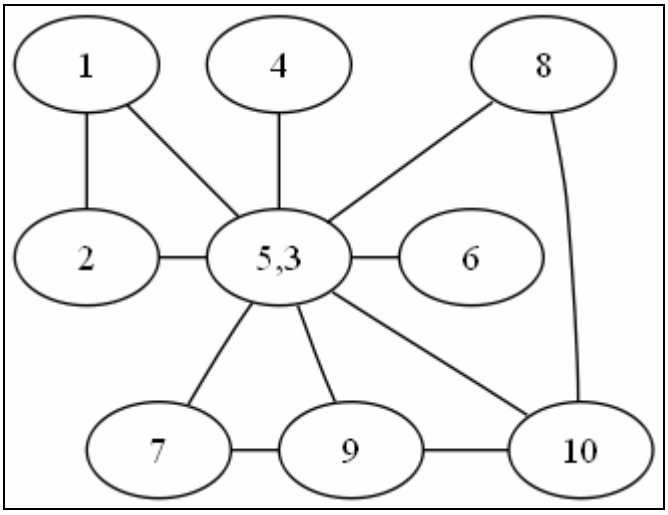
**Figuur 5.12: Graaf G**

| Knoop | Graad in graaf *G* |
|---|---|
| 1 | 3 |
| 2 | 2 |
| 3 | 4 |
| 4 | 2 |
| 5 | 7 |
| 6 | 1 |
| 7 | 2 |
| 8 | 3 |
| 9 | 3 |
| 10 | 3 |

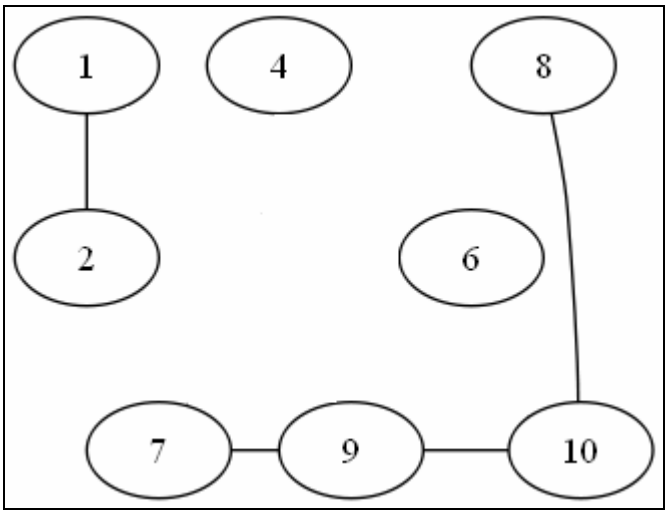
**Tabel 5.7 : Graad van de knopen in graaf G uit figuur 5.12**

| Knoop (onafhankelijk aan knoop 5) | Aantal gemeenschappelijke buren met knoop 5 |
|---|---|
| 2 | 1 |
| 3 | 3 |

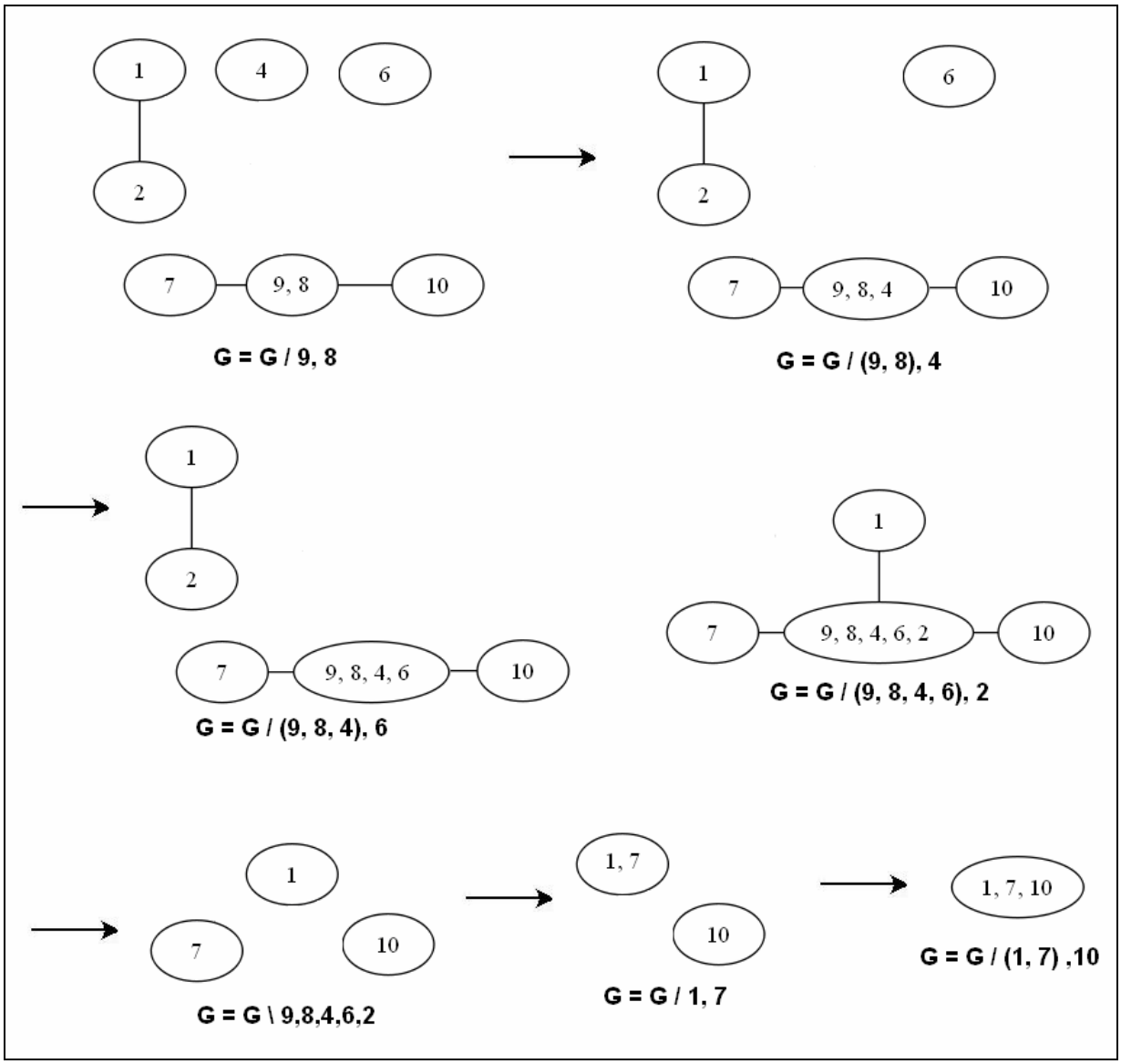
**Tabel 5.7 : Tabel met aantal gemeenschappelijke buren tussen knoop 5 en zijn onafhankelijke knopen uit graaf G uit figuur 5.12**



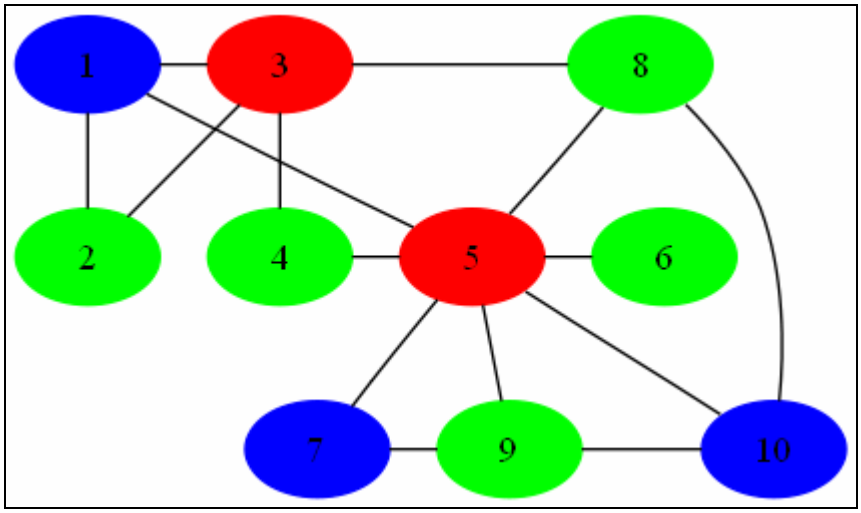

**Figuur 5.13: Nieuwe graaf G = G / 5, 3**




**Figuur 5.14: Nieuwe Graaf G na verwijderen knoop 5, 3**




**Figuur 5.15: Verdere stappen van het RLF-algoritme**




**Figuur 5.16: Kleuring van graaf G uit figuur 5.12**

### 5.4.3 Pseudocode

De implementatie van het Recursive-Largest-First (RLF)-algoritme kunnen we op informele wijze beschijven met de pseudocode uit tabel 5.8.

| RLF-algoritme |
|---|
| *Invoer*: graaf G=(V, E)  *Uitvoer*: een kleuring van G<br><br>*Initialisatie*: k=1; kleuring K = Ø;<br><br>*while* \|V\| > 0<br>  zoek knoop x met maximum graad<br>  *while* x heeft onafhankelijke knopen<br>  zoek knoop y, onafhankelijk aan x, met hoogste aantal buren gemeen met x<br>  G = G / x, y<br>  K(x)=k; geef gemergde knopen ook kleur k<br>  k++<br>  G = G – x<br><br>return kleuring K |

**Tabel 5.8: Pseudocode Recursive-Largest-First (RLF)-algoritme**

### 5.4.4 Tijdscomplexiteit

Ook RLF heeft met een complexiteit van $O(n^3)$; dezelfde theoretische grens als de vorige twee algoritmes. Maar een bestudering van het algoritme laat zien dat het RLF-algoritme in praktijk sneller zal werken dan deze twee. We vergelijken RLF met de in praktijk snelste van de twee, Cosine. Het verschil schuilt in de eerste stap van de lusdoorgang, waar we in het

RLF-algoritme op zoek gaan naar de knoop met maximum graad. Dit kan in lineaire tijd O(n) gebeuren en dit is natuurlijk een factor *n* sneller, mits keurig bijhouden van beide waarden, dan het zoeken naar het paar onafhankelijke knopen met de meeste gemeenschappelijke buren.

# Hoofdstuk 6

# Sequentiële algoritmes

Eén van de simpelste en intuïtiefste methoden om een graaf te kleuren, valt onder de noemer van sequentiële algoritmes. We gaan één voor één de knopen af, en geven de knoop de 'kleinste' kleur die niet gebruikt is voor de buren van die knoop. We stellen dus als het ware een volgorde $O = [v_1, v_2, \ldots, v_n]$ van de knopen op, in dewelke we ze gaan afhandelen. Vervolgens kleuren we $v_1$ met kleur "1". Als knoop $v_2$ een buur is van $v_1$, zijn we verplicht $v_2$ een andere kleur te geven, namelijk kleur "2". Als knoop $v_2$ echter geen buur is van $v_1$, geven we $v_2$ ook kleur "1". We blijven hetzelfde stramien toepassen voor de volgende knopen uit onze volgorde, totdat we de laatste knoop hebben gekleurd. Het is duidelijk dat deze methode geen garantie geeft om de beste oplossing te vinden, maar het is een snelle en eenvoudige manier om een kleuring te bekomen. De opzet van sequentiële algoritmes is natuurlijk om een volgorde te bepalen die de optimale kleuring bekomt of zo dicht mogelijk benaderd.

