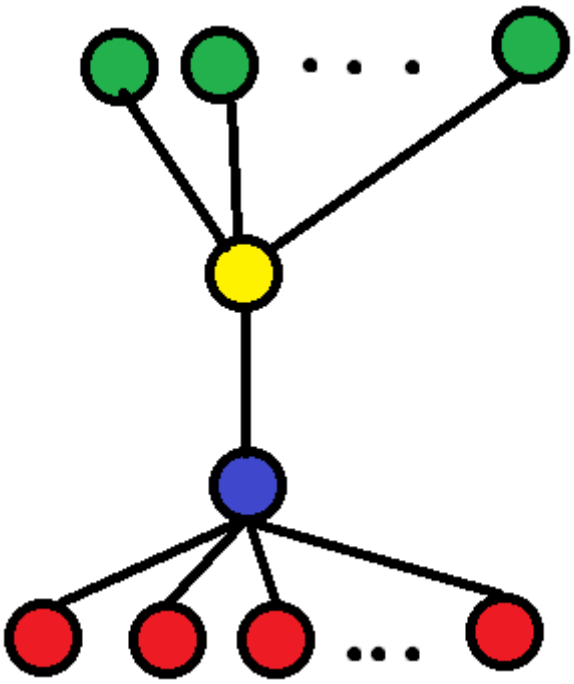
Algoritmen & Datastructuren II

Domien Van Steendam

VRAAG 1

Beschouw volgende graaf:



In bovenstaande graaf wil ik duidelijk maken dat er een arbitrair aantal groene en rode toppen zijn. We zorgen ervoor dat de blauwe top steeds degene is met de hoogste graad door het aantal rode toppen strikt groter te maken (desnoods slechts 1 top meer) dan het aantal groene toppen.

De **optimale dominerende verzameling** bestaat ongeacht bovenstaande aantallen **steeds uit 2 dezelfde toppen**: de blauwe en de gele. Alle rode toppen zijn een buur van de blauwe top en alle groene toppen zijn een buur van de gele. Met geel en blauw kunnen we dus elke top bereiken. Gegeven een positief getal k, kies het aantal groene toppen 100k en het aantal rode 100k+1 (of meer). *(Ik kies voor het extreem geval 100 k, omdat ik vind dat extreme gevallen zeer duidelijk en gemakkelijk te begrijpen zijn.)*

Als we het gegeven algoritme toepassen:

1) We starten bij de blauwe top want die heeft graad 100k+2. (De gele heeft graad 100k+1)
2) We voegen de blauwe top toe aan D en verwijder de gele en de rode toppen uit deze graaf.
3) Vervolgens blijven enkel de groene toppen over in de graaf. De groene toppen hebben alle graad 0. Ze worden een voor een bezocht en verwijderen enkel zichzelf uit de graaf, want ze hebben geen buren meer, want de gele top is verwijderd in stap 2.

Uiteindelijk bevat D de blauwe en alle 100k groene toppen. Dit zijn 100k+1 elementen en dat is duidelijk groter dan 2k.

Vraag 2 *Bewijs dat voor een Hamiltoniaanse cykel in een vlakke triangulatie er evenveel vlakken binnen als buiten de cykel liggen.*

Bewijs: Voor een vlakke graaf met n toppen en een Hamiltoniaanse cykel C geldt vanwege de grinbergs stelling [1] het volgende:

$$\sum_{k\geq 3}^{n}(\mathrm{k}-2)(\,r_k-\,r'_k)=0$$

waarbij r_k ,r'_k het aantal k-vlakken zijn (vlakken die zijn afgebakend door k toppen en k bogen) binnen respectievelijk buiten de Hamiltoniaanse cykel C.
Gegeven een Hamiltoniaanse, vlakke triangulatie T: T is tevens een vlakke, Hamiltoniaanse graaf en bijgevolg geldt de grinbergs stelling ook voor T. Bijgevolg weten we dat een vlakke triangulatie alleen driehoekige vlakken bevat en dus zijn r_k ,r'_k gelijk aan 0 voor alle $k \neq 3$.

De grinbergs stelling kunnen we voor vlakke, Hamiltoniaanse triangulaties (waaronder T) met een Hamiltoniaanse cykel C vereenvoudigen tot($r_3 - r'_3) = 0 \equiv r_3 = r'_3$. Hieruit volgt het gestelde: er zijn evenveel (driehoekige) vlakken binnen als buiten de cykel.

