Universiteit Utrecht

Faculteit Bètawetenschappen

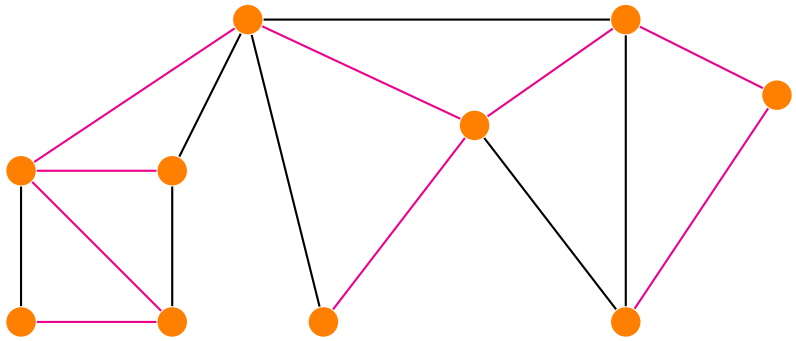
Independency Trees op verschillende graafklassen

Bachelorscriptie

Wiskunde

Door Morris Breed

Begeleider
Dr. Erik Jan van Leeuwen



18 januari 2019

Inhoudsopgave

1 Inleiding **4**

2 Independency trees en gerelateerde problemen **5**
2.1 Independency trees 5
2.2 Geparametriseerde complexiteit van Max/Min Leaf IT 8
2.3 Algoritme voor specifieke Max/Min Leaf Tree 12

3 Co-bipartiete graaf **13**

4 Splitgraaf **16**

5 Intervalgraaf **18**
5.1 Bepalen van de optimale bladverzameling 18
5.2 Bladverzameling uitbreiden naar Max Leaf IT 20
5.3 Complexiteitsanalyse 23

6 Chordale graaf **25**
6.1 Perfect elimination ordering 25
6.2 Bepalen van de optimale bladverzameling 26
6.3 Bladverzameling uitbreiden naar Max Leaf IT 27
6.4 Complexiteitsanalyse 29

7 Conclusie **30**

8 Bibliografie **31**

1 Inleiding

Het vinden van een opspannende boom met het maximale aantal bladeren in samenhangende grafen is een welbekend probleem, waar menig onderzoek aan is gewijd, bijvoorbeeld [1], [2] en [3]. Over het verwante probleem Max Leaf Independency Tree, waarin als extra eis wordt gesteld dat de bladeren een onafhankelijke verzameling knopen vormen, is minder bekend. In dit stuk zal dieper op dit probleem worden ingegaan. Eerder is aangetoond [4] dat het probleem op algemene grafen NP-moeilijk is. De complexiteit op de meeste graafklassen is echter nog onbekend. Het doel van dit onderzoek is om voor meerdere graafklassen de complexiteit van dit probleem te bepalen.

De scriptie is zodanig opgebouwd, dat het onderzoek chronologisch wordt gevolgd. Eerst zullen we de nodige definities en achtergronden behandelen, waarna we algoritmen voor verschillende graafklassen ontwerpen. De eenvoudigere graafklassen komen daarbij als eerste aan bod, want deze hebben we het eerst onderzocht. Vervolgens zullen we telkens een stap verder gaan, totdat we bij de klasse van chordale grafen aankomen. Deze opbouw zorgt ervoor dat gemaakte keuzes soms later niet de meest efficiënte blijken. Echter, de lezer wordt zo meegenomen in het proces, wat leidt tot meer begrip over hoe de uiteindelijke algoritmen tot stand zijn gekomen.

De co-bipartiete graaf is de eerste graafklasse die aan bod zal komen. We raken daardoor beter gewend aan de vraagstelling en krijgen een goed idee hoe algoritmen die een Max Leaf IT vinden eruit zien. Vervolgens behandelen we de splitgraaf en de intervalgraaf, twee graafklassen die beide behoren tot de klasse van chordale grafen, die we ten slotte bekijken. Co-bipartiete grafen zijn niet noodzakelijk chordaal, dus is deze graafklasse niet vergelijkbaar met één van de andere bekeken graafklassen.

In het eerste hoofdstuk zullen we twee artikelen over eerdere onderzoeken betreffende independency trees behandelen. Vervolgens gaan we in de hoofdstukken die daarop volgen op zoek naar algoritmen met een polynomiale looptijd voor het Max Leaf Independency Tree-probleem voor co-bipartiete grafen, splitgrafen, intervalgrafen en chordale grafen. De algoritmen voor de eerste twee graafklassen zijn resultaat van ons onderzoek, waarbij geen gebruik is gemaakt van eerdere werken. Voor de intervalgrafen en chordale grafen hebben we algoritmen [5] en [6] voor Max Independent Set bestudeerd en deze ideeën toegepast om zo tot eigen algoritmen te komen.

2 Independency trees en gerelateerde problemen

Een opspannende boom T in een samenhangende graaf $G = (V, E)$ noemen we een *independency tree* als de bladeren van T paarsgewijs niet-verbonden zijn. Dit wil zeggen dat voor alle bladeren b_1 en b_2 van T geldt dat $(b_1, b_2) \notin E$. Wij zullen het zogenaamde Max Leaf Independency Tree-probleem behandelen. We gaan daarin op zoek naar independency trees in grafen met het maximale aantal bladeren.

2.1 Independency trees

Independency trees werden door Böhme et al. [7] geïntroduceerd. Independency trees kunnen onder meer gebruikt worden om voor een samenhangende graaf te bepalen of het een *Hamiltoncykel* bevat. Een Hamiltoncykel is een gesloten pad dat iedere knoop van de graaf bevat. Als een dergelijk pad niet gesloten is, oftewel het is geen cykel, noemen we het een *Hamiltonpad*. Om het verband tussen het bestaan van independency trees en van Hamiltoncykels in een samenhangende graaf te begrijpen, behandelen we het onderstaande Algoritme 1, waarvan de looptijd polynomiaal begrensd is. Daarin wordt het begrip *cutedge* gebruikt, wat een kant $e \in E$ is, zodat $G = (V, E - \{e\})$ niet samenhangend is.

Zij $G = (V, E)$ een samenhangende graaf over ten minste drie knopen en T een willekeurige opspannende boom van G.

Stap 1: Als twee bladeren van T een kant $e \in E$ gemeenschappelijk hebben, verbinden we deze met elkaar en verkrijgen we $T + \{e\}$, dat een unieke cykel C bevat.

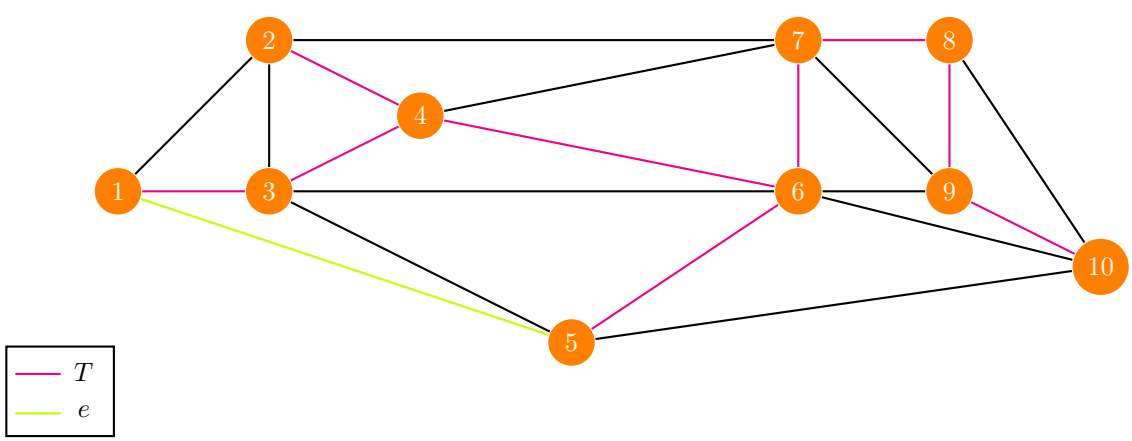
Stap 2: Ten minste één knoop $v \in C$ heeft dan een graad van minstens drie in $T + \{e\}$, mits C geen Hamiltoncykel is. We verwijderen één kant die verbonden is met v en geen cutedge is van $T + \{e\}$. De nieuwe boom noemen we T'.

Stap 3: Output opspannende boom T' met minstens één blad minder dan T.

Stap 4: Herhaal stap 1 tot en met stap 3, totdat dit niet meer mogelijk is.

Algoritme 1: Van opspannende boom tot independency tree of Hamiltonpad (Böhme et al.)

Ter illustratie geven we een voorbeeld van het toepassen van het algoritme:



Figuur 1

We zien in Figuur 1 dat knopen 1 en 5 aangrenzende bladeren zijn van opspannende boom T. Merk op dat T vier bladeren heeft, namelijk knopen 1, 2, 5 en 10. Het algoritme voegt kant $e = (1, 5)$ toe aan de nieuwe opspannende boom T'. Nu verkrijgen we de cykel $1 - 5 - 6 - 4 - 3 - 1$, waarin knoop 4 graad 3 heeft. Nu wordt kant $(3, 4)$ (kant $(4, 6)$ is ook een mogelijkheid) verwijderd, wat ons een nieuwe opspannende boom T' oplevert met drie bladeren: knopen 2, 3 en 10.

Stap 1 tot en met stap 3 van dit algoritme kunnen worden toegepast, zolang er twee aangrenzende bladeren zijn en zolang boom T geen Hamiltonpad is van G met aangrenzende uiteinden. Met andere woorden, uiteindelijk vinden we een Hamiltonpad, dat bevat is in een Hamiltoncykel, of een independency tree in G.

Door voor ieder blad van de opspannende boom ieder ander blad langs te gaan, kan men een eventuele kant tussen twee bladeren vinden. Als we n noteren voor het aantal knopen in de graaf en m voor het aantal kanten, wat we gedurende deze gehele scriptie zullen doen, kan stap 1 dus in $O(n^2)$ tijd worden uitgevoerd. We bewaren daarna gedurende het algoritme $G[B]$, de bladeren van de boom en eventuele kanten tussen deze bladeren.

Vervolgens duurt het lineaire tijd ($O(n)$) om de knoop in de cykel te bepalen met minstens graad 3 en het nagaan of een kant een cutedge is van $T + \{e\}$, kan worden gedaan door middel van breadth-first-search. We weten dat op $T + \{e\}$ alle n knopen en precies n kanten bevat, dus is de looptijd van deze BFS $O(n)$. Daarna passen we $G[B]$ aan: de knopen die incident zijn met e zijn geen bladeren meer en worden dus samen met e verwijderd, terwijl er eventueel een blad is bijgekomen, die met zijn kanten wordt toegevoegd aan $G[B]$. Zolang $G[B]$ kanten bevat, kunnen we deze stappen herhalen. Het aantal bladeren neemt per stap af, dus kunnen we maximaal n stappen uitvoeren. De totale looptijd van het algoritme wordt dus $O(n^2)$.