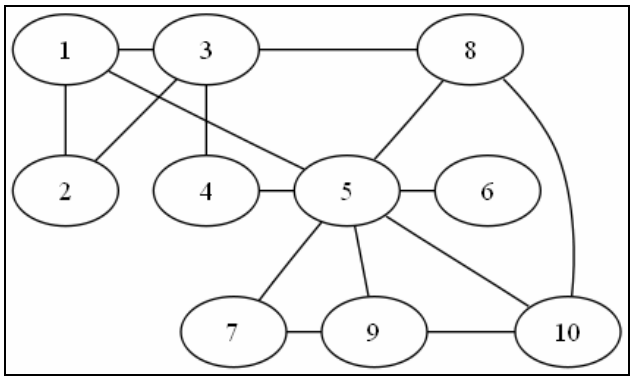
## 6.1 Welsh & Powell algoritme

### 6.1.1 Algoritme en tijdscomplexiteit

In het Welsh & Powell-algoritme bepalen we de volgorde van de knopen volgens de graad van de knoop. De volgorde is van hoogste naar laagste graad. Het algoritme wordt daarom ook wel Largest First(LF) genoemd. Het algoritme heeft een complexiteit $O(n^2)$ omdat we voor iedere knoop moeten testen met welke reeds gekleurde knopen de knoop adjacent is. De kwadratische tijdscomplexiteit klinkt aanlokkelijk, echter de kwaliteit van het resultaat, een kleuring die dit algoritme bekomt, boet in voor de snelheid. Dit algoritme wordt daarom best alleen gebruikt voor zeer grote grafen.

### 6.1.2 Voorbeeld

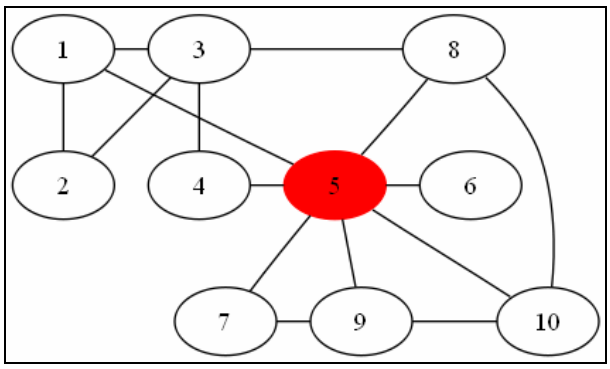
We nemen de graaf uit figuur 6.1 als voorbeeld. In tabel 6.1 zien we de volgorde waarin we kleuren aan de knopen zullen toekennen. We zullen voor dit voorbeeld ook een volgorde in de kleuren leggen. Van klein naar groot wordt dat rood, groen, blauw (we hebben voor dit voorbeeld drie kleuren nodig). De eerste knoop die we kleuren is knoop 5 en deze krijgt de kleinste kleur, rood (zie figuur 6.2). Knoop 3 krijgt ook de kleur rood (zie figuur 6.3). Maar knoop 1 moet een andere kleur krijgen, want deze is adjacent met knoop 3. Daarom geven we de kleinst beschikbare kleur, groen, aan knoop 1 (zie figuur 6.4). De andere kleurtoekenningen zijn te zien in figuren 6.5 t/m 6.11.



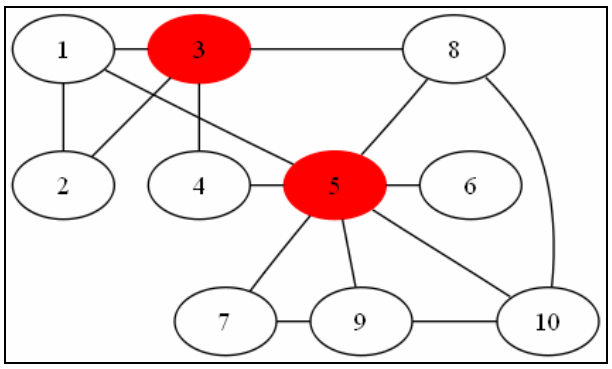

**Figuur 6.1: Graaf G**

| Volgorde | 5 | 3 | 1 | 8 | 9 | 10 | 2 | 4 | 7 | 6 |
|---|---|---|---|---|---|---|---|---|---|---|

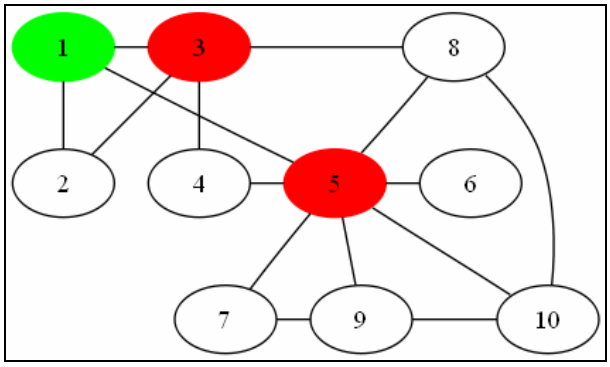
**Tabel 6.1: Volgorde volgens graad van de knopen van graaf G uit figuur 6.1**



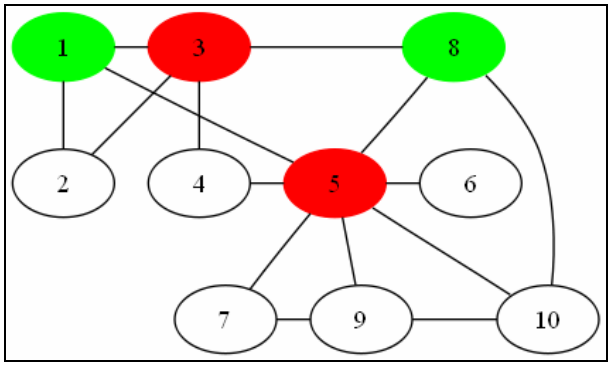

**Figuur 6.2: Kleurtoekenning aan knoop 5**



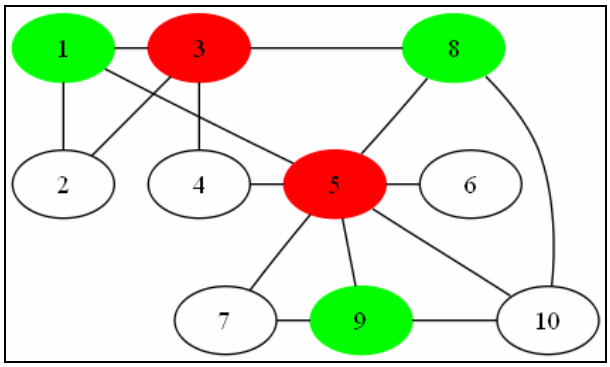

**Figuur 6.3: Kleurtoekenning aan knoop 3**



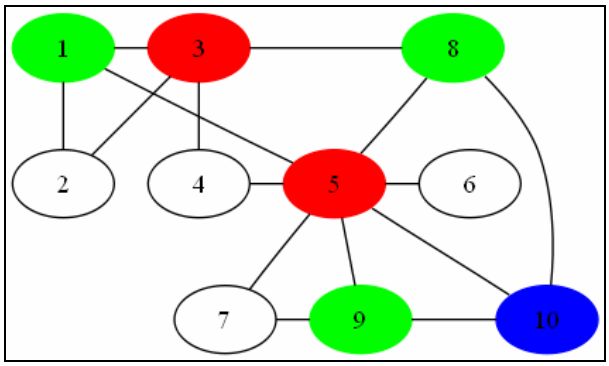

**Figuur 6.4: Kleurtoekenning aan knoop 1**



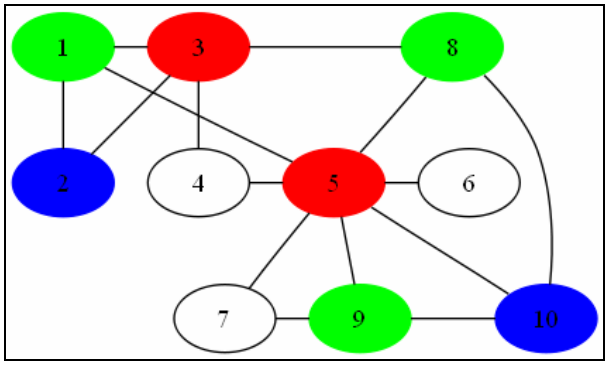

**Figuur 6.5: Kleurtoekenning aan knoop 8**



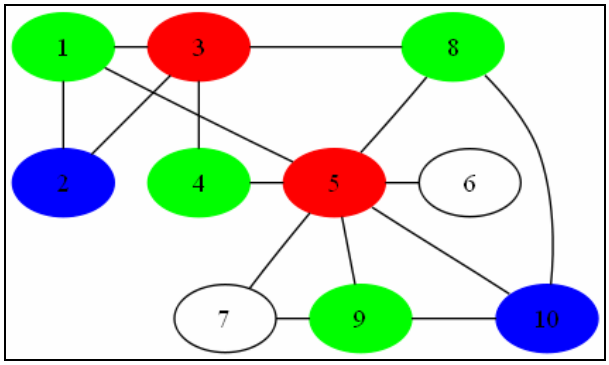

**Figuur 6.6: Kleurtoekenning aan knoop 9**



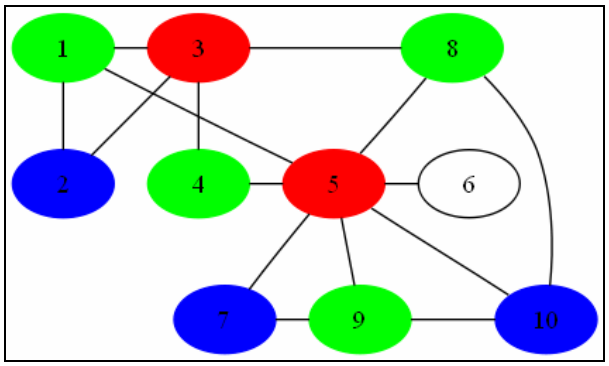

**Figuur 6.7: Kleurtoekenning aan knoop 10**



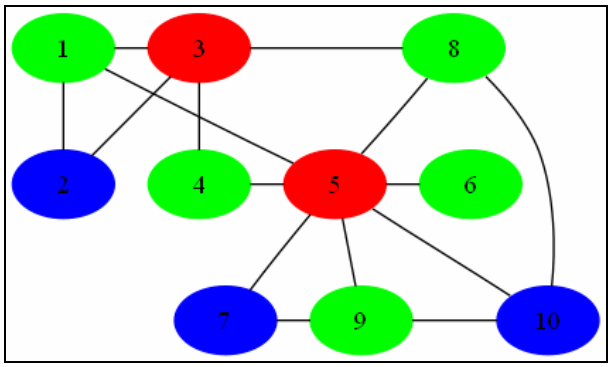

**Figuur 6.8: Kleurtoekenning aan knoop 2**




**Figuur 6.9: Kleurtoekenning aan knoop 4**




**Figuur 6.10: Kleurtoekenning aan knoop 7**




**Figuur 6.11: Kleurtoekenning aan knoop 6**

### 6.1.3 Pseudocode

De implementatie van het Welsh & Powell-algoritme kunnen we op informele wijze beschijven met de pseudocode uit tabel 6.2.

| Welsh & Powell-algoritme |
| --- |
| *Invoer*: graaf G  *Uitvoer*: een kleuring K van G<br><br>*Initialisatie*: kleuring K = $\emptyset$;<br><br>Rangschik de knopen in niet-stijgende orde volgens graad: O = [$v_1$, $v_2$ , …,$v_n$]<br><br>K($v_1$) = 1<br>for i = 2 … n<br>  K($v_i$) = kleinste kleur die niet gebruikt is bij buren van $v_i$<br><br>return kleuring K |