Qed.

Een alternatief bewijs bestaat eruit de formule van Euler (waarop Grinbergs stelling is gebaseerd), de formules voor het aantal bogen (3n-6) en het aantal vlakken (2n-4) te gebruiken.

Vraag 3 + bespreking verduidelijkingen implementatie + mogelijke optimalisaties

Dominantie in vlakke grafen
Het basisidee is als volgt:

We beginnen met een lege verzameling D. Ik voeg een voor een toppen de top met de hoogste graad toe aan D die nog niet een keer is toegevoegd.

Fase1: We kijken vervolgens of er meer toppen bereikbaar zijn, t.t.z. of het aantal toppen die D domineert gestegen is. Indien dit niet zo is, verwijderen we deze top weer uit D, want hij heeft toch geen effect.

Fase2: Na elke top aan de beurt is geweest domineert D de volledige graaf. Vervolgens gaan we achterstevoren toppen verwijderen uit D. Dit wil zeggen: we beginnen met de top in D met de laagste graad t.e.m. degene met de hoogste graad. Elke top trachten we te verwijderen zonder teniet te doen aan de dominantie-eigenschap. Na het verwijderen van zo'n top moeten alle n toppen nog steeds gedomineerd worden door D.

Helemaal achteraan dit verslag vindt u een codeflowchart van het algoritme hieronder besproken.

Pseudocode:

Voor de duidelijkheid geef ik volgende definities die ik zal gebruiken in de pseudocode:

- *Indien een top t gedomineerd wordt door D is dit equivalent aan* $t \in D \; of \; t \; is \; adjacent \; aan \; D.$
- *Voor een graaf G (V, E) en een top t* $\in$ V *definieer de buurverzameling N(t) = {u* $\in$ *V |* {u, t} $\in$ E*}.*

INVOER: Een lijst L van alle toppen uit een graaf G (V, E);
Maak een initieel lege lijst D aan.

// Fase 1
Sorteer de lijst van toppen op graad van groot naar klein (of klein naar groot). **(1)**
For (elke top t startende van degene met de hoogste graad t.e.m. de kleinste) { **(2)**

Als de volledige graaf al gedomineerd wordt door D **(3)**

Verbreek de lus

Indien nog niet alle toppen van de graaf gedomineerd worden door D

Indien t nog niet gedomineerd werd door D, voeg t aan D toe. **(4)**
Indien t wel al gedomineerd werd door D **(5)**
Kijk of het toevoegen van t ervoor zorgt dat het totaal aantal gedomineerde toppen van de graaf toeneemt. **(5)**
Indien dit zo is, voeg t toe aan D. **(5)**

}

// Fase 2
// Nu wordt de volledige graaf gedomineerd door D, we proberen nu zoveel mogelijk toppen te
//verwijderen uit D

For (elke top t startende van degene met de **laagste graad t.e.m. de grootste**) { **(6)**

Indien t niet gedomineerd wordt door $D\backslash\{t\}$ **(7)**
Continue;

Indien $\exists u \in \mathrm{N(t)}$: u wordt niet gedomineerd door $D\backslash\{t\}$ **(8)**
Continue;

Verwijder t uit D. **(9)**

UITVOER: de dominantieverzameling D van graaf G.

Opmerkingen:

- **(1)** We maken gebruik van counting sort die een lineaire tijdscomplexiteit heeft.
- **(2)** De body van de lus wordt uitgevoerd in O(deg(t)) (zie **(5)**). In het slechtste geval overlopen we iedere top uit de graaf. De tijdscomplexiteit van de volledige lus is dan $\sum_{t\in G} \deg(t)$. We werken echter met een vlakke graaf en dus is deze som begrensd door O(n). De lus is bijgevolg lineair in het aantal toppen.
- **(3)** Aan de hand van een integer variabele houden we het aantal gedomineerde toppen bij. Deze expressie evalueren kan in constante tijd.
- **(4)** Voor iedere top houden we een boolean bij waaruit we kunnen afleiden of deze top al dan niet gedomineerd wordt door D. In de implementatie gebruiken we een integer met het aantal verwijzingen vanuit D. Indien deze 0 is, wordt deze top niet gedomineerd. We veranderen bij het toevoegen van t iedere buur aan. Dit heeft kost O(deg(t)).
- **(5)** We bekijken hier in het slechtste geval alle buren van t indien t niet toegevoegd wordt. Indien blijkt dat we hem wel zouden toevoegen, moeten we bovendien nog eens iedere buur overlopen om hun boolean (zie **(4)**), indien nog niet aangevinkt, aan te passen. Dit heeft kost O(deg(t)).
- **(6)** Analoog aan **(2)** zal deze lus in lineaire tijd uitgevoerd kunnen worden. De body heeft een kost van O(deg(t)).
- ***(7)*** Dit kan in constante tijd aangezien we het aantal verwijzingen vanuit D bijhouden. *Als deze 1 is, weten we dat deze top niet kan verwijderd worden. (De verwijzing naar zichzelf wordt meegeteld).*
- **(8)** We overlopen iedere buur van t met een kost van O(deg(t)).

(9) Analoog aan het toevoegen passen we hier de dominantie boolean (en verwijzingen) van iedere buur aan met een kost van O(deg(t)).

Uit bovenstaande opmerkingen volgt dat beide lussen lineair in het aantal toppen kunnen uitgevoerd worden. Het sorteren van de toppen op graad en de twee lussen worden in serie uitgevoerd en dus is dit algoritme bijgevolg lineair in het aantal toppen.

Gretig karakter:

Dit algoritme is **gretig** in de zin dat we de toppen met hoogste graad prefereren. Lokaal gezien zal een top met de hoogste graad die top zijn die het meest dominant is. Fase 1 zoals hierboven beschreven zal ons met weinig toppen veel andere toppen laten bereiken vanwege de hoge graad.
Bij het verwijderen gedurende fase 2 zullen we starten met degene met de laagste graad, omdat die lokaal gezien het minst dominant zijn. Hoe minder buren, hoe minder de kans dat we een buur hebben die niet adjacent is aan D zonder de huidige top. We beginnen **niet** met fase 2 van hoog naar lage graad, want dit kan ervoor zorgen dat we één top verwijderen die er vervolgens voor zorgt dat we 2 andere toppen met lagere graad niet meer kunnen verwijderen. Van laag naar hoog zullen we eerst deze 2 hebben verwijderd. Dit is voordeliger voor een kleinere grootte van onze dominantieverzameling.

Hieronder volgen naast deze opmerkingen nog enkel verduidelijkingen.

Extra Verduidelijkingen in verband met mijn implementatie (zie ook documentatie):
Zoals in **(4)** al vermeld gebruik ik een integer per top bij voor het aantal verwijzingen. Er is tevens ook een globaal referentiegetal. Deze houdt bij hoeveel toppen gedomineerd worden door D. Telkens een top zijn lokaal referentiegetal van 0 naar 1 verandert, stijgt de globale ook. Alle verdere verhogingen van het referentiegetal van diezelfde top, veranderen het globale referentiegetal niet, want die top wordt al gedomineerd. Analoog zal bij het veranderen van 1 naar 0, de globale gedecrementeerd worden.
Het booleanveld “gemarkeerd” in de klasse Node is true indien die top tot de dominantieverzameling hoort.
We houden tevens een integer veld “graad” bij om in constante tijd de graad van de top op te vragen.
De meeste velden zijn public om de overhead aan getter/setter-oproepen te elimineren*. Ik heb alles zelf geschreven en weet tenslotte wat ik doe dus kan het geen kwaad om alle velden public te maken.*

Mogelijke optimalisaties:

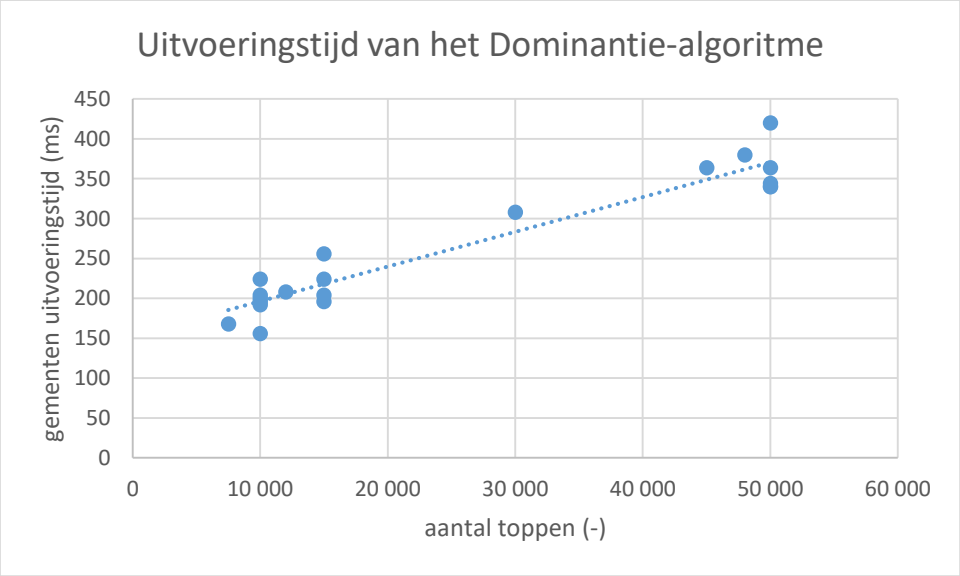
- Ik heb online een publicatie [2] gevonden waarin men de graaf reduceert naar een graaf met minder toppen, maar met dezelfde minimale dominantieverzameling aan de hand van 2 reduceer regels. De eerste reduceerregel zal die toppen verwijderen die geen bijdrage leveren aan de dominantieverzameling aangezien hun buren (met hogere graad) diezelfde toppen al domineert. De tweede reduceeregel bouwt voort op de eerste. Voor meer informatie raad ik zeker de paper aan (vanaf pagina 5). Beide regels kunnen in lineaire tijd uitgevoerd worden.
  Als optimalisatie voor dit algoritme zou ik vooraf deze reductieregels toepassen en daarna het huidige algoritme. Door eerst te reduceren elimineren we de toppen die er al zeker niet in mogen en dus ook de kans dat we die kiezen in het huidige algoritme is 0, want die toppen zijn verwijderd in de gereduceerde graaf.

Experimenten i.v.m. het lineair gedrag:

1- Als verificatie van het lineair gedrag, werd het dominantie-algoritme uitgevoerd en getimed op de meegegeven grafen testgrafen/testset/*.sec
We bekomen volgende gemeten resultaten:

Graafnummer: graafX.sec	Aantal toppen	Tijd (ms)
1	10 000	204
2	50 000	420
3	10 000	192
4	50 000	340
5	7500	168
6	30 000	308
7	12 000	208
8	45 000	364
9	10 000	156
10	10 000	224
11	10 000	200
12	10 000	196
13	50 000	340
14	50 000	364
15	50 000	344
16	48 000	380
17	15 000	196
18	15 000	256
19	15 000	204
20	15 000	224

Deze resultaten werden vervolgens geplot d.m.v. Excel:



De trendlijn verloopt voornamelijk lineair en verifieert bijgevolg onze hypothese.

Hamiltoniaanse cykels in vlakke triangulaties

Hieronder vindt u mijn algoritme in pseudocode:

INVOER: Een lijst L van alle toppen uit een graaf G (V, E);

Sorteer de lijst van toppen op graad van groot naar klein (of klein naar groot). **(1)**
Iedere top start met een oneindig grote fitness.
start = top uit V met de hoogste graad **tenzij** er een top met graad 2 is dan kiest u die **(2)**
Ken de start fitness 0 toe.
Pad = lijst met eerste element de starttop 'start'.
// Voortaan bedoelen we met 'kop' of 'de kop' die top die het laatst toegevoegd is aan het pad.
// Deze top stelt dus de kop voor van ons pad. In het begin is deze uiteraard gelijk aan start.

WHILE de lengte van het pad is niet gelijk aan het aantal toppen **(3)**
DO

IF kop heeft geen buren (die nog niet op het pad liggen) **(4)**
Break deze lus
ENDIF
IF padlengte < n-1 **AND** kop een buur heeft die enkel start en kop als buur heeft **(10)**
Undo de laatste keuze van kop (de huidige kop dus) en geef deze kop terug 'vrij'
Herstel de verwijdering van de vorige kop dus ook.
fitness(kop) := fitness(kop) -1
Herhaal de vorige iteratie van de lus
ENDIF
Zoek die buur v van de kop waarbij je volgende criteria van boven naar onder evalueert. Bij ex aequo van 2 of meer toppen evalueer je de volgende eigenschap. **(5)**

1. Hoogste fitness
2. Laagste graad
3. Indien alle vorige eigenschappen nog steeds ex aequo opleveren, kies willekeurige uit de niet-geëlimineerde toppen.