Het belangrijkste resultaat van [7] is het onderstaande theorema. Hierin worden met C_n en K_n respectievelijk de cykelgraaf en de complete graaf van n knopen bedoeld. De complete bipartiete graaf met bipartitie $\{U, V\}$, waarbij zowel U als V uit $\frac{n}{2}$ knopen bestaat, wordt aangeduid met $K_{n/2,n/2}$. Meer over deze graaf volgt in Hoofdstuk 3.

Theorema van Böhme. Zij G een samenhangende graaf met $n \geq 3$ knopen. Dan zijn de volgende beweringen equivalent:

1. G is isomorf met C_n, K_n of $K_{n/2,n/2}$ (voor een even n).
2. G bevat geen independency tree.
3. G bevat geen depth-first-search tree waarin alle bladeren paarsgewijs niet-aangrenzend zijn.
4. G bevat een Hamiltonpad en ieder Hamiltonpad is bevat in een Hamiltoncykel.

Als G isomorf is met C_n of K_n, kunnen we vanzelfsprekend niet twee niet-aangrenzende knopen als bladeren van een opspannende boom kiezen, dus bevat G geen independency tree. In Hoofdstuk 3 zullen we het geval behandelen dat G isomorf is met $K_{n/2,n/2}$. Ook de implicatie (2) $\implies$ (3) is eenvoudig in te zien. Als G namelijk wel een depth-first-search tree met niet-aangrenzende bladeren bevat, dan is dit ook een independency tree. Door middel van een contrapositief bewijs wordt implicatie (3) $\implies$ (4) duidelijk: als bewering (4) onwaar is, dan bevat G geen Hamiltonpad of G bevat een Hamiltonpad dat niet bevat is in een Hamiltoncykel. Vanzelfsprekend is de graaf dan niet compleet, dus bestaat er een pad $P = x_1, \ldots, x_k$ in G zodat x_1 en x_k geen buren zijn. We bekijken het langste pad met deze eigenschap. Dan geldt er dus dat alle buren van x_1 en x_k bevat zijn in P. Als we vervolgens een depth-first-search tree maken vanaf knoop x_1, waarbij we voorrang geven aan de knopen in P, dan is pad P bevat in de DFS tree. We merken op dat de bladeren van een DFS tree, afgezien van de beginknoop, altijd een onafhankelijke verzameling knopen vormen. Alle buren van x_1 zitten in P en $(x_1, x_k) \notin E$, dus de hele bladverzameling van de DFS tree is paarsgewijs niet-aangrenzend, dus de DFS tree is een independency tree.

Het bewijs van implicatie (4) $\implies$ (1) is complexer. Er wordt gebruik gemaakt van gevalsonderscheiding: G bevat wel of geen 3-koorde. Als C een Hamiltoncykel is, dan is een *3-koorde* een kant e zodat de kortste cykel die gevormd kan worden met e en de rest van de knopen van C lengte 3 heeft. Als bewering (4) geldt en G bevat een 3-koorde, dan volgt daaruit dat G een complete graaf is, dus isomorf is met K_n. Als G geen 3-koorde bevat, dan is G de complete bipartiete graaf $K_{n/2,n/2}$. Aan het begin van Paragraaf 2.2 zullen we details van dit bewijs bekijken.

Naast de resultaten uit dit theorema is het duidelijk dat een graaf samenhangend moet zijn, wil het een opspannende boom bevatten. We zien dus dat graaf G een independency tree bevat precies dan als G een samenhangende graaf is met minstens drie knopen die geen cykelgraaf, complete graaf of complete bipartiete graaf is. In het vervolg zullen we enkel grafen behandelen die aan deze voorwaarden voldoen.

We zagen dat de complete graaf K_n, de cykelgraaf C_n en de complete bipartiete graaf $K_{\frac{n}{2},\frac{n}{2}}$ geen independency tree bevatten. Daarnaast is een logische vervolgvraag in welke grafen iedere opspannende boom een independency tree is. Eppstein en Le [8] laten zien dat een dergelijke graaf getypeerd kan worden als een graaf die geen buren v en w bevat, die graad minstens 2 hebben en waarvoor de graaf samenhangend blijft als v en w worden verwijderd. De eerste implicatie kan als volgt worden ingezien: stel dat iedere opspannende boom in $G = (V, E)$ een independency tree is en er bestaan wel buren v, $w \in V$ met $deg(v) > 1$ en $deg(w) > 1$ zodat $G' = (V - \{v, w\}, E|_{V-\{v,w\}})$ samenhangend is. Dan kan aan een opspannende boom van G' een kant naar v en een kant naar w worden toegevoegd, want beide knopen hebben een graad van minstens twee. Deze opspannende boom van G is dan echter geen independency tree, want v en w zijn buren, dus stuiten we op een tegenspraak.

Voor de omgekeerde implicatie nemen we aan dat er geen v, $w \in V$ met de bovenstaande eigenschappen bestaan. Als er nu een opspannende boom T van G bestaat die geen independency tree is, zijn er bladeren van T die buren van elkaar zijn. Deze bladeren hebben dus een graad van minstens 2. Echter, als we bladeren van een opspannende boom verwijderen, blijft het restant van de graaf samenhangend, dus deze bladeren hebben wel de eerdergenoemde eigenschappen en we hebben wederom een tegenspraak.

We sluiten dit hoofdstuk af met het behandelen van de publicatie van Casel et al. [4]. In dit artikel wordt onderzoek gedaan naar de looptijd van algoritmen die de independency tree bepalen met het maximale of het minimale aantal bladeren in een graaf. Naast independency trees worden ook de verwante *cliquy trees* behandeld. Een cliquy tree van een graaf is een opspannende boom, waarin de bladeren een *kliek* vormen. Een kliek is een complete deelgraaf. Wij zullen het deel over cliquy trees echter buiten beschouwing laten, omdat dit niet direct relevant is voor ons onderzoek.

In het artikel spelen *geparametriseerde complexiteit* en *kernelisatie* een rol. Voor de problemen de behandeld worden, wordt de geparametriseerde complexiteit bepaald, wat inhoudt dat de complexiteit wordt uitgedrukt in één of meerdere parameters. In de publicatie worden het aantal bladeren en het aantal interne knopen (knopen van een boom die geen bladeren zijn) als parameters gebruikt. Kernelisatie is een techniek om efficiënte algoritmen te ontwerpen. Dit wordt bereikt door voordat het daadwerkelijke probleem wordt opgelost, eerst de input voor te bewerken en daarmee te vervangen door een kleinere input, die de *kernel* wordt genoemd. Vervolgens wordt het algoritme uitgevoerd op de kernel, wat het algoritme dus efficiënter maakt. Nu we de intuïtie van kernelisatie begrijpen, kunnen we de formele definitie formuleren [9]: een kernel van een geparametriseerd probleem Q met parameter k is een algoritme A dat een instantie (x, k) van Q in polynomiale tijd (in $|x|$ en k) omschrijft naar een equivalente instantie (x', k'), waarvoor geldt dat $|x'| + k' \leq g(k)$ voor een polynoom g van parameter k. We noemen instanties (x, k) en (x', k') equivalent als (x, k) een ja-instantie is precies dan als (x', k') dat is.

Voor de problemen Min Leaf Independency Tree (het vinden van een independency tree met het minimale aantal bladeren) en Max Leaf Independency Tree (het vinden van een independency tree met het maximale aantal bladeren) zijn geen algoritmen voor algemene grafen bekend die in polynomiale tijd uitvoerbaar zijn. Verder laat men zien dat er voor het minimaliseren van het aantal bladeren in een independency tree een $O(4^k)$-algoritme bestaat, waarbij k de parameter voor het aantal interne knopen is. Dit probleem heeft een kernel van $2k$ knopen. Daarnaast wordt laten zien dat het maximaliseren

van het aantal bladeren in een independency tree een complexiteit van $O(18^k)$ heeft met een kernel van $O(k \cdot 2^k)$ knopen, waarin k wederom de parameter voor het aantal interne knopen is.

Ten slotte wordt een algoritme behandeld dat voor iedere willekeurige graaf een opspannende boom bepaalt, die over een bepaalde gewenste eigenschap beschikt. Als deze eigenschap in $O(f(n))$ tijd kan worden bepaald, kan dit algoritme in $O(3^n \cdot f(n))$ een opspannende boom, die over deze eigenschap beschikt, met het maximale of minimale aantal bladeren bepalen.

2.2 Geparametriseerde complexiteit van Max/Min Leaf IT

Het bepalen of een graaf een independency tree bevat en het vinden ervan, kan worden gedaan in polynomiale tijd met behulp van Algoritme 1. Dit algoritme vindt namelijk een independency tree of een Hamiltoncykel. In het laatste geval kunnen we, zoals we zagen in het Theorema van Böhme, nagaan of de graaf isomorf is met C_n, K_n of $K_{n/2,n/2}$ om te bepalen of het een independency tree bevat. Als Algoritme 1 een Hamiltoncykel $C = x_1, \ldots, x_n$ vindt, terwijl $G = (V, E)$ wel een independency tree bevat, bevat G een independency tree met twee bladeren. Dit volgt uit het Theorema van Böhme, waarin men stelt dat grafen die een independency tree bevatten ofwel geen Hamiltonpaden bevatten ofwel een Hamiltonpad bevatten, dat geen onderdeel is van een Hamiltoncykel. Als Algoritme 1 dus een Hamiltoncykel vindt, is niet ieder Hamiltonpad bevat in een Hamiltoncykel en bestaat er dus een independency tree met twee bladeren. Ook bevat G minstens vier knopen als Algoritme 1 een Hamiltoncykel vindt, anders is het immers een complete graaf. De kanten die niet in C zitten, zijn *koorden*: kanten die knopen op de cykel met elkaar verbinden. Omdat G geen cykelgraaf is, bevat het minstens één koorde. Nu gebruiken we dezelfde strategie als bij het bewijs van implicatie (4) $\implies$ (1), waar we nu in meer detail over zullen treden. We maken gebruik van gevalsonderscheiding: C bevat wel een 3-koorde of C bevat geen 3-koorde.