**Tabel 6.2: Pseudocode Welsh & Powell algoritme**

## 6.2 DSatur

### 6.2.1 Algoritme

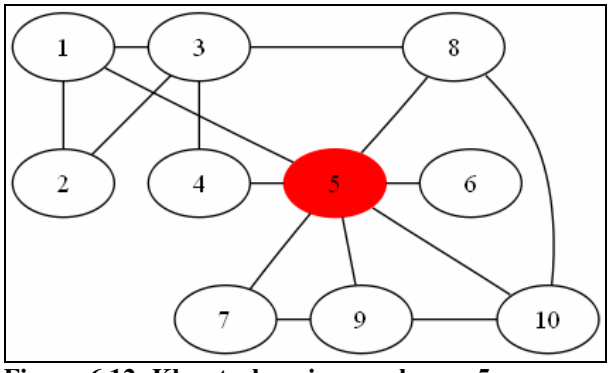
Wanneer we op voorhand de volgorde moeten bepalen waarin we de knopen zullen kleuren, negeren we informatie die we bekomen tijdens het algoritme. Daarom introduceerde Brélaz[15] een algoritme, DSatur, waarin we de volgorde waarin we knopen kleuren gaandeweg verbeteren met de beschikbare informatie. Deze informatie komt in de vorm van het aantal verschillende kleuren dat er gebruikt is bij de buren van een knoop. We noemen dit aantal de *verzadigingsgraad* van knoop *x*, en noteren het met $deg_s(x)$.
De volgorde wordt bepaald volgens de verzadigingsgraad van de knopen, van hoogste naar laagste graad. Bij een ex-aequo van de verzadigingsgraad, rangschikken we volgens de graad van de knoop, weer van hoog naar laag. De idee achter deze volgorde is: hoe meer verzadiging er in een knoop is, hoe minder mogelijkheden overblijven om de knoop te kleuren. Na iedere toekenning van een kleur aan een knoop *x*, herberekenen we de verzadigingsgraad van de buren van *x*. Met deze nieuwe informatie passen we de volgorde van de nog ongekleurde knopen, indien nodig, aan en herhalen we het proces.

### 6.2.2 Voorbeeld

We passen het DSatur-algoritme toe op de graaf uit figuur 6.1. Dit levert de startvolgorde uit tabel 6.3 op. We gebruiken weer de volgorde van kleuren: rood, groen, blauw.
Vervolgens zien we aan de linkerkant in de figuren 6.12 t/m 6.21 steeds de nieuwe grafen waar steeds één knoop meer gekleurd is. En aan de rechterkant zien we in de tabellen 6.4 t/m 6.12 de nieuwe volgordes voor die graaf. Uiteindelijk bekomen we een kleuring met 3 kleuren.

| Volgorde | 5 | 3 | 1 | 8 | 9 | 10 | 2 | 4 | 7 | 6 |
|---|---|---|---|---|---|---|---|---|---|---|
| $deg_s(x)$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $d(x)$ | 7 | 4 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 1 |

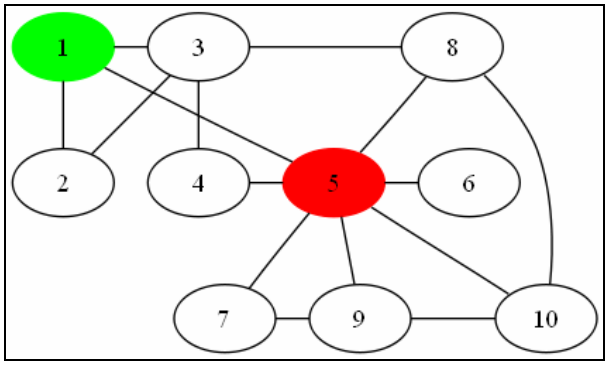
**Tabel 6.3: Volgorde volgens graad van de knopen van graaf G uit figuur 6.1**




**Figuur 6.12: Kleurtoekenning aan knoop 5**

| Volgorde | 1 | 8 | 9 | 10 | 4 | 7 | 6 | 3 | 2 |
|---|---|---|---|---|---|---|---|---|---|
| $deg_s(x)$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| $d(x)$ | 3 | 3 | 3 | 3 | 2 | 2 | 1 | 3 | 2 |

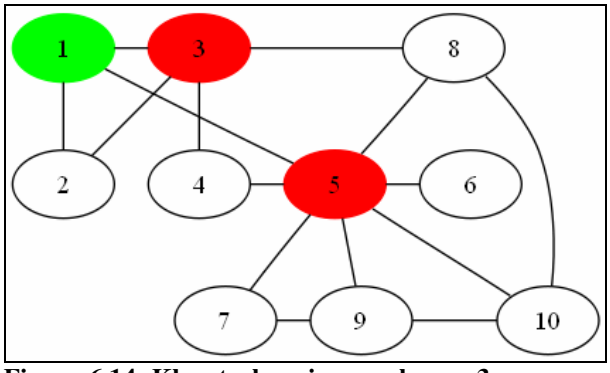
**Tabel 6.4: Nieuwe volgorde van de knopen van graaf G uit figuur 6.12**




**Figuur 6.13: Kleurtoekenning aan knoop 1**

| Volgorde | 3 | 8 | 9 | 10 | 4 | 7 | 6 | 2 |
|---|---|---|---|---|---|---|---|---|
| $deg_s(x)$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $d(x)$ | 3 | 3 | 3 | 3 | 2 | 2 | 1 | 2 |

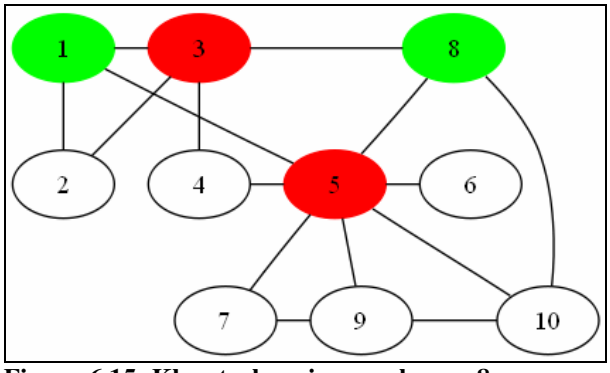
**Tabel 6.5: Nieuwe volgorde van de knopen van graaf G uit figuur 6.13**




**Figuur 6.14: Kleurtoekenning aan knoop 3**

| Volgorde | 8 | 2 | 4 | 9 | 10 | 7 | 6 |
|---|---|---|---|---|---|---|---|
| $deg_s(x)$ | 2 | 2 | 2 | 1 | 1 | 1 | 1 |
| $d(x)$ | 3 | 2 | 2 | 3 | 3 | 2 | 1 |

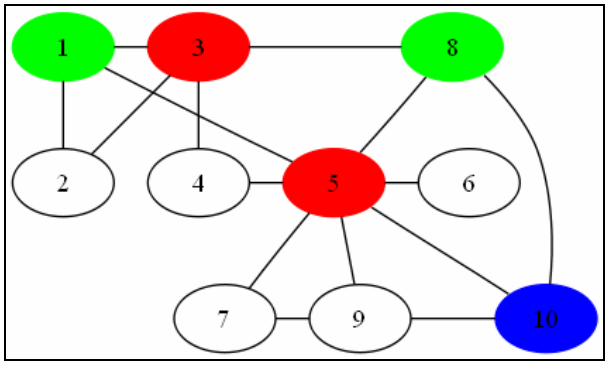
**Tabel 6.6: Nieuwe volgorde van de knopen van graaf G uit figuur 6.14**




**Figuur 6.15: Kleurtoekenning aan knoop 8**

| Volgorde | 10 | 2 | 4 | 9 | 7 | 6 |
|---|---|---|---|---|---|---|
| $deg_s(x)$ | 2 | 2 | 2 | 1 | 1 | 1 |
| $d(x)$ | 3 | 2 | 2 | 3 | 2 | 1 |

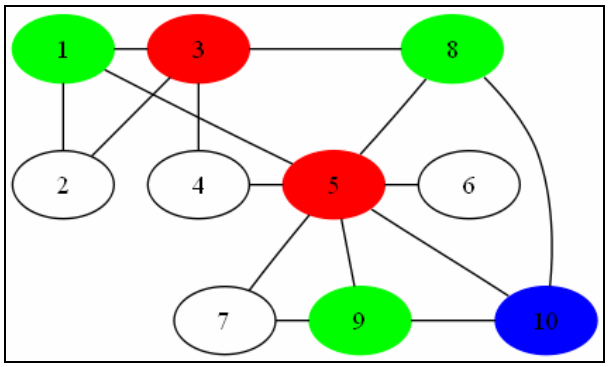
**Tabel 6.7: Nieuwe volgorde van de knopen van graaf G uit figuur 6.15**




**Figuur 6.16: Kleurtoekenning aan knoop 10**

| Volgorde | 9 | 2 | 4 | 7 | 6 |
|---|---|---|---|---|---|
| $deg_s(x)$ | 2 | 2 | 2 | 1 | 1 |
| $d(x)$ | 3 | 2 | 2 | 2 | 1 |

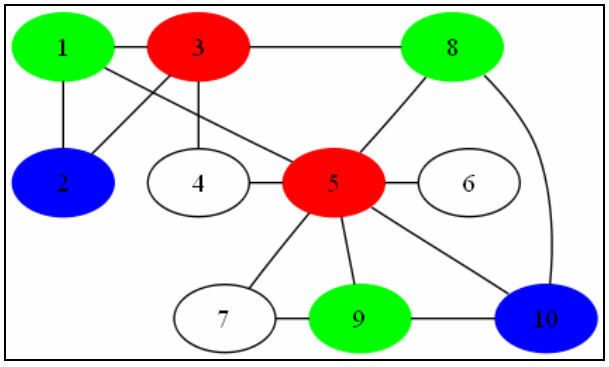
**Tabel 6.8: Nieuwe volgorde van de knopen van graaf G uit figuur 6.16**




**Figuur 6.17: Kleurtoekenning aan knoop 9**

| Volgorde | 2 | 4 | 7 | 6 |
|---|---|---|---|---|
| $deg_s(x)$ | 2 | 2 | 2 | 1 |
| $d(x)$ | 2 | 2 | 2 | 1 |

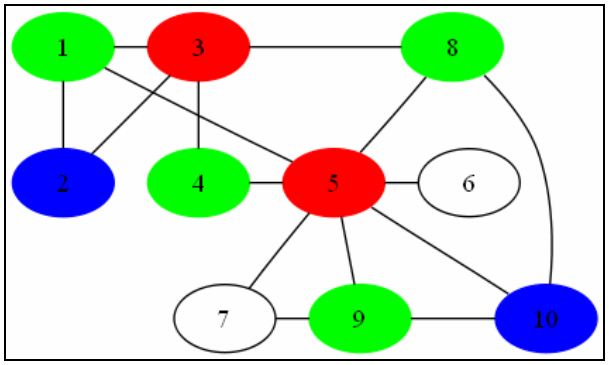
**Tabel 6.9: Nieuwe volgorde van de knopen van graaf G uit figuur 6.17**




**Figuur 6.18: Kleurtoekenning aan knoop 2**

| Volgorde | 4 | 7 | 6 |
|---|---|---|---|
| $deg_s(x)$ | 2 | 2 | 1 |
| $d(x)$ | 2 | 2 | 1 |

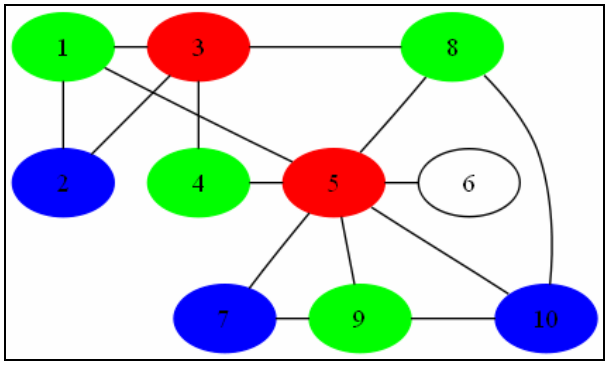
**Tabel 6.10: Nieuwe volgorde van de knopen van graaf G uit figuur 6.18**