FOR iedere buur b van de kop **(6)**

fitness(b) := min(fitness(b), fitness(kop))

ENDFOR

fitness(v) := min(fitness(v), fitness(kop)+1) **(7.1)**
Verwijder de kop uit graaf G **(7.2)**
Voeg v toe aan pad. // Merk op: hieruit volgt impliciet dat kop := v **(7.3)**

ENDWHILE

IF de kop oorspronkelijk een buur is van start **(8)** en iedere top opgenomen is in het pad **(9)**
Return pad
ELSE
Return "geen pad gevonden"
ENDIF

Opmerkingen:

(1) We maken gebruik van counting sort die een lineaire tijdscomplexiteit heeft.

(2) Dit kan in constante tijd gebruikmakend van de gesorteerde lijst uit **(1).**
Een top met graad 2 is ideaal om te starten, want daar kan je onmogelijk een verkeerde keuze voor de volgende top maken aangezien beide bogen steeds bij iedere mogelijke cykel horen.

(3) Uit de hieropvolgende opmerkingen volgt dat één uitvoering van de body van de lus met huidige kop k een kost heeft van O(deg(k)). We nemen namelijk de hoogste complexiteit tussen **(4), (5) en (6)**. De kop verandert per loop en is steeds een niet-bezochte top, t.t.z. een top die nog niet op het pad ligt. Een top is dus hoogstens 1 maal de kop. De lus kan dus in het slechtste geval n keer uitgevoerd worden. Hieruit volgt dat in het slechtste geval de volledig lus een kost heeft van $\sum_{t \in G} \deg(t)$. In een vlakke triangulatie is deze som begrens door O(n).

(4) Als de vorige koppen effectief uit de graaf verwijderd werden, zullen enkel de niet-bezochte toppen nog buren zijn van de huidige kop. Men kan deze conditie in constante tijd nagaan door te checken of er minstens 1 buur is.
Indien men de vorige koppen niet werkelijk verwijdert, kan dit alsnog in O(deg(kop)) geëvalueerd worden.

(5) Men overloopt enkel de buren van de kop. Dit heeft kost O(deg(kop)).

(6) De body heeft duidelijk een constante kost, want de minimum oproep heeft ook een constante kost. De volledig lus kost bijgevolg O(deg(kop)).

(7) Het verwijderen in **(7.2)** overloopt iedere buur van de kop dus O(deg(kop)) en verwittigt iedere buur <u>in constante tijd</u> van de verwijdering. **(7.2)** heeft dus een kost van O(deg(kop)). **(7.3)** kan simpelweg in een constante kost bij het gebruik van een geschikte datastructuur zoals een LinkedList (deze werd gebruikt in mijn implementatie). **(7)** heeft een totale tijdscomplexiteit van O(deg(kop)).

(8) Opnieuw een kost van O(deg(kop)), want men overloopt de buren van kop tot een match met start.

(9) Dit kan in constante tijd gecheckt worden met bijvoorbeeld een integer die het aantal opgenomen topen bijhoudt. Dus kost O(1).

(10) We voeren de inverse bewerking toe van het verwijderen van een top: we overlopen iedere oorspronkelijk buur en laten weten dat de kop weer 'vrij' is. Dit heeft een kost van O(deg(kop)).

MaxComplexiteit((1), (8), (9)) = O(n). Het volledige algoritme heeft een tijdscomplexiteit van $O(n)$ met n het aantal toppen.

Mogelijke optimalisaties:

- Het pad laten groeien vanuit beide eindpunten. Nu zoeken we vanuit de kop naar nieuwe punten tot en met we vastzitten of bij start uitkomen.