Stel dat $C = x_1, \ldots, x_n$ een 3-koorde, zeg $(x_1, x_3) \in E$ bevat. Dan is $x_2, x_1, x_3, x_4, \ldots, x_n$ een Hamiltonpad van G. Als $(x_2, x_n) \notin E$, hebben we onze independency tree gevonden. Als $(x_2, x_n) \in E$, bevat G dus wederom een 3-koorde, namelijk (x_2, x_n), en kunnen we bovenstaande procedure nogmaals uitvoeren. Na het herhaaldelijk toepassen van deze strategie, vindt men uiteindelijk ofwel een Hamiltonpad met niet-aangrenzende uiteinden ofwel dat graaf G alle 3-koorden (x_i, x_{i+2}) $\forall 1 \leq i \leq n-2$ bevat. Als we geen Hamiltonpad met niet-aangrenzende uiteinden vinden, bevat G dus ook bijvoorbeeld $(x_2, x_n) \in E$, waardoor het ook Hamiltonpaden $x_4, x_5, \ldots, x_n, x_2, x_3, x_1$ en $x_5, \ldots, x_n, x_2, x_3, x_4, x_1$ bevat. Omdat we aannemen dat we geen independency tree vinden, geldt dan ook dat $(x_1, x_4), (x_1, x_5) \in E$. Als we dit blijven herhalen, komen we erachter dat $(x_1, y) \in E$ $\forall y \in V - \{x_1\}$. Dit argument is niet alleen van toepassing op x_1, maar van toepassing op iedere knoop, dus als we op deze manier geen independency tree met twee bladeren vinden, is de graaf compleet.

We bekijken nu het geval dat $C = x_1, \ldots, x_n$ geen 3-koorde bevat. We zagen eerder, aangezien G geen cykelgraaf is, dat C minstens één koorde, zeg $(x_1, x_i) \in E$, bevat. We weten dan dat G minstens zes knopen bevat, anders is (x_1, x_i) een 3-koorde. Verder bevat G dan Hamiltonpaden $x_2, x_3, \ldots, x_i, x_1, x_n, x_{n-1}, \ldots, x_{i+1}$ en $x_n, x_{n-1}, \ldots, x_i, x_1, x_2, \ldots, x_{i-1}$. Als nu $(x_2, x_{i+1}) \notin E$ of $(x_n, x_{i-1}) \notin E$, hebben we een independency tree gevonden. Als zowel (x_2, x_{i+1}) als (x_n, x_{i-1}) wel bestaat, bevat G ook Hamiltonpad $x_{i-2}, x_{i-3}, \ldots, x_2, x_{i+1}, x_{i+2}, \ldots, x_n, x_{i-1}, x_i, x_1$. Wederom vinden we dus een independency tree als er geldt dat $(x_1, x_{i-2}) \notin E$. Als we na herhaling van deze stappen geen independency tree vinden, kunnen we concluderen, aangezien C geen 3-koorden bevat, dat i even is. Aangezien $(x_1, x_n) \in E$, geldt dan ook dat n even is. Sterker nog, we zien dat dan geldt dat $(x_1, x_j) \in E$ $\forall j \in \{2, 4, \ldots, n\}$. Opnieuw kunnen we dit argument formuleren voor alle knopen, niet alleen voor x_1, waardoor we zien dat alle knopen met oneven indices enkel verbonden zijn met knopen met even indices en andersom. De graaf is dan dus de complete bipartiete graaf $K_{n/2,n/2}$.

Het probleem Min Leaf IT is daarentegen een NP-moeilijk probleem door reductie van Hamiltonpad [7], waarvan bekend is dat het NP-moeilijk is. We bekijken hiervoor de variant Min Leaf IT (leaf), waarbij we parameter l gebruiken voor het maximale aantal bladeren en deze de waarde 3 toekennen. De reductie ziet er dan als volgt uit: laat G een graaf zijn met een knoop v. Vervolgens voegen we knopen v_1, v_2 toe aan G, die enkel kanten hebben naar v en noemen we de nieuw verkregen graaf G'. De toegevoegde knopen hebben graad 1, dus zijn logischerwijs bladeren van iedere independency tree

van G'. Nu geldt dat G' een independency tree met maximaal 3 bladeren heeft precies dan als G een Hamiltonpad bevat, die begint bij knoop v. We zullen later zien dat ook MAX LEAF IT een NP-moeilijk probleem is.

In het vervolg van het artikel wordt gekeken naar de geparametriseerde complexiteit van MIN LEAF IT en MAX LEAF IT voor parameters l = aantal bladeren en k = aantal interne knopen. Daarnaast zullen de problemen worden vergeleken met op het eerste oog verwante problemen zoals CONNECTED VERTEX COVER (het vinden van de kleinste vertex cover, die een samenhangende deelgraaf induceert), MAX LEAF SPANNING TREE (het vinden van een opspannende boom met het maximale aantal bladeren) en MAX-INTERNAL SPANNING TREE (het vinden van een opspannende boom met het maximale aantal interne knopen).

Een *dominating set* van een graaf $G = (V, E)$ is een deelverzameling knopen $D \subset V$, zodanig dat iedere knoop in $V - D$ verbonden is met een knoop in D. Een *vertex cover* is een deelverzameling $C \subset V$ zodat iedere kant minstens één eindpunt in C heeft. Als de deelverzameling D een samenhangende deelgraaf induceert, oftewel $G[D] = (D, E|_D)$ is samenhangend, wordt deze ook 'connected' genoemd. De vergelijkbare definitie geldt voor een connected vertex cover. De interne knopen van een opspannende boom vormen een connected dominating set, terwijl de verzameling interne knopen van een independency tree een connected vertex cover is.

Er geldt dat voor n het aantal knopen in G, d de grootte van de kleinste connected dominating set en l het aantal bladeren van de opspannende boom met het maximale aantal bladeren, dat:

$$n = d + l$$

Daaruit volgt dat de problemen CONNECTED DOMINATING SET en MAX LEAF SPANNING TREE equivalent zijn.

Als we dit willen uitbreiden naar het vinden van een Max Leaf IT, dan voegen we de eis toe dat de verzameling bladeren onafhankelijk moet zijn, oftewel we vervangen de kleinste connected dominating set door de kleinste connected vertex cover. Door middel van een tegenvoorbeeld laat men echter zien, dat de volgende vergelijking, waarin c staat voor de kleinste connected vertex cover niet geldt.

$$n = c + l$$

Hierdoor biedt het probleem CONNECTED VERTEX COVER weinig steun bij het zoeken naar een Max Leaf IT.

We kunnen de problemen anders bekijken door gebruik te maken van de parameter l, die staat voor het aantal bladeren. Het probleem MAX LEAF IT (leaf), met parameter l, houdt in dat we bepalen of de graaf een independency tree met minstens l bladeren bevat. Voor MIN LEAF IT (leaf) onderzoeken we of er een independency tree met maximaal l bladeren bestaat.

De eerder beschreven reductie van HAMILTONPAD naar MIN LEAF IT liet zien dat MIN LEAF IT een NP-moeilijk probleem is. Voor geparametriseerde vorm MIN LEAF IT (leaf) werkt exact dezelfde reductie. Een probleem is namelijk *Para NP-Moeilijk*, als het voor een constante parameter al NP-moeilijk is. In de reductie zagen we dat MIN LEAF IT NP-moeilijk is voor $l = 3$, waaruit we concluderen dat MIN LEAF IT (leaf) Para NP-moeilijk is.

In het artikel wordt een reductie van INDEPENDENT SET (het vinden van een onafhankelijke verzameling knopen van grootte minstens l) naar MAX LEAF IT (leaf) beschreven: als we aan een graaf G een grote kliek met kanten naar iedere knoop in $G = (V, E)$ toevoegen en de nieuwe graaf G' noemen, is het eenvoudig om een opspannende boom te bepalen met een specifieke verzameling bladeren $I \subset V$. We kunnen namelijk een Hamiltonpad vinden die begint en eindigt in de toegevoegde kliek en daartussen iedere knoop in $V - I$ langsgaat. Vervolgens verbinden we de eindpunten van het Hamiltonpad met de knopen in I en hebben we een independency tree van G'.

Deze geparametriseerde versie van het probleem INDEPENDENT SET is $W[1]$-moeilijk, wat kan worden gezien als de equivalent van NP-moeilijk voor geparametriseerde problemen, dus MAX LEAF IT (leaf) is ook $W[1]$-moeilijk. Hieruit volgt dat MAX LEAF IT NP-moeilijk is, want dezelfde reductie kan

worden gebruikt van Maximum Independent Set, waardoor $I \subset V$ de grootst mogelijke verzameling onafhankelijke knopen wordt. Zo vinden we de Max Leaf IT in G' precies dan als we in G de Maximum Independent Set vinden.

Het gebruik van de parameter l levert ons dus niet veel op; de problemen blijven moeilijk op te lossen.

Een andere parameter die gebruikt kan worden is k, voor het aantal interne knopen. De bijbehorende problemen Max Leaf IT (internal) en Min Leaf IT (internal) bepalen of de graaf een independency tree met respectievelijk maximaal k en minimaal k interne knopen bevat. We zullen zien dat Min Leaf IT (internal) behoort tot de klasse *Fixed Parameter Tractable* (FPT). Problemen in deze klasse hebben een complexiteit van $O(f(k) \cdot |x|^{O(1)})$ voor een zekere parameter k, functie f en waarbij x de invoer en $|x|$ dus de invoergrootte is. Een cruciaal gegeven van de complexiteit van problemen in FPT is dat het geen termen bevat als n^k. Net zoals $W[1]$ kan worden gezien als de geparametriseerde equivalent van NP-moeilijkheid, is FPT vergelijkbaar met de klasse P, die problemen bevat die in polynomiale tijd oplosbaar zijn.

Een probleem behoort tot FPT voor parameter k precies dan als het *beslisbaar* is en een kernel heeft [10]. Een probleem wordt beslisbaar genoemd als er een algoritme bestaat waarmee het opgelost kan worden. Stel dat een beslisbaar probleem een kernel heeft. Dan kan een instantie (x, k) dus in polynomiale tijd, oftewel $O(x^{O(1)})$, worden omgeschreven naar een equivalente instantie (x', k'), waarbij $x' + k' \leq g(k)$ voor een zekere functie g. Daarnaast is het probleem beslisbaar, dus bestaat er een $O(f(x))$-algoritme wat de correcte oplossing geeft. Als we nu de kernel uitvoeren en vervolgens dit algoritme gebruiken om het probleem op te lossen, verkrijgen we een $O(f(g(k)) + x^{O(1)})$-algoritme, dus behoort het probleem tot FPT. Als we nu aannemen dat een probleem tot FPT behoort, weten we dat het beslisbaar is in $O(f(k) \cdot x^{O(1)})$ tijd. We willen dus alleen nog laten zien dat het een kernel bevat. Hiervoor onderscheiden we twee gevallen: $f(k) < x$ en $x \leq f(k)$. In het eerste geval heeft het $O(f(k) \cdot x^{O(1)})$-algoritme een looptijd van $O(x^{1+O(1)})$, oftewel het probleem is beslisbaar in polynomiale tijd. Dan nemen we als kernel de uitvoer en zijn we klaar. In het tweede geval is de invoer al kleiner dan een functie van parameter k, dus hebben we ook een kernel in polynomiale tijd.