**Figuur 6.19: Kleurtoekenning aan knoop 4**

| Volgorde | 7 | 6 |
|---|---|---|
| $deg_s(x)$ | 2 | 1 |
| $d(x)$ | 2 | 1 |

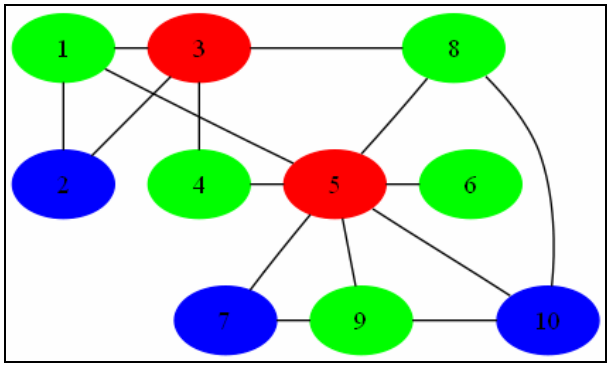
**Tabel 6.11: Nieuwe volgorde van de knopen van graaf G uit figuur 6.19**




**Figuur 6.20: Kleurtoekenning aan knoop 7**

| Volgorde | 6 |
|---|---|
| $deg_s(x)$ | 1 |
| $d(x)$ | 1 |

**Tabel 6.12: Nieuwe volgorde van de knopen van graaf G uit figuur 6.20**




**Figuur 6.21: Kleurtoekenning aan knoop 6**

### 6.2.3 Pseudocode

De implementatie van het DSatur-algoritme kunnen we op informele wijze beschijven met de pseudocode uit tabel 6.13.

| DSatur-algoritme |
| --- |
| *Invoer*: graaf G  *Uitvoer*: een kleuring van G<br><br>*Initialisatie*: kleuring K = $\emptyset$;<br><br>Rangschik de knopen volgens niet-stijgende verzadigingsgraad $deg_s(v_1) \geq deg_s(v_2) \geq \cdots \geq deg_s(v_n)$, bij ex-aequo volgens graad, hoog naar laag van de knoop:<br>O = [$v_1$, $v_2$, …,$v_n$]<br><br>K($v_1$) = 1<br>for i = 2 … n<br>  update de verzadigingsgraad van de buren van $v_{i-1}$<br>  herschik de volgorde (indien nodig)<br>  K($v_i$) = kleinste kleur niet gebruikt bij buren van $v_i$<br><br>return kleuring K |

**Tabel 6.13: Pseudocode DSATUR-algoritme**

### 6.2.4 Tijdscomplexiteit

Het DSatur-algoritme heeft met een zorgvuldige implementatie een complexiteit van $O(n^2)$. Door het bijhouden van een tabel, die we construeren in $O(n^2)$, met de verzadigingsgraden van de knopen, kunnen we ook de hoofdlus, omdat we enkel updates moeten doen aan de buren van de laatst gewijzigde knoop, beperken tot $O(n^2)$.

# Hoofdstuk 7

# Implementatie

## 7.1 Algemeen

Bij een masterproef informatica hoort natuurlijk ook een onderzoeksproduct. Voor deze masterproef is dat zowel de implementatie van enkele heuristische algoritmes voor het oplossen van het graafkleurenprobleem, alsook enkele experimenten met deze algoritmes. De experimenten vormen het onderwerp van het volgende hoofdstuk. In dit hoofdstuk komen de implementaties zelf aan bod.

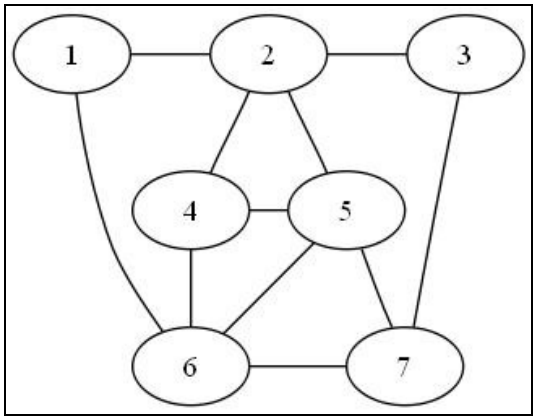
De allereerste beslissing die gemaakt werd, was die van de programmeertaal. Hier ging de keuze naar het programmeren in Java en dit omwille van mijn persoonlijke voorkeur voor Java over andere programmeertalen. Een masterproef vergt bloed, zweet en tranen en dus kan er best gewerkt worden in de meest gebruiksvriendelijke omstandigheden.

Wanneer er een implementatie wordt gemaakt van een (klassiek) probleem uit de grafentheorie, ligt het voor de hand om een goede uitgangspositie te creëren.
Het opstellen van een goede interface en representatie van de graafstructuur is dan ook het eerste aandachtspunt.

Bij een graaf-implementatie hoort natuurlijk een keuze of er gewerkt wordt met een matrix- of een lijstvoorstelling, of zelfs allebei. De voorkeur ging naar een matrixvoorstelling, omdat op die manier de informatie over de bogen van de graaf onmiddellijk beschikbaar is. Een nadeel van een matrixvoorstelling kan zijn dat, als er slechts weinig bogen zijn, het grootste deel van de matrix onbenut blijft. Dit is bij het graafkleurenprobleem, en vooral dan bij deze masterproef, minder het geval omdat we die algoritmes onderzoeken die op grote grafen toch een snel en goed resultaat boeken.

Het feit dat we ons voor de input van het graafkleurenproblemen beperken tot simpele, ongerichte, samenhangende grafen betekent dat we nog verdere aanpassingen aan de matrixvoorstelling kunnen maken. Het gevolg van de beperking tot ongericht grafen is dat één helft van de matrix redundant is. En de beperking tot simpele grafen maakt zelfs de informatie op de diagonaal van de matrix overtollig.

In tabel 7.1 is te zien hoe een matrixvoorstelling van een simpele en ongerichte graaf uit figuur 7.1 eruit ziet. We zien dat er enkel een waarde is als $i \geq j$. We kunnen deze informatie ook bewaren in één enkele array (zie tabel 7.2). De grootte van deze array is dan gelijk aan $n*(n-1)/2$ voor een graaf met $n$ knopen. We kunnen nog steeds in constante tijd de booginformatie opzoeken, de informatie van boog $\{i, j\}$ staat op index $i*(i-1)/2+j$.




**Figuur 7.1:Graaf G**

| i \ j | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | | | | | | | |
| 2 | x | | | | | | |
| 3 | - | x | | | | | |
| 4 | - | x | - | | | | |
| 5 | - | x | - | x | | | |
| 6 | x | - | - | x | x | | |
| 7 | - | - | x | - | x | x | |

**Tabel 7.1: Lower Triangular Matrix van graaf G uit figuur??**

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| x | - | x | - | x | - | - | x | - | x | x | - | - | x | x | - | - | x | - | x | x |

**Tabel 7.2: Enkelvoudige array-voorstelling van graaf G uit figuur 7.1**

Ook werd er aandacht besteed aan de herbruikbaarheid en onderhoudbaarheid van de code door de creaties van interfaces en/of abstracte klassen, en door het gebruik van ontwerppatronen.
Zo vormen de klasses Graph, GraphColoring en GraphColoringAlgorithm de ruggengraat van het project waarop verder gebouwd kan worden. Dit maakt het voor iedereen mogelijk om ook in de toekomst op een doorzichtige manier aanpassingen en toevoegingen te maken. Zo kan, indien gewenst, ook een lijstvoorstelling van een graaf geïmplementeerd worden zonder aanpassing van de huidige algoritmes. Of kunnen er andere algoritmes toegevoegd worden, die gebruik maken van de bestaande structuren.

De volgende algoritmes werden geïmplementeerd:

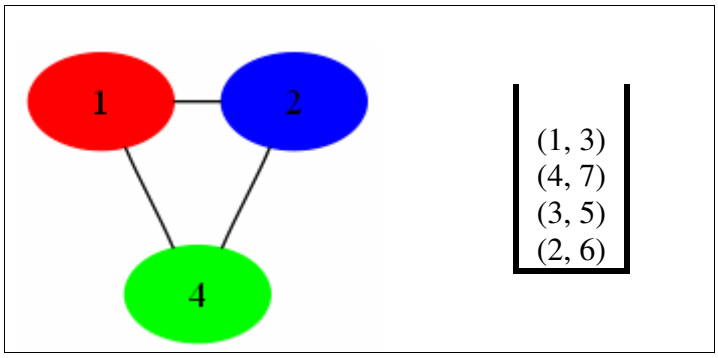
- Dutton & Brigham
- Cosine
- RLF (Recursive Largest First)
- DSatur

## 7.2 Dutton & Brigham en Cosine

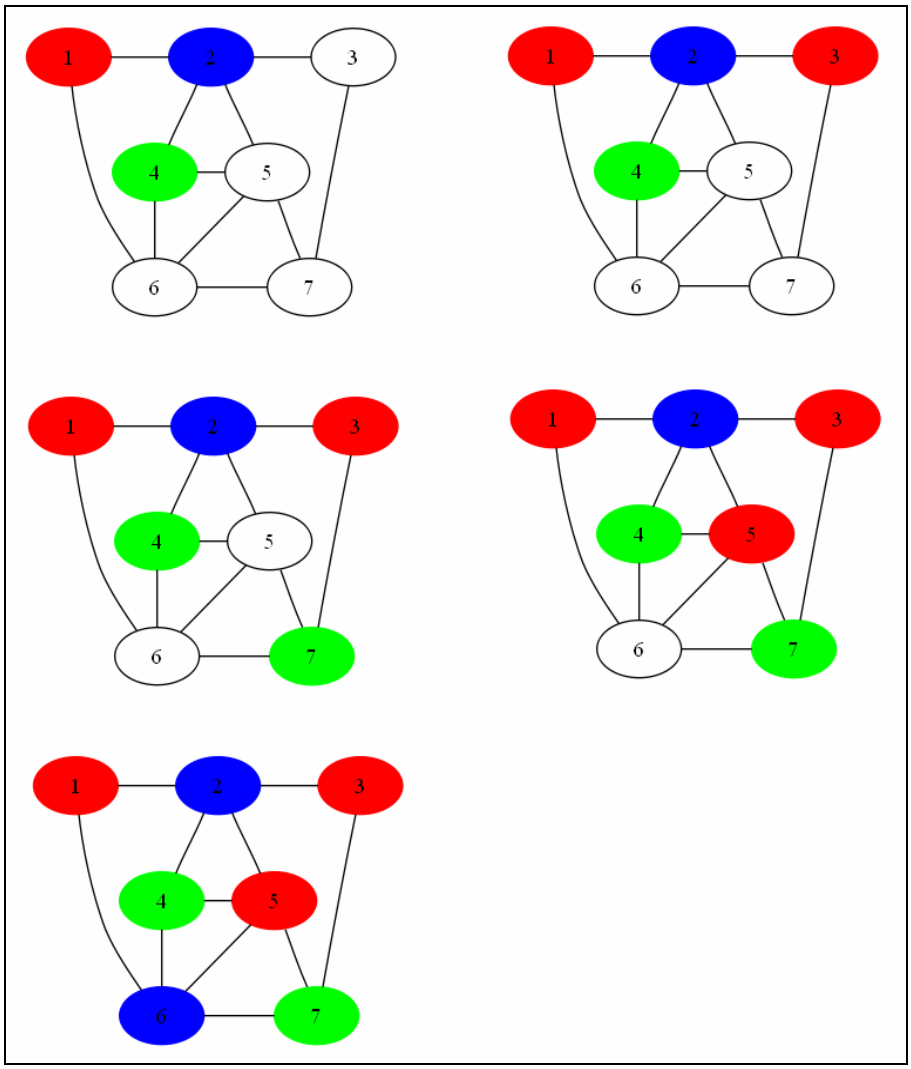
Voor de eerste twee algoritmes is het het vermelden waard hoe het kleuren van de graaf gebeurt. We beginnen namelijk slechts de knopen te kleuren nadat het herhaaldelijk mergen van knopen een complete graaf heeft opgeleverd. Een complete graaf is makkelijk in te kleuren door iedere knoop een andere kleur toe te kennen.
Daarna moeten echter nog de andere knopen van de originele graaf gekleurd worden.
Twee knopen die gemergd werden krijgen dezelfde kleur. Het is dus noodzakelijk om vanuit de complete graaf stap voor stap, merge per merge, terug de originele graaf op te bouwen.