<u>Gretig karakter + verduidelijkingen implementatie:</u>

Indien er een top met graad 2 is, startten we daar, omdat we daar geen verkeerde buren kunnen kiezen, want beide bogen horen steeds bij elke mogelijke Hamiltoniaanse cykel.
Bijgevolg kiezen we steeds voor een top die zo ver mogelijk van de start ligt zodat we niet geïsoleerd rond start ronddwalen en mogelijks een extern gebied uitsluiten. Dit heb ik gerealiseerd met het concept "fitness". Hoe hoger de fitness van een top, hoe aantrekkelijker hij is om gekozen te worden. De start heeft fitness 0 en al zijn buren krijgen fitness 1. Bij selectie van een nieuwe kop krijgen alle

buren van de kop de fitness van de kop + 1. Bij een ex aequo van de fitness kiezen we voor de laagste graad, want hoe minder bogen hoe minder kans op het kiezen van een foute boog.

Algemene implementatie bespreking

De specifieke implementatie voor de dominantieverzameling en Hamiltoniaanse cykels maken gebruik van enkele gemeenschappelijke klassen. Deze klassen worden in deze sectie besproken.

Package graaf

Node

Dit is de klasse die een top van een graaf voorstelt. Een graaf wordt voorgesteld als een rij van instanties van deze klasse Node. Instanties van deze klasse bevat enkele velden specifiek voor het vinden van de dominantieverzameling en Hamiltoniaanse cykels opdat de algoritmen sneller uitgevoerd kunnen worden. De methode add/delete-Reference worden gebruikt door het dominantieverzameling-algoritme zoals besproken in theoretische vraag 3.

Sorteerder

Een sorteerder kan een rij van Nodes (en dus ook een graaf) sorteren op hun iD-nummer en hun graad. Beide kunnen in O(n) uitgevoerd worden zoals besproken in de documentatie aldaar.

Package sec

SimpleEdgeCodereader

Dit is de interface voor iedere lezer van bestanden in SEC-formaat.

SecReader

Deze klasse implementeert de SimpleEdgeCodeReader interface. Per read()-oproep wordt exact 1 graaf uitgelezen. De parsing gebeurt door middel van een rij met evenveel entries als bogen. Indien een boognummer voor de eerste keer ingelezen wordt, wordt de top (van klasse Node) in de rij gestoken op dezelfde index als boognummer. Als het boognummer de volgende keer uitgelezen wordt, zit er al een top op die index. Men kan bijgevolg de vorige Node en de huidige met elkaar verbinden. Op die manier zal ieder paar adjacente toppen elkaar terugvinden.

Experimenten

In deze sectie ga ik verschillende ideeën voor de algoritmen vergelijken. Hieronder kunnen ook enkele van de bovenstaande optimalisaties doorgevoerd worden. In de zipfile werd de output van ieder experiment meegegeven, u kan deze vinden in de “Experimenten” folder. De bestandsnamen zijn zo gekozen opdat men duidelijk weet bij welk experiment ze horen.

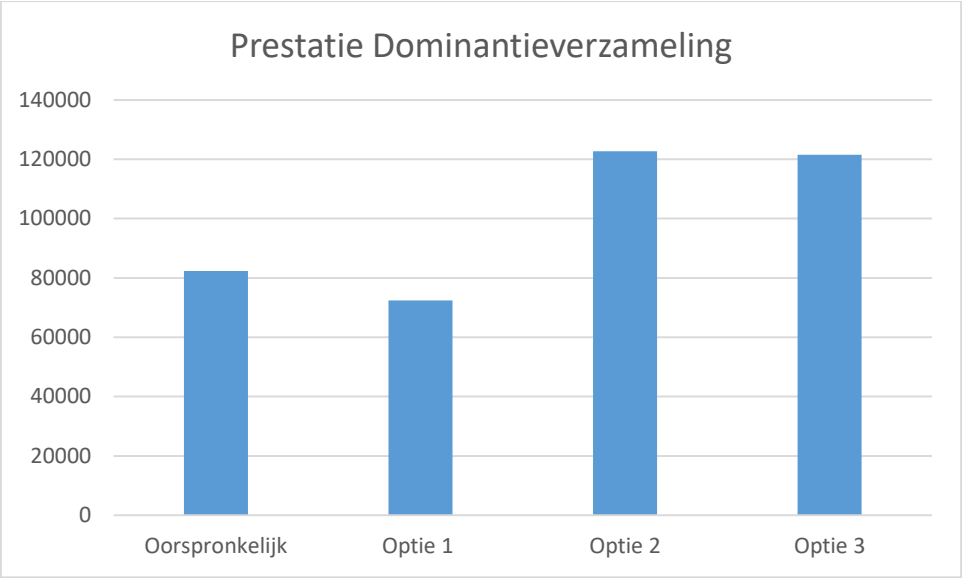
Dominantie in vlakke grafen

Momenteel voegen we toe van hoogste naar kleinste graad en verwijderen we van laagste naar hoogste graad. Er zijn nog 3 andere combinaties namelijk:

1. Toevoegen: Hoog naar laag
   Verwijderen: Hoog naar laag
2. Toevoegen: Laag naar Hoog
   Verwijderen: Hoog naar laag

3. Toevoegen: Laag naar Hoog
   Verwijderen: Laag naar hoog

We proberen deze allemaal en gieten de resultaten in een histogram. We geven het aantal gevonden. Op de verticale as ziet u het totaal aantal dominantietoppen over alle testbestanden (die specifiek gemaakt zijn voor het dominantiealgoritme te testen).

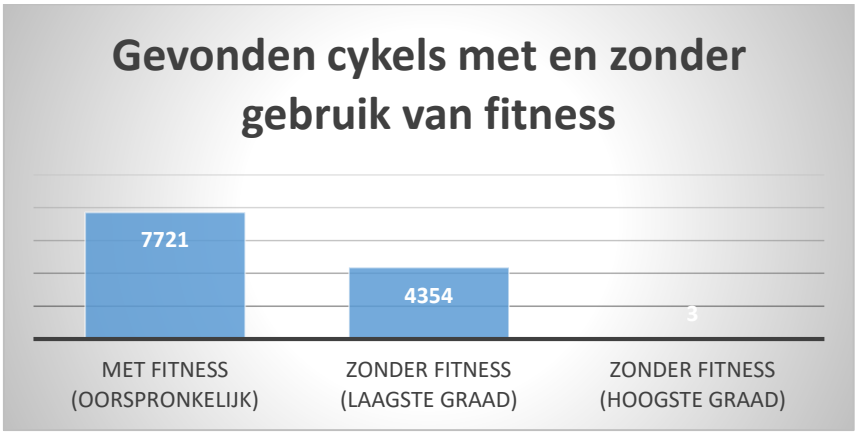


Optie 1 presteert beter als ons oorspronkelijk algoritme.

Hamiltoniaanse cykels in vlakke triangulaties

Zonder Fitness:

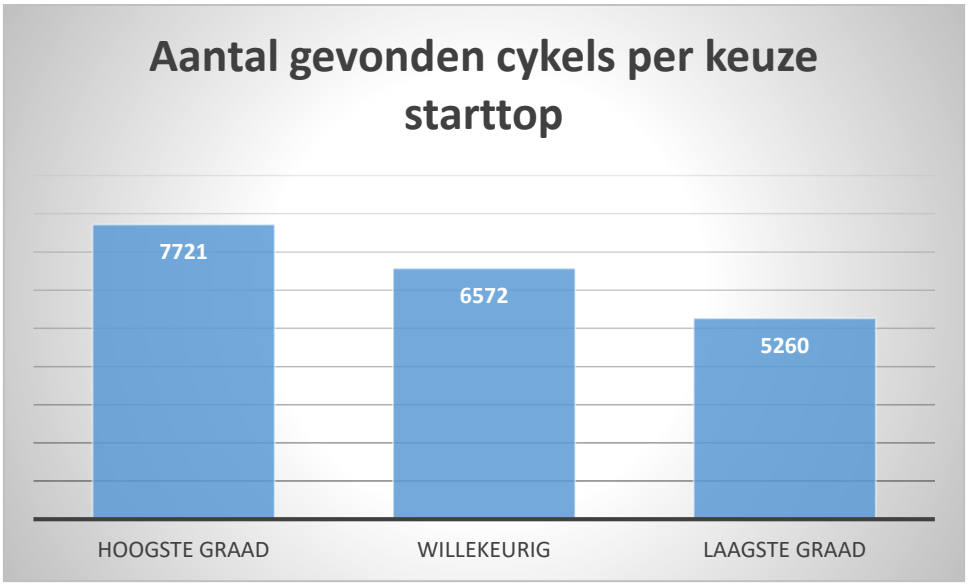
Ik test mijn algoritme in dit deel zonder rekening te houden met de fitnesses van de toppen. Voor de keuze van de volgende top, gebruiken we de graad als criterium. We gebruiken twee strategieën: de eerste maal kiezen we steeds de buur met de laagste graad en de tweede maal diegene met de hoogste graad.