In het artikel laat men zien dat Max Leaf IT (internal) een kernel heeft van maximaal $2^k(k+1) + k^2 + 2k$ knopen, door gebruik te maken van het feit dat een knoop van graad groter dan k geen blad kan zijn van een independency tree met maximaal k interne knopen. Als een dergelijke knoop namelijk wel een blad is, is deze in G verbonden aan minstens één ander blad en is de boom geen independency tree. Ten eerste laten we V' de verzameling knopen zijn met een graad die groter is dan k. Deze knopen kunnen geen bladeren zijn, dus worden ze interne knopen in een independency tree. Als deze verzameling meer dan k knopen bevat, kunnen we daarom direct concluderen dat er geen independency tree met maximaal k interne knopen bestaat. Daarnaast noemen we R de verzameling knopen die minstens één buur buiten V' hebben en $L = V \setminus (V' \cup R)$ de overige knopen. We merken op dat knopen in L alleen buren in V' hebben, want anders zaten ze in R. Daaruit volgt ook dat L een onafhankelijke verzameling knopen is.

Voor een willekeurige $S \subset V'$ definiëren we V_S als volgt: $\{v \in L \mid N(v) = S\}$. Men laat zien dat als $|V_S| > k + 1$, we dan $|V_S| - k - 1$ knopen kunnen verwijderen, zodat we een equivalente instantie van het probleem overhouden. Als namelijk $C \subset V_S$ met $|C| = |V_S| - k - 1$ en we bekijken een independency tree in $G[V \setminus C]$ met maximaal k interne knopen, dan zien we dat de knopen uit V' ook in $G[V \setminus C]$ een graad groter dan k hebben, dus ook interne knopen van een independency tree zijn. Het idee is dus om te bepalen of er een independency tree in $G[V \setminus C]$ bestaat met maximaal k interne knopen, omdat we vervolgens de knopen in C kunnen toevoegen door middel van kanten naar knopen in V'. De knopen in C hebben alleen buren in V', dus het aantal interne knopen verandert daar niet door en de verzameling bladeren blijft onafhankelijk.

Zij T een independency tree van G met interne knopen I, waarvoor $|I| \leq k$. Voor een knoop $v \in V_S$ bestaan er twee mogelijkheden: v bevindt zich in I of v is een blad van T met een kant naar een knoop in V'. Er geldt dus dat $|V_S \setminus I|$ knopen uit V_S voldoen aan de tweede mogelijkheid. Deze knopen kunnen we verwijderen, wat ons een equivalente instantie geeft, mits de knoop in V' waar het blad uit $V_S \setminus I$ mee verbonden is geen blad wordt. Per V_S kunnen we dus minstens $|V_S \setminus I| - |V'|$ knopen verwijderen. De boom bevat maximaal k interne knopen, dus er geldt dat $V' \subset I$, dus kunnen we minimaal $|V_S| - k$ knopen verwijderen per V_S.

Voor iedere V_S houden we dan $|V_S| - (|V_S| - k - 1) = k + 1$ knopen over. Voor ieder element in V' zijn er twee mogelijkheden: of het element zit in S of het element zit niet in S. De verzameling V' bestaat uit maximaal k elementen, dus bestaan er maximaal 2^k verschillende deelverzamelingen $S \subset V'$, dus bevat L na de reductie maximaal $2^k \cdot (k + 1)$ knopen.

Nu gaan we bekijken hoeveel knopen R bevat. We merken op dat het samenstellen van een bedekking van de kanten in $G[R]$ een verzameling knopen van minimale grootte $\frac{|R|}{k+1}$ vereist, omdat de graad van knopen in R maximaal k is. Als R meer dan $k(k + 1)$ knopen bevat, dan zijn voor een bedekking van de kanten van $G[R]$ minstens k knopen nodig. Dan bestaat er dus geen vertex cover van G met maximaal k knopen en dus geen independency tree met maximaal k interne knopen. Er geldt dus dat $|R| \leq k(k + 1)$. Daarnaast bevat V' maximaal k knopen, dus bevat de uiteindelijke kernel maximaal $2^k \cdot (k + 1) + k(k + 1) + k = 2^k(k + 1) + k^2 + 2k$ knopen.

Nu we een exponentiële kernel hebben vastgesteld voor Max Leaf IT (internal), is een mogelijke vervolgvraag of er ook een kernel van polynomiale grootte bestaat voor dit probleem. Men laat echter zien dat dit niet het geval is, door middel van reductie van het bekende probleem Red-Blue Dominating Set, dat ook geen dergelijke kernel bevat.

Een brute-force algoritme loslaten op de kernel van $2^k(k + 1) + k^2 + 2k$ knopen voor Max Leaf IT (internal) is vanwege de exponentiële omvang van de kernel inefficiënt. In het artikel laat men zien dat er een slimmer algoritme bestaat, namelijk één die een looptijd van $O(18^k)$ heeft. We zagen eerder dat de interne knopen van een independency tree een vertex cover vormen. Als G een independency tree T met maximaal k interne knopen bevat, bestaat er dus een minimale vertex cover C van grootte maximaal k, die de interne knopen van T bevat. Verder wordt aangetoond dat vervolgens in $O(9^k)$ tijd kan worden bepaald of G een independency tree T bevat met maximaal k interne knopen, zodanig dat alle knopen in C interne knopen van T zijn. Dit algoritme is een aanpassing van het bekende Dreyfus-Wagner algoritme voor het Steiner Tree-probleem, waarin voor een deelverzameling knopen $S \subset V$ wordt gezocht naar een boom die alle knopen in S bevat. Het algoritme en het bewijs zijn te vinden in het artikel. Er zijn maximaal 2^k minimale vertex covers met maximaal k knopen. Als voor iedere vertex cover het $O(9^k)$-algoritme wordt toegepast, levert dit een algoritme voor Max Leaf IT (internal) op met een looptijd van $O(18^k)$.

Om een FPT-algoritme voor Min Leaf IT (internal) te ontwerpen, vergelijken we het met het probleem Max-Internal Spanning Tree, waarin wordt bepaald of een graaf een opspannende boom met maximaal k bladeren bevat. Het probleem Max-Internal Spanning Tree bevindt zich in FPT, bevat een kernel van hoogstens $2k$ knopen en de complexiteit is reeds bekend: $O(4^k)$ [11]. Om deze vergelijking te maken laten we T de opspannende boom van G zijn met het minimale aantal bladeren, ofwel het maximale aantal interne knopen. Als T een Hamiltonpad is, is T een independency tree met twee bladeren of is T bevat in een Hamiltoncykel. We bekijken enkel grafen die een independency tree bevatten, dus bevat G volgens het Theorema van Böhme in het laatste geval ook een Hamiltonpad dat niet bevat is in een Hamiltoncykel. De graaf bevat dus een independency tree met twee bladeren. Als T geen Hamiltonpad is, maar ten minste drie bladeren heeft en geen independency tree is, kunnen we Algoritme 1 toepassen. Dit levert een opspannende boom T' van G op met minstens één blad minder dan T. Echter, T was de opspannende boom van G zijn met het minimale aantal bladeren, dus levert dit een tegenspraak op. Boom T is dus een independency tree. Hieruit volgt dat als G een independency tree bevat, dat dan het minimale aantal bladeren van een opspannende boom van G gelijk is aan het minimale aantal bladeren van een independency tree van G.

Dit stelt ons in staat om de kernel en het algoritme van Max-Internal Spanning Tree te gebruiken voor Min Leaf IT (internal). Nadat we namelijk nagaan of G een independency tree bevat door te bekijken of G isomorf is met C_n, K_n of $K_{n/2,n/2}$, is het bepalen van het minimale aantal bladeren van een opspannende boom equivalent met het bepalen van het minimale aantal bladeren van een independency tree, zoals we hiervoor zagen. Om de daadwerkelijke independency tree te vinden met het minimale aantal bladeren, bepalen we eerst de opspannende boom T met het minimale aantal bladeren. Als T geen independency tree is, gebruiken we Algoritme 1, dat in polynomiale tijd loopt, om deze om te schrijven naar een independency tree met zelfs meer interne knopen of naar een Hamiltonpad, bevat in

een Hamiltoncykel. Graaf G is niet isomorf is met C_n, K_n of $K_{n/2,n/2}$, dus bestaat er, als Algoritme 1 op een Hamiltonpad stuit, zoals we aan het begin van deze paragraaf zagen, een Hamiltonpad in G, dat niet bevat is in een cykel. Een dergelijk Hamiltonpad is dus een Hamiltonpad waarvan de uiteinden niet-aangrenzend zijn. De graaf G bevat dan dus een independency tree met twee bladeren, het minimale aantal. Zodoende behoort Min Leaf IT (internal) net als Max-Internal Spanning Tree tot FPT, heeft het een kernel van maximaal $2k$ knopen en een algoritme met looptijd $O(4^k)$.

2.3 Algoritme voor specifieke Max/Min Leaf Tree

Het artikel wordt afgesloten met de presentatie van een algoritme voor Min Leaf IT (internal) en Max Leaf IT (internal) met exponentiële looptijd. Sterker, dit algoritme kan een opspannende boom bepalen, waarbij de bladverzameling de maximale/minimale grootte heeft en voldoet aan een gewenste eigenschap. De looptijd van het algoritme hangt af van de tijd die nodig is om deze eigenschap na te gaan.

Voor het algoritme worden alle mogelijke partities van de knopen van de graaf in bladeren en interne knopen bekeken. Vervolgens wordt voor iedere partitie bepaald of er een opspannende boom voor bestaat. Het bepalen hiervan is een NP-moeilijk probleem door middel van reductie van Hamiltonpad. Er geldt namelijk dat een graaf een Hamiltonpad bevat van knoop x tot knoop y precies dan als de graaf een opspannende boom bevat waarvan x en y de enige bladeren zijn.