We moeten dus beginnnen met de laatste merge die we deden. Deze acties doen onmiddellijk denken aan een stapel en dat is exact wat gebruikt wordt in de implementatie.
Iedere merge, meerbepaald een paar integers (x, y), *pushen* we op de stack totdat we een complete graaf hebben bekomen. We kleuren vervolgens de bekomen complete graaf door iedere knoop een andere kleur te geven. Daarna kunnen we door de paren van de stack te *poppen* de originele graaf kleuren.

Voor de graaf in figuur 7.1 bekomen we de complete graaf en stack in figuur 7.2. In de originele graaf geven we eerst knopen 1, 2 en 4 hun kleur. Vervolgens geven we knoop 3 dezelfde kleur als knoop 1, knoop 7 dezelfde als knoop 4, knoop 5 dezelfde als knoop 3, en tenslotte knoop 6 dezelfde als knoop 2. Dit proces is ook weergegeven in figuur 7.3.



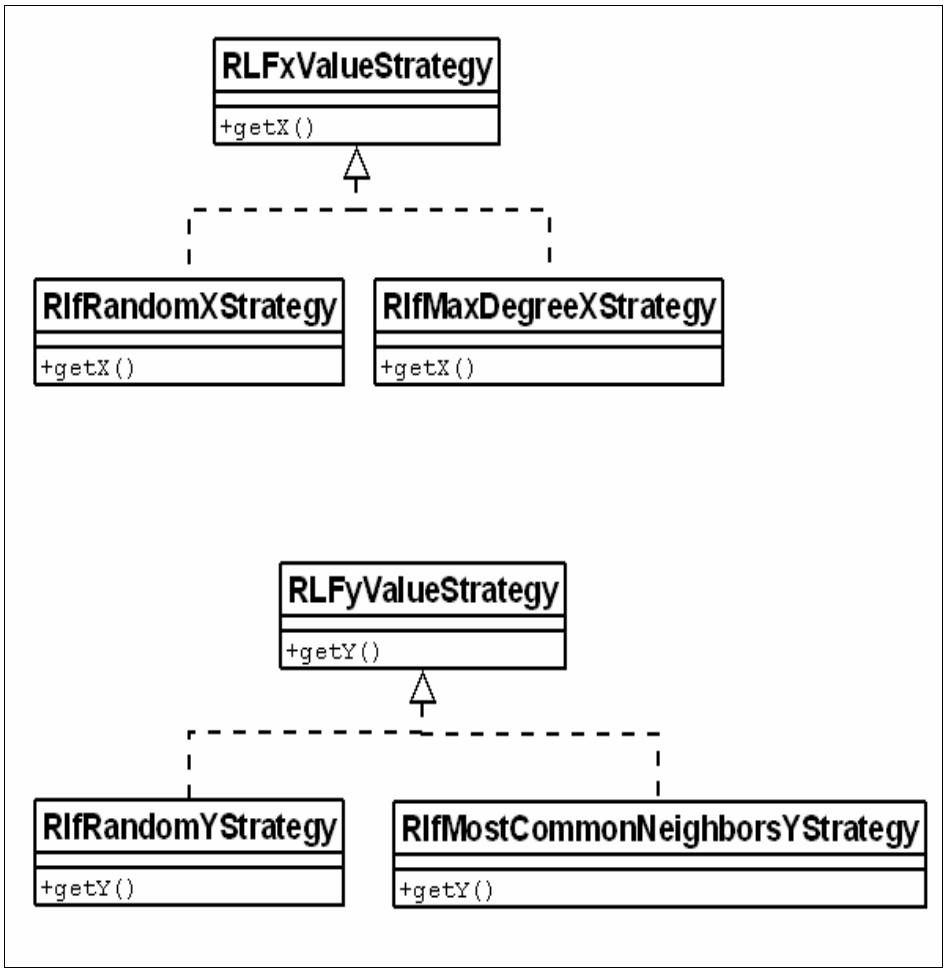

**Figuur 7.2: Complete graaf en de bekomen stack**




**Figuur 7.3: Toekenning van de kleuren**

## 7.3 RLF

Herinner dat we ook bij het RLF-algoritme steeds opnieuw op zoek zijn naar de twee knopen *x* en *y*, die volgens het RLF-criterium het beste zijn om te mergen. Het algoritme schrijft voor knoop *x* die knoop *x* te kiezen met de hoogste graad in de (overblijvende) graaf. En voor knoop *y* kiezen we dan die knoop die het meeste buren gemeen heeft met de eerder gekozen knoop *x*.
Om te controleren en experimenteren hoeveel invloed deze criteria hebben op de uiteindelijke kleuring, implementeerden we voor zowel de x-keuze als de y-keuze ook een strategie die een willekeurige knoop selecteert. Om deze opties dynamisch te selecteren bij het uitvoeren, werd er gebruik gemaakt van het strategy-pattern[16 & 17]. Dit ontwerppatroon heeft tot doel om later met een minimale inspanning nog andere criteria te programmeren en in te voegen, zonder dat het basisalgoritme herschreven moet worden.




**Figuur 7.4: Strategy pattern**

## 7.4 DSatur

Het DSatur-algoritme vereist het bijhouden van een dynamische rangschikking van de knopen volgens de verzadigingsgraad. Deze rangschikking moet iedere iteratie worden bijgewerkt. De keuze voor de Java TreeSet[17] lag voor dit doeleinde voor de hand. De Java TreeSet implementatie verstrekt een gewaarborgde tijdscomplexiteit van O(log n) voor de basisoperaties *add*, *remove* en *contains*. Het is niet zo dat iedere iteratie de volledige rangschikking opnieuw wordt gemaakt. Het is mogelijk om enkel de waarden van die knopen, die door de vorige iteratie beïnvloed werden, opnieuw te berekenen en in de rangschikking aan te passen.

## 7.5 Uitvoeren

Bij het uitvoeren van het programma kunnen er enkele parameters meegegeven worden. Zo kunnen we een keuze maken welk algoritme we willen gebruiken, hoeveel keer we het algoritme willen uitvoeren, en naar welk bestand we de oplossing, een kleuring voor de invoergraaf, willen wegschrijven. Het is ook mogelijk om meerdere problemen, die we willen oplossen, in één keer in te voeren.

# Hoofdstuk 8

# Experimenten

Het voornaamste doel van het implementeren van de algoritmes is uiteraard om deze experimenteel te kunnen vergelijken. Hiervoor hebben we een verzameling grafen nodig die we als invoer voor de algoritmes kunnen gebruiken. De standaard hiervoor werd reeds gezet tijdens de "The Second DIMACS Challenge: NP Hard Problems: Maximum Clique, Graph Coloring, and Satisfiability"[18] die plaatsvond tussen september 1992 en september 1993. Het doel van een DIMACS challenge is het onderzoek naar de experimentele analyse van algoritmes te bevorderen en te coördineren. Vooral problemen waarvoor de worst case analyse zeer pessimistisch is komen aan bod. In de "The Second DIMACS Challenge" werden drie moeilijke computationele problemen behandeld. Eén van die problemen was dus het graafkleurenprobleem.
Voor deze uitdaging ontwierpen ze niet aleen een standaard formaat[19] voor de voorstelling van grafen, maar stelden ook een aantal instanties in dit bestandsformaat ter beschikking. Deze bestanden bevatten grafen, in dit standaard formaat, waarvoor we de beste graafkleuring moeten vinden. In tabel(??) zien we de bestanden van de Second Challenge, met voor iedere graaf het aantal knopen en bogen in de graaf, en op enkele grafen na, ook het aantal kleuren van de optimale kleuring.

| | File | # nodes | # edges | # colors needed |
|---|---|---|---|---|
| 1 | anna.col | 136 | 986 | 11 |
| 2 | david.col | 87 | 812 | 11 |
| 3 | fpsol2.i.1.col | 496 | 11654 | 65 |
| 4 | fpsol2.i.2.col | 451 | 8691 | 30 |
| 5 | fpsol2.i.3.col | 425 | 8688 | 30 |
| 6 | games120.col | 120 | 1276 | 9 |
| 7 | homer.col | 561 | 3258 | 13 |
| 8 | huck.col | 74 | 602 | 11 |
| 9 | inithx.i.1.col | 864 | 18707 | 54 |
| 10 | inithx.i.2.col | 645 | 13979 | 31 |
| 11 | inithx.i.3.col | 621 | 13969 | 31 |
| 12 | jean.col | 80 | 508 | 10 |
| 13 | le450_5a.col | 450 | 5714 | 5 |
| 14 | le450_5b.col | 450 | 5734 | 5 |
| 15 | le450_5c.col | 450 | 9803 | 5 |
| 16 | le450_5d.col | 450 | 9757 | 5 |
| 17 | le450_15a.col | 450 | 8168 | 15 |
| 18 | le450_15b.col | 450 | 8169 | 15 |
| 19 | le450_15c.col | 450 | 16680 | 15 |
| 20 | le450_15d.col | 450 | 16750 | 15 |
| 21 | le450_25a.col | 450 | 8260 | 25 |

Gaat verder op volgende pagina

| | | | | |
|---|---|---|---|---|
| 22 | le450_25b.col | 450 | 8263 | 25 |
| 23 | le450_25c.col | 450 | 17343 | 25 |
| 24 | le450_25d.col | 450 | 17425 | 25 |
| 25 | miles250.col | 128 | 774 | 8 |
| 26 | miles500.col | 128 | 2340 | 20 |
| 27 | miles750.col | 128 | 4226 | 31 |
| 28 | miles1000.col | 128 | 6432 | 42 |
| 29 | miles1500.col | 128 | 10396 | 73 |
| 30 | mulsol.i.1.col | 197 | 3925 | 49 |
| 31 | mulsol.i.2.col | 188 | 3885 | 31 |
| 32 | mulsol.i.3.col | 184 | 3916 | 31 |
| 33 | mulsol.i.4.col | 185 | 3946 | 31 |
| 34 | mulsol.i.5.col | 186 | 3973 | 31 |
| 35 | myciel3.col | 11 | 20 | 4 |
| 36 | myciel4.col | 23 | 71 | 5 |
| 37 | myciel5.col | 47 | 236 | 6 |
| 38 | myciel6.col | 95 | 755 | 7 |
| 39 | myciel7.col | 191 | 2360 | 8 |
| 40 | queen5_5.col | 25 | 320 | 5 |
| 41 | queen6_6.col | 36 | 580 | 7 |
| 42 | queen7_7.col | 49 | 952 | 7 |
| 43 | queen8_8.col | 64 | 1456 | 9 |
| 44 | queen8_12.col | 96 | 2736 | 12 |
| 45 | queen9_9.col | 81 | 2112 | 10 |
| 46 | queen10_10.col | 100 | 2940 | |
| 47 | queen11_11.col | 121 | 3960 | 11 |
| 48 | queen12_12.col | 144 | 5192 | |
| 49 | queen13_13.col | 169 | 6656 | 13 |
| 50 | queen14_14.col | 196 | 8372 | |
| 51 | queen15_15.col | 225 | 10360 | |
| 52 | queen16_16.col | 256 | 12640 | |
| 53 | school1.col | 385 | 19095 | |
| 54 | school1_nsh.col | 352 | 14612 | |
| 55 | zeroin.i.1.col | 211 | 4100 | 49 |
| 56 | zeroin.i.2.col | 211 | 3541 | 30 |
| 57 | zeroin.i.3.col | 206 | 3540 | 30 |

**Tabel 8.1: De grafen van de Second DIMACS Challenge**

Het is dus op deze grafen dat we de geïmplementeerde algoritmes hebben losgelaten. In de volgende paragrafen zullen we de resultaten tonen en interpreteren.