Fitness en hoogste graad is duidelijk vreselijk slecht. Fitness en laagste graad vindt ongeveer de helft minder dan het origineel. Het gebruik van het concept “fitness” is dus geen slecht idee.

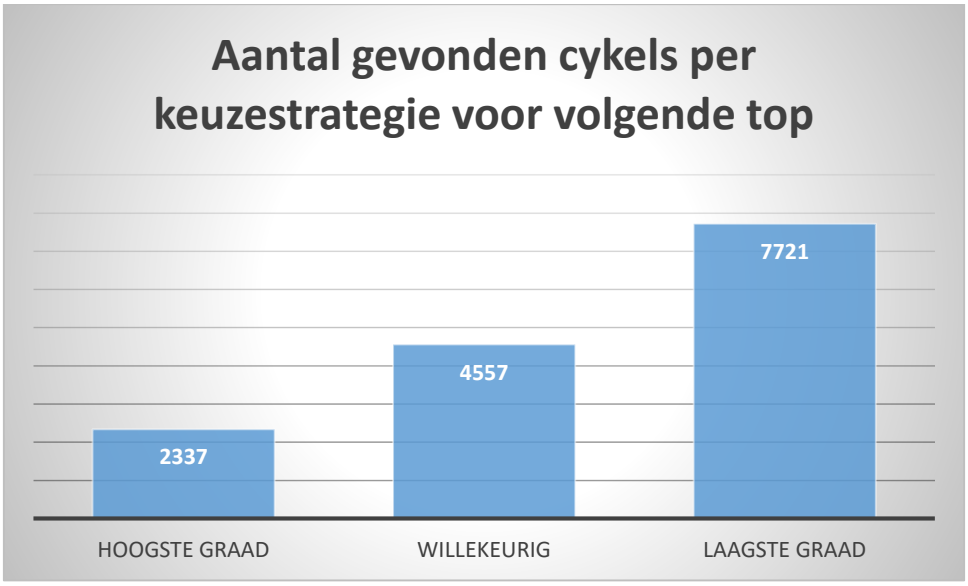
Keuze starttop:
Het huidige algoritme start bij die top met de hoogste graad. We voeren een experiment uit op alle testset grafen, maar starten hierbij met de top met de laagste graad en eens met een willekeurige

starttop. We stellen een histogram op die het aantal gevonden cykels weergeeft per methode. In totaal zijn er 9149 grafen getest. In de zip vindt u tevens 3 databestanden met naam, StarttopX.txt, gebruikt om deze histogram op te stellen. Voor de random strategie werd de beste uit 3 uitvoeringen geselecteerd.



De manier waarop we het oorspronkelijk deden, namelijk met die top met de hoogste graad starten, blijkt het best te presteren.

<u>Keuze volgende top bij toppen met gelijke fitness:</u>
Wat ik ook anders kan doen, is de keuze van de volgende top op het pad. Momenteel kiezen we die top met de laagste graad indien er meerdere zijn met **dezelfde fitness**. We zullen analoog aan het kiezen van de starttop hier een experiment uitvoeren waarbij we degene kiezen met de hoogste graad, laagste graad en een willekeurige. We kiezen als starttop steeds degene met de hoogste graad aangezien deze het best lijkt te presenteren (zie hierboven).
We verwachten dat de keuze van de kleinste graad het beste zal presenteren, want daar is de kans op het kiezen van een slechte volgende top (als die er is) kleiner aangezien er minder buren zijn.
De resultaten ziet u in volgend histogram.



De resultaten bevestigen onze verwachtingen: de laagste graad strategie presteert het beste.

Referenties

[1] Meer informatie over de stelling van Grinberg: www.en.wikipedia.org/wiki/Grinberg's_theorem

Het bewijs maakt gebruik van Eulers formule en vindt u onderaan in het volgend document: http://www.sfu.ca/~mdevos/notes/graph/445_hamiltonian.pdf

[2] "Polynomial-Time Data Reduction for Dominating Set∗" meegeleverd in de zip.