Het algoritme maakt gebruik van dynamisch programmeren. We definiëren A_i als de verzameling partities in bladeren en interne knopen van een opspannende boom voor iedere deelgraaf van graaf $G = (V, E)$ met i knopen. Vanuit A_i kan vervolgens A_{i+1} worden samengesteld door voor iedere partitie na te gaan of de bijbehorende opspannende boom kan worden uitgebreid met een knoop. Uiteindelijk arriveren we dan bij A_n, de verzameling die de partities van V bevat in bladeren en interne knopen van een opspannende boom. Hieruit kunnen de partities die voldoen aan de gewenste eigenschappen worden gefilterd, voor independency trees bepalen we bijvoorbeeld voor welke partities de bladverzameling onafhankelijk is. Ten slotte kiest men de partitie waarbij de bladverzameling de minimale of maximale grootte heeft.

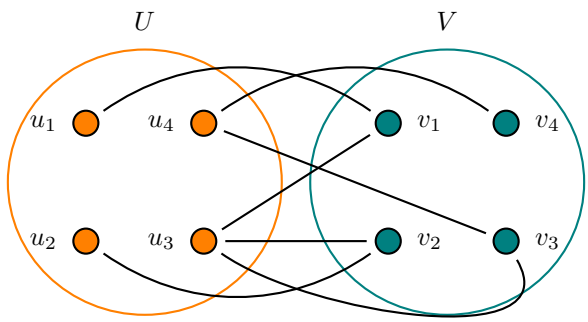
Om A_i te bepalen worden alle deelverzamelingen van V van grootte i bekeken. Hiervoor zijn $\binom{n}{i}$ mogelijkheden. Vervolgens bestaan er voor iedere knoop twee opties: de knoop bevindt zich in de verzameling bladeren of de knoop bevindt zich in de verzameling interne knopen. In totaal geeft dit voor iedere deelverzameling 2^i mogelijkheden. Voor het bepalen van iedere A_i worden dus $\binom{n}{i} \cdot 2^i$ mogelijkheden bekeken, waardoor voor het bepalen van $A_1, \ldots, A_n$ in totaal $\sum_{i=1}^{n} \binom{n}{i} \cdot 2^i = 3^n - 1$ partities worden bekeken. Vaststellen of een partitie kan worden uitgebreid met een knoop kost $n+m$ tijd (details hiervan zijn te vinden in het artikel), waardoor A_n in $O(3^n \cdot (n+m))$ tijd kan worden bepaald. Als vervolgens in $O(f(n))$ tijd de gewenste eigenschap kan worden nagegaan, vindt men dus de opspannende boom met het maximale of minimale aantal bladeren, die ook voldoen aan deze eigenschap in $O(3^n \cdot f(n))$ tijd. Aangezien nagaan of een verzameling knopen onafhankelijk is, kan worden gedaan in polynomiale tijd, levert deze aanpak algoritmen op voor Min Leaf IT en Max Leaf IT met looptijd $O(3^n)$.

3 Co-bipartiete graaf

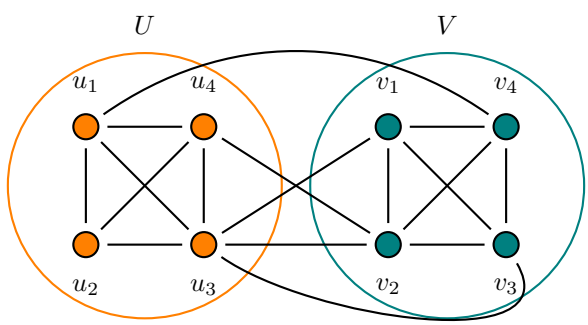
Een *bipartiete graaf* G is een graaf waarvan we de knopen kunnen verdelen over $\{U, V\}$, zodanig dat een knoop in U enkel verbonden is met knopen in V en vice versa. De knopen in U zijn dus paarsgewijs niet-verbonden en hetzelfde geldt voor de knopen in V. Het complement van een bipartiete graaf noemen we een *co-bipartiete graaf*. Dit is dus wederom een graaf G met een partitie $\{U, V\}$, waarbij nu, naast kanten die beide deelverzamelingen verbinden, U en V beide een kliek vormen. Een kant van een knoop in U naar een knoop in V zullen we een *brug* noemen in dit stuk. We gaan in dit hoofdstuk op zoek naar een algoritme met een polynomiaal begrensde looptijd dat een Max Leaf IT, een independency tree met het maximale aantal bladeren, vindt.

Voordat we dit doen, behandelen we het laatste deel van het bewijs van de eerste implicatie van het Theorema van Böhme, namelijk dat als G isomorf is met de complete bipartiete graaf $K_{n/2,n/2}$, dat G dan geen independency tree bevat. De graaf $K_{n/2,n/2}$ is een bipartiete graaf, waarbij beide verzamelingen U en V van de partitie dezelfde kardinaliteit hebben, namelijk $\frac{n}{2}$, en waarbij iedere knoop van U is verbonden met iedere knoop in V. Als er een independency tree T in een dergelijke graaf zou bestaan, zouden alle bladeren zich in één van de twee deelverzamelingen, zeg U, moeten bevinden. De knopen in V zijn dan geen bladeren, dus hebben een graad van minstens 2 in T. Echter, dan bevat T ten minste $\frac{n}{2} \cdot 2 = n$ kanten, dus bevat T een cykel.

In de volgende propositie zullen we laten zien dat een independency tree in een co-bipartiete graaf maximaal twee bladeren bevat. Vervolgens bekijken we hoe we een dergelijke independency tree kunnen bepalen.



Figuur 2: Een bipartiete graaf



Figuur 3: Een co-bipartiete graaf

Propositie 1. Zij $G = (V_G, E_G)$ een co-bipartiete graaf met partitie $\{U, V\}$. Iedere independency tree T in G heeft dan maximaal twee bladeren.

Bewijs. We nemen het omgekeerde aan: er bestaat een independency tree T in G met ten minste drie bladeren. Dan bevat U of V minstens twee bladeren van T. Neem zonder verlies van algemeenheid aan dat twee bladeren, zeg u_1 en u_2, van T zich in U bevinden. Echter, U is een kliek, dus $(u_1, u_2) \in E_G$. Hieruit concluderen we dat T geen independency tree is en we dus een tegenspraak hebben. □

Als iedere knoop in U verbonden is met iedere knoop in V, is de graaf volledig en bevat G geen independency tree. We bekijken dus een co-bipartiete graaf G waarin er een knoop in U en een knoop in V bestaan die geen kant onderling hebben. Nu hebben we nog één eigenschap nodig om een algoritme te kunnen vormgeven. We willen namelijk een blad in $b_1 \in U$ en een blad in $b_2 \in V$ bepalen, zodat we een brug (u, v) kunnen vinden tussen U en V met $u \neq b_1$ en $v \neq b_2$. Immers, dan kunnen we een Hamiltonpad in U van b_1 naar u en een Hamiltonpad in V van b_2 naar v bepalen, deze vervolgens verbinden met (u, v), wat een independency tree oplevert met b_1 en b_2 als bladeren. Deze benodigde eigenschap bewijzen we in Propositie 2.

Propositie 2. Zij $G = (V_G, E_G)$ een co-bipartiete graaf met partitie $\{U, V\}$, waarbij $|U|, |V| \geq 2$. Dan zijn er $u_B, b_1 \in U$ en $v_B, b_2 \in V$ zodat: $(u_B, v_B) \in E$, $(b_1, b_2) \notin E$, $u_B \neq b_1$, $v_B \neq b_2$.

Bewijs. Neem $u \in U$, $v \in V$ zodanig dat $(u, v) \notin E$. Als er nu $u_B \in U$, $v_B \in V$ bestaan met $(u_B, v_B) \in E$ en $u_B \neq u$, $v_B \neq v$, nemen we $b_1 = u$ en $b_2 = v$ en zijn we klaar.

Als nu voor alle $u^* \in U$, $v^* \in V$ met $(u^*, v^*) \in E$ geldt dat $u^* = u$ of $v^* = v$, onderscheiden we drie gevallen: u is verbonden met iedere knoop in $V - \{v\}$ en v is verbonden met iedere knoop in $U - \{u\}$; u is verbonden met iedere knoop in $V - \{v\}$, maar v niet met iedere knoop in $U - \{u\}$; u is niet verbonden met iedere knoop in $V - \{v\}$ en v is niet verbonden met iedere knoop in $U - \{u\}$.

In het eerste geval merken we op dat U of V dan tenminste drie knopen bevat. Als namelijk geldt dat $|U| = |V| = 2$, is G de cykelgraaf van vier knopen en bevat deze geen independency tree. Zonder verlies van algemeenheid nemen we aan dat $|V| \geq 3$. Nu nemen we een willekeurige brug die verbonden is met u, zeg (u, v^*). Vervolgens nemen we u en v^* als u_B en v_B en $b_1 \in U - \{u\}$ en $b_2 \in V - \{v, v^*\}$ willekeurig. Aangezien iedere brug incident is met u of v geldt daarvoor dat $(b_1, b_2) \notin E$.

In het tweede geval is u of v niet met iedere mogelijke knoop in het andere deel van de partitie is verbonden, zeg v, maar u is dat wel, dan is er dus een $u^* \in U - \{u\}$ zodat $(u^*, v) \notin E$. Dan nemen we $b_1 = u^*, b_2 = v, u_B = u$ en $v_B \in V - \{v\}$ willekeurig.

Ten slotte bekijken we geval drie: zowel u als v is niet verbonden met iedere knoop in respectievelijk $V - \{v\}$ en $U - \{u\}$. We weten dat de graaf samenhangend is, dus is er minstens één brug die verbonden is met u of v, zeg u met brug $(u, v^*) \in E$. Dan geldt dat $v^* \neq v$, want $(u, v) \notin E$. Daarnaast is v niet verbonden met alle knopen in $U - \{u\}$, dus is er een $u^* \in U - \{u\}$ zodat $(u^*, v) \notin E$. We nemen dan u en v^* als u_B en v_B en u^* als b_1 en v als b_2. □

Met deze informatie kunnen we nu een independency tree in G met Algoritme 2 vinden. Dit is vanwege Propositie 1 ook gelijk een Max Leaf IT.

Theorema 1. Zij $G = (V, E)$ een co-bipartiete graaf met partitie $\{U, V\}$. Dan is het probleem Max Leaf Independency Tree oplosbaar op G in $O(|U| \cdot |V|)$ tijd.

Bewijs. De correctheid van Algoritme 2 volgt uit Propositie 1 en Propositie 2.

De partitie $\{U, V\}$ van de co-bipartiete graaf vinden, kan in $O(n + m)$ tijd door middel van breadth-first-search [12]. We bepalen namelijk voor het complement van de graaf, een bipartiete graaf, de partitie $\{U, V\}$. Daarvoor beginnen we onze BFS bij een willekeurige knoop v, die we aan een deelverzameling toevoegen, bijvoorbeeld U. Knoop v heeft dan enkel buren in V, want de graaf is bipartiet, dus voegen we vervolgens alle buren van v toe aan V, waarna we hun buren weer toevoegen aan U. Dit herhalen we totdat we alle knopen aan U of V hebben toegevoegd. Zo vinden we de partitie $\{U, V\}$ van het complement van de graaf, wat automatisch ook de partitie van de graaf is.