## 8.1 Resultaten van experimenten met RLF en variaties

In tabel 8.2 staan de resultaten van de experimenten met drie variaties van het RLF-algoritme, zoals ook beschreven in subsectie 7.3:

- RLF default
- RLF random y: kies willekeurig een knoop y
- RLF random x & y: kies willekeurig zowel knoop x als y

Het resultaat is steeds het beste uit een aantal runs: één run, vijf runs, tien, ... , zolang dat het algoritme het aantal runs kon uitvoeren in een aanvaardbare tijd.
De eerste vier kolommen bevatten informatie over de graaf: de naam, het aantal knopen, het aantal bogen en het minimale aantal kleuren dat nodig is voor een geldige kleur. De waarden die in vet gedrukt staan in deze laatste, zijn het chromatisch getal. De andere zijn het minimale aantal dat we gevonden hebben over alle algoritmes en alle runs.

We kunnen al onmiddellijk opmerken dat de algoritmes goed presteren en vaak een kleuring produceren met $\chi(G)$ kleuren. Verder zien we ook duidelijk dat de criteria voor het RLF-algoritme wel degelijk van belang zijn. Wanneer we de criteria laten vallen en willekeurig gaan kiezen, hebben we soms tot 50, 100 of zelfs 500 keer meer runs nodig om hetzelfde resultaat te halen. Enkele van deze gevallen zijn gemarkeerd. Ook is te zien dat er grafen zijn waarbij de waarde van het standaard RLF-algoritme nooit gehaald worden door zijn varianten. Enkele voorbeelden hiervan zien we binnen de vette kaders.

Onderaan de tabel tenslotte, staan per kolom het "genormaliseerd steekproefgemiddelde" ($\bar{x} = \sum \frac{x_i}{n}$ waarbij $x_i = \frac{\#kleuren}{\min}$ ) en de steekproefvariantie( $s^2 = \frac{1}{n-1}\sum \left(x_i - \bar{x}\right)^2$ ). Deze bevestigen onze eerdere observaties dat de criteria van het standaard RLF-algoritme werken.

We concluderen dus dat de criteria van RLF-algoritme werken. Het standaard RLF-algoritme presteert beter dan de variant waarin we enkel de y-knoop willekeurig kiezen. Deze presteert op zijn beurt dan weer beter dan de variant waarin we zowel de x- als y-knoop willekeurig kiezen.

| | # knopen | # bogen | *min* | RLF default: best out of | | | | | | RLF random y: best out of | | | | | | RLF random x & y: best out of | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 1 | 5 | 10 | 50 | 100 | 500 | 1 | 5 | 10 | 50 | 100 | 500 | 1 | 5 | 10 | 50 | 100 | 500 |
| 1. anna.col | 136 | 986 | **11** | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 12 | 11 | 11 | 11 | 11 | 11 |
| 2. david.col | 87 | 812 | **11** | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 13 | 11 | 12 | 11 | 11 | 11 |
| 3. fpsol2.i.1.col | 496 | 11654 | **65** | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 |
| 4. fpsol2.i.2.col | 451 | 8691 | **30** | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 |
| 5. fpsol2.i.3.col | 425 | 8688 | **30** | 30 | 30 | 30 | 30 | 30 | 30 | 32 | 30 | 30 | 30 | 30 | 30 | 31 | 31 | 30 | 30 | 30 | 30 |
| 6. games120.col | 120 | 1276 | **9** | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 10 | 9 | 9 | 9 | 9 | 9 |
| 7. homer.col | 561 | 3258 | **13** | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 15 | 14 | 14 | 13 | 13 | 13 |
| 8. huck.col | 74 | 602 | **11** | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 |
| 9. inithx.i.1.col | 864 | 18707 | **54** | 54 | 54 | 54 | 54 | 54 | 54 | 55 | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 |
| 10. inithx.i.2.col | 645 | 13979 | **31** | 31 | 31 | 31 | 31 | 31 | 31 | 32 | 31 | 31 | 31 | 31 | 31 | 32 | 31 | 31 | 31 | 31 | 31 |
| 11. inithx.i.3.col | 621 | 13969 | **31** | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 |
| 12. jean.col | 80 | 508 | **10** | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 13. le450_5a.col | 450 | 5714 | **5** | 8 | 7 | 8 | 7 | 7 | 7 | 13 | 12 | 12 | 12 | 12 | 12 | 13 | 12 | 12 | 12 | 12 | 12 |
| 14. le450_5b.col | 450 | 5734 | **5** | 7 | 7 | 7 | 7 | 7 | 7 | 13 | 13 | 12 | 12 | 12 | 12 | 13 | 13 | 13 | 12 | 12 | 12 |
| 15. le450_5c.col | 450 | 9803 | **5** | 5 | 5 | 5 | 5 | 5 | 5 | 17 | 15 | 16 | 16 | 15 | 15 | 16 | 16 | 16 | 15 | 15 | 14 |
| 16. le450_5d.col | 450 | 9757 | **5** | 5 | 5 | 5 | 5 | 5 | 5 | 17 | 16 | 15 | 15 | 15 | 15 | 17 | 16 | 15 | 15 | 15 | 14 |
| 17. le450_15a.col | 450 | 8168 | **15** | 17 | 16 | 16 | 16 | 16 | 16 | 22 | 21 | 20 | 20 | 20 | 20 | 22 | 21 | 21 | 21 | 21 | 20 |
| 18. le450_15b.col | 450 | 8169 | **15** | 16 | 16 | 16 | 16 | 16 | 16 | 22 | 21 | 20 | 20 | 19 | 20 | 22 | 21 | 21 | 21 | 21 | 20 |
| 19. le450_15c.col | 450 | 16680 | **15** | 23 | 23 | 22 | 22 | 22 | 22 | 30 | 29 | 29 | 28 | 29 | 28 | 32 | 31 | 30 | 30 | 30 | 29 |
| 20. le450_15d.col | 450 | 16750 | **15** | 23 | 23 | 23 | 23 | 23 | 22 | 29 | 29 | 29 | 29 | 29 | 28 | 32 | 30 | 30 | 30 | 30 | 30 |
| 21. le450_25a.col | 450 | 8260 | **25** | 25 | 25 | 25 | 25 | 25 | 25 | 27 | 27 | 27 | 27 | 27 | 26 | 29 | 29 | 28 | 28 | 27 | 27 |
| 22. le450_25b.col | 450 | 8263 | **25** | 25 | 25 | 25 | 25 | 25 | 25 | 26 | 27 | 27 | 26 | 26 | 26 | 28 | 28 | 28 | 27 | 27 | 27 |
| 23. le450_25c.col | 450 | 17343 | **25** | 28 | 28 | 27 | 28 | 27 | 27 | 36 | 35 | 34 | 34 | 33 | 34 | 37 | 36 | 36 | 36 | 35 | 35 |
| 24. le450_25d.col | 450 | 17425 | **25** | 29 | 28 | 28 | 27 | 27 | 27 | 35 | 35 | 35 | 34 | 34 | 33 | 38 | 37 | 36 | 36 | 35 | 35 |
| 25. miles250.col | 128 | 774 | **8** | 8 | 8 | 8 | 8 | 8 | 8 | 9 | 8 | 8 | 8 | 8 | 8 | 9 | 9 | 9 | 8 | 8 | 8 |
| 26. miles500.col | 128 | 2340 | **20** | 20 | 20 | 20 | 20 | 20 | 20 | 21 | 20 | 20 | 20 | 20 | 20 | 22 | 22 | 21 | 21 | 20 | 20 |
| 27. miles750.col | 128 | 4226 | **31** | 32 | 31 | 31 | 31 | 31 | 31 | 34 | 33 | 33 | 32 | 32 | 32 | 35 | 34 | 34 | 33 | 33 | 32 |
| 28. miles1000.col | 128 | 6432 | **42** | 42 | 42 | 42 | 42 | 42 | 42 | 43 | 42 | 42 | 42 | 42 | 42 | 46 | 45 | 45 | 44 | 44 | 42 |
| 29. miles1500.col | 128 | 10396 | **73** | 73 | 73 | 73 | 73 | 73 | 73 | 73 | 73 | 73 | 73 | 73 | 73 | 74 | 73 | 74 | 73 | 73 | 73 |
| 30. mulsol.i.1.col | 197 | 3925 | **49** | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 |
| 31. mulsol.i.2.col | 188 | 3885 | **31** | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 |
| 32. mulsol.i.3.col | 184 | 3916 | **31** | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 |
| 33. mulsol.i.4.col | 185 | 3946 | **31** | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 |
| 34. mulsol.i.5.col | 186 | 3973 | **31** | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 |
| 35. myciel3.col | 11 | 20 | **4** | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 36. myciel4.col | 23 | 71 | **5** | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 37. myciel5.col | 47 | 236 | **6** | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 38. myciel6.col | 95 | 755 | **7** | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| 39. myciel7.col | 191 | 2360 | **8** | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |

| | # knopen | # bogen | *min* | RLF default: best out of | | | | | | RLF random y: best out of | | | | | | RLF random x & y: best out of | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 1 | 5 | 10 | 50 | 100 | 500 | 1 | 5 | 10 | 50 | 100 | 500 | 1 | 5 | 10 | 50 | 100 | 500 |
| 40. queen5_5.col | 25 | 320 | **5** | 5 | 5 | 5 | 5 | 5 | 5 | 7 | 5 | 5 | 5 | 5 | 5 | 8 | 6 | 7 | 5 | 5 | 5 |
| 41. queen6_6.col | 36 | 580 | **7** | 8 | 7 | 8 | 8 | 7 | 7 | 9 | 8 | 8 | 8 | 8 | 7 | 10 | 8 | 9 | 8 | 8 | 8 |
| 42. queen7_7.col | 49 | 952 | **7** | 9 | 7 | 9 | 7 | 7 | 7 | 9 | 10 | 10 | 9 | 9 | 9 | 11 | 10 | 10 | 10 | 10 | 9 |
| 43. queen8_8.col | 64 | 1456 | **9** | 10 | 10 | 10 | 10 | 10 | 10 | 13 | 12 | 11 | 11 | 11 | 10 | 13 | 12 | 12 | 11 | 11 | 10 |
| 44. queen8_12.col | 96 | 2736 | **12** | 13 | 13 | 12 | 12 | 12 | 12 | 15 | 14 | 14 | 14 | 14 | 13 | 15 | 15 | 15 | 14 | 14 | 13 |
| 45. queen9_9.col | 81 | 2112 | **10** | 11 | 11 | 11 | 10 | 11 | 10 | 14 | 12 | 13 | 12 | 12 | 12 | 14 | 13 | 13 | 13 | 12 | 12 |
| 46. queen10_10.col | 100 | 2940 | 12 | 13 | 12 | 12 | 12 | 12 | 12 | 15 | 14 | 14 | 13 | 13 | 13 | 15 | 15 | 14 | 14 | 14 | 13 |
| 47. queen11_11.col | 121 | 3960 | **11** | 14 | 14 | 13 | 13 | 13 | 13 | 16 | 15 | 15 | 15 | 15 | 15 | 17 | 15 | 16 | 15 | 15 | 15 |
| 48. queen12_12.col | 144 | 5192 | 14 | 15 | 15 | 15 | 14 | 14 | 14 | 17 | 17 | 17 | 16 | 16 | 16 | 19 | 18 | 17 | 17 | 17 | 17 |
| 49. queen13_13.col | 169 | 6656 | **13** | 16 | 16 | 16 | 15 | 15 | 15 | 19 | 18 | 18 | 18 | 18 | 17 | 19 | 18 | 19 | 18 | 18 | 17 |
| 50. queen14_14.col | 196 | 8372 | 16 | 17 | 17 | 17 | 17 | 16 | 16 | 20 | 19 | 19 | 18 | 19 | 18 | 21 | 20 | 20 | 19 | 19 | 19 |
| 51. queen15_15.col | 225 | 10360 | 17 | 18 | 18 | 18 | 18 | 18 | 17 | 22 | 21 | 20 | 20 | 20 | 20 | 22 | 21 | 21 | 21 | 20 | 20 |
| 52. queen16_16.col | 256 | 12640 | 19 | 19 | 19 | 19 | 19 | 19 | 19 | 22 | 22 | 22 | 21 | 22 | 21 | 23 | 23 | 22 | 22 | 22 | 22 |
| 53. school1.col | 385 | 19095 | 17 | 26 | 25 | 26 | 25 | 25 | 25 | 37 | 38 | 36 | 36 | 34 | 31 | 45 | 40 | 42 | 39 | 41 | 39 |
| 54. school1_nsh.col | 352 | 14612 | 22 | 25 | 22 | 22 | 22 | 22 | 22 | 35 | 35 | 36 | 35 | 35 | 34 | 40 | 38 | 37 | 36 | 36 | 36 |
| 55. zeroin.i.1.col | 211 | 4100 | **49** | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 |
| 56. zeroin.i.2.col | 211 | 3541 | **30** | 30 | 30 | 30 | 30 | 30 | 30 | 32 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 |
| 57. zeroin.i.3.col | 206 | 3540 | **30** | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 |
| Genormaliseerd (steekproef)gemiddelde | | | | 1,075 | 1,057 | 1,064 | 1,05 | 1,048 | 1,043811 | 1,3098 | 1,266 | 1,257 | 1,244 | 1,24 | 1,22544 | 1,3572 | 1,301 | 1,301 | 1,267 | 1,263 | 1,24107 |
| (steekproef)Variantie | | | | 0,0228 | 0,018 | 0,022 | 0,017 | 0,017 | 0,01588 | 0,2973 | 0,253 | 0,241 | 0,239 | 0,223 | 0,21811 | 0,3084 | 0,271 | 0,26 | 0,237 | 0,242 | 0,21301 |

**Tabel 8.2: Resultaten RLF en variaties**

## 8.2 Resultaten van experimenten met DSatur en Cosine algoritmes

In tabel 8.3 staan de resultaten van de experimenten van het DSatur, Dutton & Brigham, Cosine en RLF-algoritme. We werken weer met markeringen en kaders, met dezelfde betekenissen reeds in subsectie 8.1 verklaard, om opvallende resultaten aan te duiden. We moeten echter melden dat de resultaten van het Dutton & Brigham-algoritme slechts een kleine steekproef van 1 run is. Daarom zullen we hierover geen conclusies trekken. We merken weinig verschillen in de resultaten tussen Cosine en DSatur. Enkel in graaf 53 is er een opvallend verschil in het voordeel van DSatur, echter zonder verklaring, want er is over deze graaf geen verdere informatie voor handen in het bestand, beschikbaar gesteld door DIMACS. Wel is het zo dat het "genormaliseerd gemiddelde" aanduidt dat DSatur beter presteert dan Cosine, maar ze moeten beiden de duimen leggen voor het RLF-algoritme. Opvallend zijn de 4 grafen, 13 t/m 16, waarbij het RLF-algoritme opmerkelijk betere resultaten behaald dan DSatur en Cosine. Slechts voor één graaf uit de verzameling, graaf 53, levert DSatur een beter resultaat dan RLF.

We concluderen dat het RLF-algoritme de beste resultaten boekt, gevolgd door het DSatur-algoritme en daarna het Cosine-algoritme.

## 8.3 Uitvoeringstijd van de algoritmes

In tabel 8.4 zien we de gemiddelde uitvoeringstijden van de verschillende algoritmes op de instanties van de Dimacs Challenge. We zien dat de twee varianten van het RLF-algoritme het snelste een resultaat boeken. De verdere volgorde, van snelste naar traagste, is DSatur-, Cosine-, RLF- en Dutton & Brigham-algoritme.

| | # knopen | # bogen | *min* | Cosine: best out of | | | D&B | Dsatur: best out of | | | | | RLF default: best out of | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 1 | 5 | 10 | 1 | 1 | 5 | 10 | 50 | 100 | 1 | 5 | 10 | 50 | 100 | 500 |
| 1. anna.col | 136 | 986 | **11** | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 |
| 2. david.col | 87 | 812 | **11** | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 |
| 3. fpsol2.i.1.col | 496 | 11654 | **65** | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 |
| 4. fpsol2.i.2.col | 451 | 8691 | **30** | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 |
| 5. fpsol2.i.3.col | 425 | 8688 | **30** | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 |
| 6. games120.col | 120 | 1276 | **9** | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 |
| 7. homer.col | 561 | 3258 | **13** | 14 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 |
| 8. huck.col | 74 | 602 | **11** | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 |
| 9. inithx.i.1.col | 864 | 18707 | **54** | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 |
| 10. inithx.i.2.col | 645 | 13979 | **31** | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 |
| 11. inithx.i.3.col | 621 | 13969 | **31** | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 |
| 12. jean.col | 80 | 508 | **10** | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 13. le450_5a.col | 450 | 5714 | **5** | 10 | 10 | 10 | 11 | 9 | 10 | 10 | 9 | 9 | 8 | 7 | 8 | 7 | 7 | 7 |
| 14. le450_5b.col | 450 | 5734 | **5** | 11 | 11 | 10 | 11 | 9 | 9 | 9 | 9 | 9 | 7 | 7 | 7 | 7 | 7 | 7 |
| 15. le450_5c.col | 450 | 9803 | **5** | 13 | 10 | 10 | 10 | 7 | 6 | 6 | 6 | 6 | 5 | 5 | 5 | 5 | 5 | 5 |
| 16. le450_5d.col | 450 | 9757 | **5** | 11 | 10 | 11 | 12 | 12 | 12 | 11 | 11 | 11 | 5 | 5 | 5 | 5 | 5 | 5 |
| 17. le450_15a.col | 450 | 8168 | **15** | 18 | 18 | 18 | 18 | 18 | 16 | 16 | 16 | 16 | 17 | 16 | 16 | 16 | 16 | 16 |
| 18. le450_15b.col | 450 | 8169 | **15** | 17 | 18 | 18 | 18 | 18 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
| 19. le450_15c.col | 450 | 16680 | **15** | 26 | 25 | 25 | 25 | 24 | 23 | 23 | 23 | 23 | 23 | 23 | 22 | 22 | 22 | 22 |
| 20. le450_15d.col | 450 | 16750 | **15** | 26 | 26 | 25 | 26 | 24 | 24 | 24 | 23 | 23 | 23 | 23 | 23 | 23 | 23 | 22 |
| 21. le450_25a.col | 450 | 8260 | **25** | 26 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 |
| 22. le450_25b.col | 450 | 8263 | **25** | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 |
| 23. le450_25c.col | 450 | 17343 | **25** | 32 | 31 | 30 | 30 | 29 | 29 | 28 | 28 | 28 | 28 | 28 | 27 | 28 | 27 | 27 |
| 24. le450_25d.col | 450 | 17425 | **25** | 32 | 30 | 30 | 30 | 29 | 28 | 28 | 28 | 28 | 29 | 28 | 28 | 27 | 27 | 27 |
| 25. miles250.col | 128 | 774 | **8** | 8 | 9 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
| 26. miles500.col | 128 | 2340 | **20** | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 |
| 27. miles750.col | 128 | 4226 | **31** | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 32 | 31 | 31 | 31 | 31 | 31 |
| 28. miles1000.col | 128 | 6432 | **42** | 43 | 43 | 43 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 |
| 29. miles1500.col | 128 | 10396 | **73** | 73 | 73 | 73 | 73 | 73 | 73 | 73 | 73 | 73 | 73 | 73 | 73 | 73 | 73 | 73 |
| 30. mulsol.i.1.col | 197 | 3925 | **49** | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 |
| 31. mulsol.i.2.col | 188 | 3885 | **31** | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 |
| 32. mulsol.i.3.col | 184 | 3916 | **31** | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 |
| 33. mulsol.i.4.col | 185 | 3946 | **31** | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 |
| 34. mulsol.i.5.col | 186 | 3973 | **31** | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 |
| 35. myciel3.col | 11 | 20 | **4** | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 36. myciel4.col | 23 | 71 | **5** | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 37. myciel5.col | 47 | 236 | **6** | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 38. myciel6.col | 95 | 755 | **7** | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| 39. myciel7.col | 191 | 2360 | **8** | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |

| | # knopen | # bogen | *min* | Cosine: best out of | | | D&B | Dsatur: best out of | | | | | RLF default: best out of | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 1 | 5 | 10 | 1 | 1 | 5 | 10 | 50 | 100 | 1 | 5 | 10 | 50 | 100 | 500 |
| 40. queen5_5.col | 25 | 320 | **5** | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 41. queen6_6.col | 36 | 580 | **7** | 9 | 8 | 8 | 9 | 10 | 9 | 9 | 9 | 9 | 8 | 7 | 8 | 8 | 7 | 7 |
| 42. queen7_7.col | 49 | 952 | **7** | 9 | 10 | 9 | 11 | 10 | 9 | 10 | 9 | 9 | 9 | 7 | 9 | 7 | 7 | 7 |
| 43. queen8_8.col | 64 | 1456 | **9** | 12 | 11 | 10 | 12 | 12 | 11 | 11 | 11 | 11 | 10 | 10 | 10 | 10 | 10 | 10 |
| 44. queen8_12.col | 96 | 2736 | **12** | 15 | 14 | 14 | 14 | 14 | 13 | 13 | 12 | 13 | 13 | 13 | 12 | 12 | 12 | 12 |
| 45. queen9_9.col | 81 | 2112 | **10** | 13 | 12 | 12 | 12 | 13 | 12 | 11 | 11 | 11 | 11 | 11 | 11 | 10 | 11 | 10 |
| 46. queen10_10.col | 100 | 2940 | 12 | 14 | 14 | 14 | 14 | 15 | 13 | 13 | 13 | 13 | 13 | 12 | 12 | 12 | 12 | 12 |
| 47. queen11_11.col | 121 | 3960 | **11** | 15 | 15 | 15 | 16 | 16 | 14 | 14 | 14 | 14 | 14 | 14 | 13 | 13 | 13 | 13 |
| 48. queen12_12.col | 144 | 5192 | 14 | 17 | 17 | 16 | 16 | 17 | 16 | 16 | 15 | 15 | 15 | 15 | 15 | 14 | 14 | 14 |
| 49. queen13_13.col | 169 | 6656 | **13** | 18 | 18 | 18 | 17 | 18 | 17 | 17 | 17 | 17 | 16 | 16 | 16 | 15 | 15 | 15 |
| 50. queen14_14.col | 196 | 8372 | 16 | 20 | 19 | 19 | 19 | 19 | 18 | 18 | 18 | 18 | 17 | 17 | 17 | 17 | 16 | 16 |
| 51. queen15_15.col | 225 | 10360 | 17 | 21 | 20 | 20 | 21 | 21 | 19 | 19 | 19 | 19 | 18 | 18 | 18 | 18 | 18 | 17 |
| 52. queen16_16.col | 256 | 12640 | 19 | 22 | 21 | 21 | 23 | 21 | 21 | 21 | 20 | 20 | 19 | 19 | 19 | 19 | 19 | 19 |
| 53. school1.col | 385 | 19095 | 17 | 28 | 31 | 25 | 32 | 17 | 17 | 17 | 17 | 17 | 26 | 25 | 26 | 25 | 25 | 25 |
| 54. school1_nsh.col | 352 | 14612 | 22 | 33 | 31 | 31 | 27 | 26 | 25 | 25 | 25 | 25 | 25 | 22 | 22 | 22 | 22 | 22 |
| 55. zeroin.i.1.col | 211 | 4100 | **49** | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 |
| 56. zeroin.i.2.col | 211 | 3541 | **30** | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 |
| 57. zeroin.i.3.col | 206 | 3540 | **30** | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 |
| Genormaliseerd gemiddeld | | | | 1,1991 | 1,177 | 1,161 | 1,1902 | 1,148 | 1,12 | 1,116 | 1,105 | 1,107 | 1,075 | 1,057 | 1,064 | 1,05 | 1,048 | 1,04381 |
| (steekproef)Variantie | | | | 0,1211 | 0,093 | 0,085 | 0,1164 | 0,0689 | 0,069 | 0,061 | 0,054 | 0,053 | 0,0228 | 0,018 | 0,022 | 0,017 | 0,017 | 0,01588 |