Zij $G = (V_G, E_G)$ een co-bipartiete graaf met partitie $\{U, V\}$.

Stap 1: Bepaal $u^*, b_1 \in U$ en $v^*, b_1 \in V$ zodat $(u^*, v^*) \in E$, $(b_1, b_2) \notin E$ en voeg (u^*, v^*) toe aan independency tree T.

Stap 2: Neem een Hamiltonpad van u^* naar b_1 in U en een Hamiltonpad van v^* naar b_2 in V. Voeg beide Hamiltonpaden toe aan T.

Stap 3: Output Max Leaf IT T.

Algoritme 2: Het bepalen van Max Leaf IT T in co-bipartiete graaf G

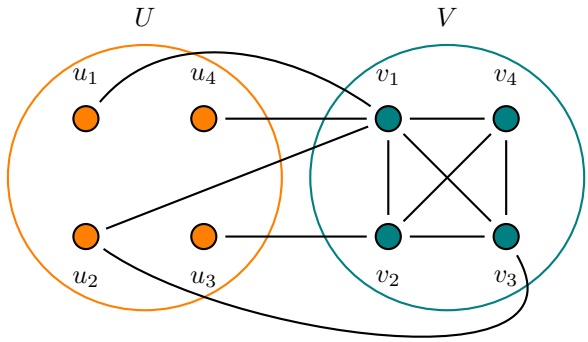
Het bewijs van Propositie 2 geeft ook een goede strategie om stap 1 van het algoritme uit te voeren. We kunnen namelijk in $O(|U| \cdot |V|)$ tijd de $u \in U$, $v \in V$ zodanig dat $(u, v) \notin E$ vinden en bepalen of er $u_B \in U$, $v_B \in V$ bestaan met $(u_B, v_B) \in E$ en $u_B \neq u$, $v_B \neq v$.

Als dergelijke u_B en v_B niet bestaan, kunnen we door voor u alle knopen uit $V - \{v\}$ en voor v alle knopen uit $U - \{u\}$ na te gaan, vaststellen in welke van de drie gevallen we ons begeven. Dit kost dus $O(|U|+|V|)$ tijd. Om ten slotte de daadwerkelijk juiste knopen te bepalen, hebben we in geval één $O(|U|)$ tijd nodig om te bepalen welk deel van de partitie minstens drie knopen bevat en verder enkel constante tijd. In geval twee gaan we voor v alle knopen in $U - \{u\}$ na, wat dus ook $O(|U|)$ tijd kost. Voor geval drie kunnen we in $O(|U| + |V|)$ tijd een brug die verbonden is met u of v vinden en vervolgens in $O(|U|)$ tijd de knoop u^* die niet verbonden is met v (of de knoop v^* die niet verbonden is met u, als de gevonden brug incident met v is).

Het vinden van Hamiltonpaden in klieks voor stap 2 kan vervolgens in lineaire tijd, dus de totale looptijd van Algoritme 2 is $O(|U| \cdot |V|)$. $\square$

4 Splitgraaf

Een splitgraaf is een graaf waarvoor er een partitie $\{U, V\}$ van de knopen bestaat, zodat U een onafhankelijke deelverzameling is en V een kliek. Verder bestaat er een aantal kanten tussen U en V. Allereerst merken we op dat slechts één knoop in V een blad kan zijn van een independency tree. Immers, V is een kliek, dus als twee knopen uit V bladeren zijn van een opspannende boom, hebben deze twee bladeren een kant gemeenschappelijk. Een independency tree kan dus niet meer dan $|U| + 1$ bladeren hebben. Verder weten we dat iedere knoop in U minstens één brug naar V heeft, anders is de graaf niet samenhangend. We onderscheiden twee gevallen: V bevat een knoop zonder brug naar U en V bevat enkel knopen met bruggen naar U. In Propositie 3 zullen we het eerste geval bekijken en in Propositie 4 het tweede geval.



Figuur 4: Een splitgraaf

Propositie 3. Zij $G = (V|_G, E|_G)$ een splitgraaf met partitie $\{U, V\}$, waarbij V een knoop v_k bevat zonder brug naar U. Dan bestaat er een independency tree T in G met $|U| + 1$ bladeren.

Bewijs. Vanwege de samenhang van G bestaat er een $v_i \in V$ met minstens één brug naar een knoop, zeg u_j, van U. Bepaal een Hamiltonpad $v_i, \ldots, v_k$ van V en voeg deze toe aan T. Voeg vervolgens (v_i, u_j) toe aan T. Ten slotte voegen we voor iedere knoop in U een willekeurige brug naar V toe aan T. Dit levert een independency tree T op waarbij alle knopen in U en v_k de bladeren vormen. □

Als V geen knopen bevat zonder bruggen naar U, verandert de situatie enigszins. We nemen voor de Max Leaf IT wederom alle knopen in U als bladeren. Als we dan een Hamiltonpad van V bepalen, moeten beide uiteinden hiervan in de independency tree worden verbonden aan verschillende knopen in U. Anders is één van beide namelijk een blad van de gevonden boom en vinden we dus geen independency tree. Deze eigenschap bewijzen we in de volgende propositie. De Max Leaf IT heeft in dit geval dus $|U|$ bladeren.

Propositie 4. Zij $G = (V|_G, E|_G)$ een splitgraaf met partitie $\{U, V\}$, waarbij $|U|, |V| \geq 2$ en iedere knoop in V verbonden is met minstens één knoop in U. Dan bestaan er $v_i, v_j \in V$ met $(v_i, u_a), (v_j, u_b) \in E|_G$ voor $u_a, u_b \in U$ met $u_a \neq u_b$.

Bewijs. Vanwege de samenhang van G bestaat er een $v_i \in V$ met minstens één brug naar een knoop, zeg u_j, van U. Neem nu aan dat de stelling niet geldt. Dan is iedere knoop in U dus alleen verbonden met v_i. Echter, voor $v_j \in V$ met $v_j \neq v_i$ geldt dan dat deze geen brug heeft naar U, wat een tegenspraak is. □

Met deze kennis kunnen we het onderstaande Algoritme 3 samenstellen dat een Max Leaf IT vindt voor een splitgraaf $G = (V|_G, E|_G)$ met partitie $\{U, V\}$.

Zij $G = (V|_G, E|_G)$ een splitgraaf met partitie $\{U, V\}$ en $|U|, |V| \geq 2$.

Stap 0 Bepaal of er een $v_i \in V$ bestaat zonder brug naar U.

Zo ja:

Stap 1: Vind een Hamiltonpad $v_i, \ldots v_k$ van V, waarvoor brug $(v_k, u_j) \in E_G$ bestaat voor een $u_j \in U$. Voeg zowel het Hamiltonpad als (v_k, u_j) toe aan T.

Stap 2: Voeg voor iedere overige knoop in U een willekeurige brug naar V toe.

Stap 3: Output Max Leaf IT T.

Zo nee:

Stap 1: Bepaal de $v_i, v_j \in V$ met $(v_i, u_a), (v_j, u_b) \in E$ voor $u_a, u_b \in U$ met $u_a \neq u_b$.

Stap 2: Vind een Hamiltonpad $v_i, \ldots, v_j$ van V en voeg deze samen met (v_i, u_a) en (v_j, u_b) toe aan T.

Stap 3: Voeg voor iedere overige knoop in U een willekeurige brug naar V toe.

Stap 4: Output Max Leaf IT T.

Algoritme 3: Het bepalen van Max Leaf IT T in splitgraaf G

Theorema 2. Zij $G = (V, E)$ een splitgraaf met partitie $\{U, V\}$. Dan is het probleem Max Leaf Independency Tree oplosbaar op G in $O(n)$ tijd.

Bewijs. De correctheid van Algoritme 3 volgt uit Propositie 3 en Propositie 4.

De partitie $\{U, V\}$ van splitgraaf G kunnen we in $O(n+m)$ tijd bepalen [13; 14]. Dit kunnen we doen door de knopen te sorteren op oplopende graad. In splitgrafen resulteert dit in een *perfect elimination ordering* van de knopen, waar we in Hoofdstuk 6 dieper op in zullen gaan. Vervolgens bepalen we de kardinaliteit k van de grootste kliek in G, wat mogelijk is in $O(n+m)$ tijd [15]. Nu kunnen we de knopen die onderdeel zijn van de grootste kliek bepalen, omdat deze knopen graad minstens $k-1$ hebben. De knopen in de kliek vormen V en de overige knopen zijn de onafhankelijke deelverzameling U van de splitgraaf.

Door de graden van de knopen in V te bekijken, kan stap 0 in $O(|V|)$ tijd worden uitgevoerd. In het ja-geval kunnen we de gewenste brug vinden door wederom de graden van deze knopen na te gaan. Als deze namelijk groter is dan $|V|$, is de knoop verbonden met een knoop in U. De uiteindelijke looptijd van het algoritme is dan dus $O(|V|)$.

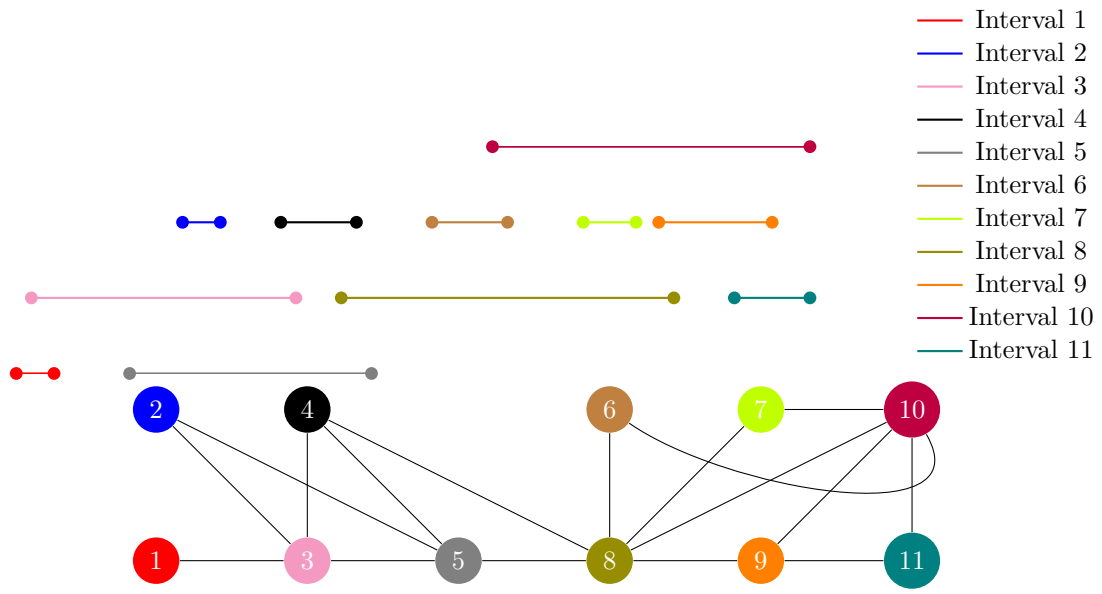
Bij een 'nee' bij stap 0, kunnen we stap 1 als volgt uitvoeren: we bekijken wederom de graden van de knopen in V. Als er een knoop v_i is die graad minstens $|V| + 2$ heeft, nemen we vervolgens een willekeurige andere knoop v_j in V met een willekeurige brug $(v_j, u_b) \in E$. Knoop v_i heeft dan minstens twee buren in U, dus kunnen we de u_a bepalen met $(v_i, u_a) \in E$ en $u_a \neq u_b$.

Als er geen knoop in V is die graad minstens $|V|+2$ heeft, heeft iedere knoop in V dus precies één brug naar U. We nemen dan een willekeurige brug, zeg $(v_i, u_a) \in E$. Vervolgens nemen we een willekeurige $u_b \in U$, met een willekeurige brug naar V, zeg $(v_j, u_b) \in E$. De knopen in V zijn slechts met één knoop in U, dus er geldt $v_i \neq v_j$.

Ook in het nee-geval zijn de overige stappen uitvoerbaar in lineaire tijd, waardoor het totale algoritme in lineaire tijd kan worden uitgevoerd. □

5 Intervalgraaf

De volgende graafklasse die we gaan bekijken is die van de *intervalgraaf.* Een dergelijke graaf is een representatie van een verzameling intervallen over de reële lijn. Deze intervallen worden ieder weergegeven als een knoop in de bijbehorende graaf, waarbij twee knopen door een kant verbonden worden precies dan als de intervallen een punt gemeenschappelijk hebben. We identificeren de knopen van een intervalgraaf met de bijbehorende intervallen. In het bijzonder stelt dit ons in staat om te spreken over doorsnijdingen van knopen, waar we de doorsnijding van de bijbehorende intervallen mee bedoelen. Daarnaast heeft een interval een linkerpunt en een rechterpunt. We noteren dit voor interval i als $links(i)$ en $rechts(i)$ of voor knoop v als $links(v)$ en $rechts(v)$.



Figuur 5: Een intervalgraaf

5.1 Bepalen van de optimale bladverzameling

Om een Max Leaf IT T te vinden in een intervalgraaf $G = (V, E)$, bepalen we eerst een onafhankelijke verzameling knopen $B \subset E$, die de bladeren van T zullen vormen. Eerst merken we op dat knopen van graad 1 een blad zijn in iedere opspannende boom. Intervallen die slechts één ander interval doorsnijden, moeten dus in ieder geval aan B worden toegevoegd. Een *cutvertex* kan daarentegen geen blad zijn van een opspannende boom. Een cutvertex is een knoop v, waarvoor de graaf $G' = (V - \{v\}, E|_{V-\{v\}})$ niet samenhangend is. Als een dergelijke knoop een blad zou zijn van een boom, is een deel van de graaf geen onderdeel van de boom en is de boom dus niet opspannend. Een verzameling bladeren, of *bladverzameling,* is dus een onafhankelijke verzameling knopen, die alle knopen van graad 1, maar geen cutvertices bevat. De bladverzameling met de grootst mogelijke kardinaliteit bepalen we door middel van Algoritme 4. In dit algoritme behandelen we verzameling J op volgorde. In stap 1 worden de knopen van graad 1 toegevoegd aan de verzameling, waarna in stap 2 de overige knopen erachter worden toegevoegd.

Nu willen we bewijzen dat de gevonden bladverzameling B ook daadwerkelijk de grootst mogelijke kardinaliteit heeft. Hiervoor introduceren we eerst een propositie om deze vervolgens te gebruiken om de optimaliteit van B aan te tonen. We merken op dat voor een onafhankelijke verzameling knopen geldt dat geen intervallen van de bijbehorende knopen doorsnijden. Daardoor kan een dergelijke verzameling knopen worden gesorteerd op de rechterpunten van de intervallen, zodat $B = \{b_1, \ldots, b_k\}$ met $rechts(b_i) < rechts(b_j)\ \forall\ 1 \leq i < j \leq k$.

Zij $I = \{I_1, \ldots, I_n\}$ een collectie intervallen met bijbehorende intervalgraaf $G = (V, E)$.

Stap 1: Bepaal welke intervallen precies één ander interval doorsnijden en voeg deze toe aan een lege verzameling J.

Stap 2: Voeg de overige intervallen van I, gesorteerd op oplopende rechterpunten, achter de al toegevoegde intervallen toe aan J.

Stap 3: Neem x = het eerste interval van J.
Als $G' = (V - \{x\}, E|_{V-\{x\}})$ samenhangend is: voeg x toe aan bladverzameling B en verwijder x en alle intervallen die x doorsnijden uit J.

Als G' niet samenhangend is: verwijder x uit J.

Stap 4: Herhaal stap 3 totdat J leeg is.

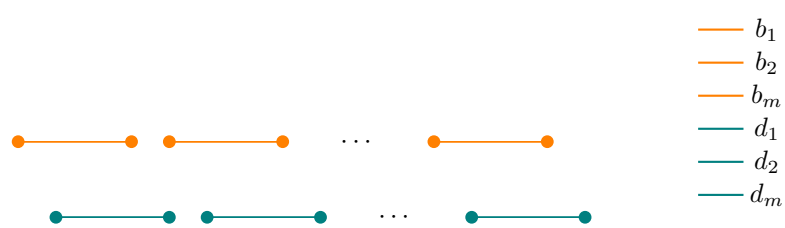
Stap 5: Output B.

Algoritme 4: Het bepalen van optimale bladverzameling B voor een intervalgraaf

Propositie 5. Zij $G = (V, E)$ een intervalgraaf. Laat vervolgens $B = \{b_1, \ldots, b_k\} \subset V$ de door Algoritme 4 bepaalde bladverzameling zijn en $D = \{d_1, \ldots, d_l\} \subset E$ een willekeurig andere bladverzameling. Sorteer beide verzamelingen op de rechterpunten van de intervallen. Dan geldt: $rechts(d_i) \geq rechts(b_i)$ $\forall\ 1 \leq i \leq \min\{k, l\}$.

Bewijs. We maken gebruik van inductie. Ten eerste bekijken we b_1. Door de vormgeving van Algoritme 4 is b_1 de eerste knoop die geen cutvertex is van G. Er geldt dus dat $rechts(d_1) \geq rechts(b_1)$. Nu nemen we aan dat $rechts(d_i) \geq rechts(b_i)$ $\forall\ 1 \leq i \leq m$ voor een zekere $m < \min\{k, l\}$. De situatie ziet er dan uit zoals in Figuur 6.

Stel nu dat $rechts(d_{m+1}) < rechts(b_{m+1})$. We weten dat $rechts(d_m) \geq rechts(b_m)$ (inductiehypothese) en dat $rechts(d_m) < links(d_{m+1})$. Er geldt dus dat $d_{m+1} \cap b_m = \emptyset$. Uiteraard geldt ook $b_{m+1} \cap b_m = \emptyset$. Na het toevoegen van b_m aan B door het algoritme, bevinden b_{m+1} en d_{m+1} zich dus beide nog in J. Echter, doordat $rechts(d_{m+1}) < rechts(b_{m+1})$, wordt d_{m+1} eerder toegevoegd dan b_{m+1}, wat een tegenspraak oplevert. Er geldt dus dat $rechts(d_{m+1}) \geq rechts(b_{m+1})$. □



Figuur 6

Uit Propositie 6 volgt de optimaliteit van bladverzameling B:

Propositie 6. Zij $G = (V, E)$ een intervalgraaf en $B = \{b_1, \ldots, b_k\} \subset V$ de door Algoritme 4 bepaalde bladverzameling. Dan heeft G geen independency tree met meer dan k bladeren.

Bewijs. Zij $O = \{o_1, \ldots, o_l\} \subset V$ de verzameling bladeren van een Max Leaf IT van G. Verzameling O is een bladverzameling, dus volgens Propositie 5 geldt dat $rechts(o_i) \geq rechts(b_i)\ \forall\ 1 \leq i \leq \min\{k, l\}$ en in het bijzonder dat $rechts(o_{\min\{k,l\}}) \geq rechts(b_{\min\{k,l\}})$.

Stel nu dat $l > k$. Neem dan $k < j \leq l$ willekeurig. Dan geldt dat $links(o_j) > rechts(o_k) \geq rechts(b_k)$ en dus dat $o_j \cap b_i = \emptyset\ \forall\ 1 \leq i \leq k$.

Nadat Algoritme 4 b_k heeft toegevoegd aan B, bevindt o_j zich dus nog in J. Deze zal dus worden toegevoegd aan B, wat een tegenspraak oplevert. □

5.2 Bladverzameling uitbreiden naar Max Leaf IT

We hebben nu een algoritme dat ons de verzameling bladeren van een Max Leaf IT oplevert. Dit willen we uitbreiden, zodat we ook de independency tree zelf kunnen bepalen. We gebruiken hiervoor de volgende strategie: we bepalen eerst aan de hand van Algoritme 1 een independency tree of een Hamiltonpad dat bevat is in een Hamiltoncykel in de graaf $G' = (V - B, E|_{V-B})$. Vervolgens verbinden we ieder blad van de independency tree of de uiteinden van het Hamiltonpad met minstens één blad uit B. Ten slotte voegen we de overige bladeren uit B toe, wat de Max Leaf IT oplevert. Deze strategie resulteert in Algoritme 5. In het vervolg zullen we de nodige proposities behandelen om te laten zien dat het algoritme ook daadwerkelijk de gewenste boom bepaalt.

Zij $G = (V, E)$ een intervalgraaf.

Stap 1: Vind de bladverzameling B met maximale grootte door middel van Algoritme 4.

Stap 2: Bepaal een willekeurige opspannende boom T' van $G' = (V - B, E|_{V-B})$.

Stap 3: Voer Algoritme 1 uit op T', zodat T' een independency tree of een Hamiltonpad bevat in een Hamiltoncykel van G' is.

Stap 4: Als T' een independency tree is: voeg T' toe aan T en voeg vervolgens voor ieder blad van T' een kant toe naar een blad in B.