**Tabel 8.3: Resultaten Experimenten met RLF, Cosine, Dutton & Brigham, DSatur algoritmes**

| | RLF | RLFy | RLFxy | Dutton & Brigham | Cosine | Dsatur |
|---|---|---|---|---|---|---|
| anna.col | 24.9 | 1.5 | 7.9 | 54.6 | 37.8 | 10.9 |
| david.col | 7.7 | 1.6 | 1.5 | 13.7 | 5.3 | 7.7 |
| fpsol2.i.1.col | 1096.9 | 34.5 | 32.8 | 4457.9 | 1.374 | 598.5 |
| fpsol2.i.2.col | 417.3 | 25.1 | 28.2 | 1976.5 | 1.124 | 339.1 |
| fpsol2.i.3.col | 334.5 | 18.6 | 28.2 | 1643.6 | 995 | 312.6 |
| games120.col | 18.5 | 1.5 | 3.2 | 31.2 | 26 | 0 |
| homer.col | 2856.5 | 62.6 | 54.8 | 3856.3 | 2.659 | 109.4 |
| huck.col | 6.2 | 0 | 0 | 6.3 | 7 | 6.2 |
| inithx.i.1.col | 5512.5 | 112.7 | 128.2 | 20203 | 7911.5 | 1350 |
| inithx.i.2.col | 1567.3 | 73.4 | 78.2 | 5948.5 | 4001.3 | 748.4 |
| inithx.i.3.col | 1368.7 | 51.6 | 62.5 | 5265.5 | 4.021 | 704.7 |
| jean.col | 7.9 | 1.6 | 0 | 7.8 | 4 | 3.1 |
| le450_5a.col | 989 | 29.9 | 54.7 | 1962.5 | 962 | 239.1 |
| le450_5b.col | 1003 | 37.4 | 29.8 | 1912.6 | 968.5 | 203 |
| le450_5c.col | 912.4 | 28.1 | 29.8 | 1912.6 | 904 | 359.3 |
| le450_5d.col | 939.3 | 31.4 | 29.8 | 1948.4 | 902 | 367.5 |
| le450_15a.col | 708 | 32.7 | 32.9 | 1921.9 | 694 | 317.2 |
| le450_15b.col | 725.2 | 43.9 | 40.7 | 1911 | 709.8 | 300.1 |
| le450_15c.col | 639.1 | 26.5 | 26.7 | 1924.8 | 647 | 726.8 |
| le450_15d.col | 635.9 | 28.3 | 31.3 | 1973.5 | 664.3 | 750 |
| le450_25a.col | 587.6 | 28 | 31.1 | 1917.3 | 583 | 335.9 |
| le450_25b.col | 639.3 | 34.5 | 37.5 | 1940.7 | 650 | 315.5 |
| le450_25c.col | 561 | 28.1 | 28.2 | 1939 | 583 | 790.5 |
| le450_25d.col | 559.3 | 26.6 | 35.8 | 1925.1 | 589 | 782.9 |
| miles250.col | 26.6 | 1.6 | 1.6 | 42.1 | 27 | 6.1 |
| miles500.col | 23.6 | 3.1 | 3.2 | 40.7 | 21 | 15.7 |
| miles750.col | 14.2 | 1.6 | 1.5 | 42.1 | 22.3 | 37.5 |
| miles1000.col | 14.2 | 1.6 | 1.5 | 36 | 19.8 | 67 |
| miles1500.col | 1.6 | 1.6 | 0 | 31.2 | 23.5 | 140.8 |
| mulsol.i.1.col | 45.2 | 1.6 | 7.9 | 156.4 | 76 | 93.6 |
| mulsol.i.2.col | 31.3 | 6.2 | 4.7 | 126.6 | 69 | 90.5 |
| mulsol.i.3.col | 26.6 | 1.6 | 4.7 | 118.6 | 75.5 | 101.5 |
| mulsol.i.4.col | 23.5 | 4.6 | 1.6 | 123.3 | 73 | 87.3 |
| mulsol.i.5.col | 28 | 1.5 | 4.7 | 123.4 | 75 | 78.2 |
| myciel3.col | 0 | 0 | 0 | 0 | 0 | 0 |
| myciel4.col | 0 | 0 | 0 | 0 | 0 | 1.6 |
| myciel5.col | 3.2 | 1.6 | 1.6 | 6.4 | 4 | 1.5 |
| myciel6.col | 4.8 | 0 | 0 | 15.6 | 4 | 7.8 |
| myciel7.col | 53.1 | 4.6 | 4.7 | 159.6 | 60 | 40.6 |
| queen5_5.col | 0 | 0 | 0 | 1.5 | 0 | 0 |
| queen6_6.col | 0 | 0 | 0 | 1.6 | 1.3 | 3.2 |
| queen7_7.col | 1.6 | 0 | 0 | 1.5 | 1 | 3.2 |
| queen8_8.col | 3.1 | 0 | 0 | 7.9 | 1.3 | 7.9 |
| queen8_12.col | 12.5 | 1.6 | 1.6 | 17.3 | 10.5 | 18.9 |
| queen9_9.col | 11.1 | 1.6 | 0 | 10.9 | 9 | 12.4 |
| queen10_10.col | 10.8 | 0 | 1.6 | 18.7 | 5.3 | 19 |
| queen11_11.col | 15.7 | 1.6 | 1.5 | 35.8 | 17 | 31.5 |
| queen12_12.col | 28.2 | 1.6 | 4.5 | 59.3 | 26 | 45.3 |
| queen13_13.col | 46.8 | 7.9 | 3 | 95.2 | 44.5 | 62.6 |
| queen14_14.col | 62.6 | 4.6 | 4.7 | 148.2 | 67.8 | 89.2 |
| queen15_15.col | 97.1 | 9.4 | 9.2 | 224.9 | 99 | 120.4 |
| queen16_16.col | 140.7 | 6.2 | 12.7 | 334.4 | 146 | 158.2 |
| school1.col | 267.1 | 23.3 | 20.1 | 1184.3 | 330.8 | 860.9 |
| school1_nsh.col | 243.8 | 17.2 | 17.3 | 879.5 | 280 | 584.2 |
| zeroin.i.1.col | 64 | 8 | 7.8 | 212.5 | 82 | 109.1 |
| zeroin.i.2.col | 42.3 | 7.7 | 3.1 | 185.8 | 81 | 79.6 |
| zeroin.i.3.col | 36.1 | 3.1 | 1.6 | 175.3 | 76.8 | 76.6 |
| Totale Tijd | 23497.4 | 885.1 | 960.2 | 71270.9 | 32851 | 12730.3 |

**Tabel 8.4: Uitvoeringstijden van de algoritmes**

# Conclusie

In deze masterproef bestudeerden we het graafkleurenprobleem. Dit probleem is voor vele mensen, misschien onbewust, geen onbekend probleem. De één kwam er mee in aanraking in de vorm van de vierkleurenstelling, de andere in de vorm van boogkleuren. Maar allemaal leerden ze de moeilijkheid van het probleem kennen. Daarom ook bestudeerden we enkele heuristische algoritmes die de oplossing zo goed mogelijk proberen te benaderen.
We implementeerden en experimenteerden met enkele van deze algoritmes, met het doel te zien of er een algoritme is dat de beste resultaten boekt.
Maar de grafen waarop we onze experimenten uitvoerden, leverden ons geen uitsluitsel welke van de geïmplementeerde algoritmes nu het beste is om te gebruiken indien we een kleurenprobleem willen oplossen. We kunnen dus enkel besluiten dat het graafkleuren-probleem, hoewel we enkele mogelijkheden voor handen hebben om het op te lossen, een aartsmoeilijk probleem is en blijft.

# Bronvermeldingen

[1] Diestel. Reinhard, Graph theory. Springer-Verlag, New York, 2005. 3rd edition, second edition 2000, first edition 1997.

[2] Klotz, Walter, Graph Coloring Algorithms (1999).

[3] de Werra, D. Heuristics for graph coloring. In Computational graph theory, G. Tinhofer, E. Mayoraz, H. Noltemeir, and M. Syslo, Eds. Springer Verlag NY, 1989, pp. 171-185.

[4] Schrijver, Alexander, Grafen: Kleuren en Routeren,

[5] Misra, J. and Gries, D., A Constructive Proof of Vizing's Theorem, Inform. Process. Lett. 41, 131-133, 1992.

[6] C. E. Shannon, A theorem on coloring the lines of a network, J. Math. Phys. 28 (1949), 148-151

[7] Kőnig, Dénes (1916). Gráfok és alkalmazásuk a determinánsok és a halmazok elméletére. Matematikai és Természettudományi Értesítő 34: 104–119

[8] Graphs, Networks and Design - Graphs 3: Planarity and Colouring (Block 3) by Open University Course Team, Softcover, Open University Worldwide

[9] K. Appel and W. Haken, Every planar map is four colorable, Contemporary Math. 98 (1989)

[10] Karp, Richard M. (1972). Reducibility among combinatorial problems, in Raymond E. Miller and James W. Thatcher (editors): Complexity of Computer Computations. New York: Plenum, 85–103

[11] Zykov, A. A., On some properties of linear complexes, Amer. Math. Soc. Translations 79 (1952), p. 81.

[12] Brigham, R. D. & Dutton, R. D., A new graph coloring algorithm, The Computer Journal 24 (1981), 85-86.

[13] Hertz, A., A fast algorithm for coloring Meyniel graphs, Journal of Combinatorial Theory B 50 (1990), 231-240.

[14] Leighton, F. T., A graph coloring algorithm for large scheduling problems, Journal of Research of the National Bureau of Standards 84 (1979),489-503.

[15] Brélaz, D., New methods to color the vertices of a graph, Communications of the Assoc. of Comput. Machinery 22 (1979), 251-256.

[16] Gamma, Erich & Helm, Richard & Johnson, Ralph & Vlissides, John M., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional Computing Series

[17] Java™ Platform, Standard Edition 6 API Specification,
http://java.sun.com/javase/6/docs/api/

[18] Volume Twenty Six: Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge" Editors: David S. Johnson and Michael A. Trick, 1996

[19] Clique and coloring problems: Graph format
http://www.dimacs.rutgers.edu