Als T' een Hamiltonpad is, bevat in Hamiltoncykel $T' + \{e\}$: bepaal het Hamiltonpad in $V - B$ waarbij de twee uiteinden verbonden zijn met twee verschillende bladeren in B en voeg het Hamiltonpad en deze twee kanten toe aan T.

Stap 5: Voeg kanten van de overgebleven bladeren in B naar T' toe aan T.

Stap 6: Output T.

Algoritme 5: Het bepalen van Max Leaf IT T in intervalgraaf G

Reeds hebben we aangetoond dat stap 1 tot en met stap 3 probleemloos kunnen worden uitgevoerd. Echter, stap 4 zou voor problemen kunnen zorgen. Als er namelijk een blad b' van T' niet verbonden kan worden met een blad in B, dan zal b' ook een blad van de gevonden boom T zijn. Aangezien B de maximale grootte heeft, is T dan geen independency tree. Als T' nadat Algoritme 1 is toegepast een independency tree is, rest ons dus te laten zien dat alle bladeren van T' verbonden kunnen worden aan een blad in B. In het geval dat T' een Hamiltonpad is, moeten er twee buren in de Hamiltoncykel $T + \{e\}$ bestaan die verbonden zijn met verschillende bladeren in B. We zullen een aantal proposities behandelen, die ons uiteindelijk in staat stellen deze eigenschappen te bewijzen.

Ten eerste behandelen we de situatie dat T' een independency tree is. We laten eerst zien ieder blad van T' verbonden is met een blad uit B. Vervolgens breiden we dit uit naar een bewijs van de gewenste eigenschap.

Propositie 7. Zij $G = (V, E)$ een intervalgraaf, B de gevonden bladverzameling, T' een independency tree van $G' = (V - B, E|_{V-B})$ met bladverzameling B'. Dan geldt: voor iedere knoop $b' \in B'$ bestaat er een $b \in B$ zodat $(b', b) \in E$.

Bewijs. Neem het omgekeerde aan: er bestaat een $b' \in B'$ met $(b', b) \notin E \ \forall \ b \in B$. Dan geldt dus dat $b' \cap b = \emptyset \ \forall \ b \in B$, dus is $B \cup \{b'\}$ een onafhankelijke verzameling knopen. Als $B \cup \{b'\}$ een grotere bladverzameling is dan B, dan is dit in tegenspraak met Propositie 6, dus b' is een cutvertex van G.

Echter, b' is een blad van independency tree T', dus b' is geen cutvertex van G'. Knoop b' verbindt dus knopen in $V - B$ met knopen in B, dus moet $b' \cap b \neq \emptyset$ voor een $b \in B$ en hebben we een tegenspraak. □

We willen aantonen dat indien T' een independency tree is van $G' = (V - B, E|_{V-B})$ met bladverzameling B', het mogelijk is om iedere knoop in B' te verbinden met minstens één blad in B. Met andere woorden, we willen laten zien dat er een koppeling van B' bestaat in de bipartiete graaf $H = (B \cup B', E_{|B \cup B'})$. Het Theorema van Hall [16] geeft een noodzakelijke en toereikende voorwaarde:

Theorema van Hall. Zij $G = (V, E)$ een eindige bipartiete graaf met partitie $\{A, B\}$. Dan bevat G een koppeling van A precies dan als $|N(S)| \geq |S| \ \forall \ S \subset A$.

Hierbij is $N(S)$ de *omgeving* van S, oftewel de verzameling knopen die verbonden zijn met minstens één knoop in S. De knopen in een deelverzameling $S \subset B'$ zijn alleen verbonden met knopen in B in H, omdat B' een verzameling onafhankelijk knopen is. De omgeving van $S \subset B'$ ziet er dus als volgt uit: $N(S) = \{b \in B \mid (b', b) \in E \text{ voor een } b' \in S\}$. Voor de omgeving van een deelverzameling van B geldt hetzelfde.

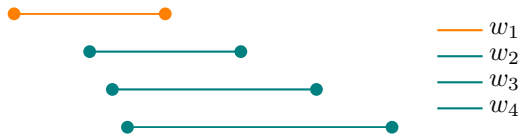
Aan de hand van het Theorema van Hall kunnen we bewijzen dat er een koppeling van B' in bipartiete graaf $H = (B \cup B', E_{|B \cup B'})$ bestaat.

Propositie 8. Zij $G = (V, E)$ een intervalgraaf, B de gevonden bladverzameling en T' een independency tree van $G' = (V - B, E|_{V-B})$ met bladverzameling B'. Dan bestaat er een koppeling van B' in bipartiete graaf $H = (B \cup B', E_{|B \cup B'})$.

Bewijs. Zij $S \subset B'$ willekeurig. We noteren $S = \{b'_1, b'_2, \ldots, b'_k\}$ en $N(S) = \{b_1, b_2, \ldots, b_l\} \subset B$, waarbij beide verzamelingen gesorteerd zijn op de rechterpunten van de intervallen. Ook $W = S \cup N(S) = \{w_1, \ldots, w_{k+l}\}$ en $V = \{v_1, \ldots, v_n\}$ zijn gesorteerd op de rechterpunten van de bijbehorende intervallen.

We maken eerst gebruik van inductie om te bewijzen dat $rechts(b'_i) \geq rechts(b_i) \ \forall \ 1 \leq i < \min\{k, l\}$.

We bekijken $w_1 \in W$, waarvoor geldt dat $rechts(w_i) \geq rechts(w_1) \ \forall 1 \leq i \leq k + l$. Als w_1 graad 1 in G heeft, is het een blad van iedere opspannende boom, dus geldt $w_1 \in B$ en $w_1 \in N(S)$. Als w_1 graad ≥ 2 in G heeft, zijn de buren van w_1 ook automatisch buren van elkaar. Dit illustreren we met Figuur 7.

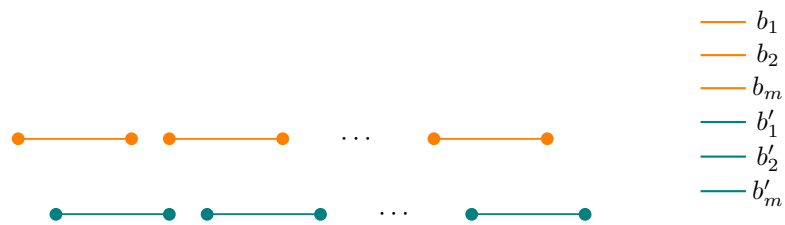


Figuur 7

Zoals we in het bewijs van Propositie 7 zagen, kan een knoop $b' \in B'$ alleen een cutvertex zijn van G als de bijbehorende cutedge $(b', b) \in E$ is voor een $b \in B$. Als w_1 een cutvertex van G is, is (w_1, w_i) dus de bijbehorende cutedge voor een $1 < i \leq k + l$. Echter, er moet ook gelden dat $rechts(w_i) < rechts(w_1)$, want anders geldt dat $(w_2, w_i) \in E$ en is (w_1, w_i) geen cutedge. Dit is een tegenspraak met het feit dat $rechts(w_i) \geq rechts(w_1) \ \forall 1 \leq i \leq k+l$, waaruit volgt dat w_1 geen cutvertex van G is. Dus ook als de graad

van w_1 minstens 2 is, zal het algoritme w_1 toevoegen aan B en geldt dus $w_1 \in N(S)$. We concluderen dat $b_1 = w_1$ en $rechts(w_i) \geq rechts(b_1)$ $\forall 1 \leq i \leq n$ en in het bijzonder $rechts(b'_1) \geq rechts(b_1)$

Nu nemen we aan dat $rechts(b'_i) \geq rechts(b_i)$ $\forall$ $1 \leq i \leq m$ voor een zekere $1 \leq m < \min\{k, l\}$. Deze situatie ziet eruit zoals in Figuur 8.



Figuur 8

Vervolgens bekijken we b'_{m+1}. Wederom volgt uit Propositie 7 dat $|N(b'_{m+1})| \geq 1$. Ook geldt dat $rechts(b_m) \leq rechts(b'_m) < links(b'_{m+1})$, dus $b'_{m+1} \cap b_i = \emptyset$ $\forall$ $1 \leq i \leq m$. Als vervolgens geldt dat $links(b_{m+1}) > rechts(b'_{m+1})$, dan $|N(b'_{m+1})| = 0$ en hebben we een tegenspraak, dus er geldt dat $links(b_{m+1}) \leq rechts(b'_{m+1})$. We nemen nu aan dat $rechts(b_{m+1}) > rechts(b'_{m+1})$. Dit ziet er dan uit zoals in Figuur 9.

Figuur 9

Er geldt dan dat b_{m+1} wel en b'_{m+1} niet is toegevoegd aan B, terwijl b'_{m+1} geen kant heeft met b_i voor $1 \leq i \leq m$ en terwijl $rechts(b_{m+1}) > rechts(b'_{m+1})$. Knoop b'_{m+1} is dus een cutvertex van G. Echter, b'_{m+1} is een blad van opspannende boom T' van G', dus b'_{m+1} is geen cutvertex van G'. Daarnaast kan b'_{m+1} geen andere buur in B hebben, want $rechts(b_{m+1}) > rechts(b'_{m+1})$, dus is de bijbehorende cutedge $(b'_{m+1}, b_{m+1}) \in E$.

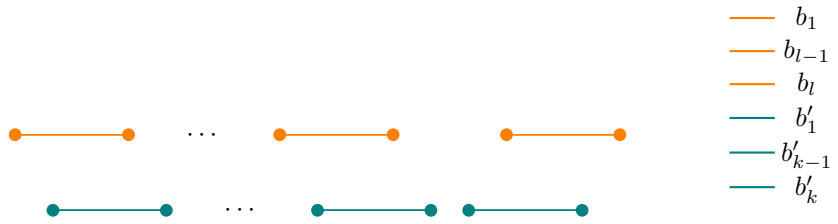
Stel dat er een $v \in V$ bestaat met $rechts(v) \geq rechts(b'_{m+1})$. Dan moet gelden dat $(w, b_{m+1}) \in E$ voor een $w \in V$ (mogelijk $v = w$) met $rechts(w) \geq rechts(b'_{m+1})$, anders is G geen samenhangende graaf. Echter, als $(w, b_{m+1}) \in E$, dan is (b'_{m+1}, b_{m+1}) geen cutedge en hebben we een tegenspraak. Er geldt dus dat $b_{m+1} = b_{|B|} = v_n$ en $b'_{m+1} = b'_{|B'|}$.

We concluderen dat $rechts(b'_i) \geq rechts(b_i)$ $\forall$ $1 \leq i < \min\{k, l\}$ en dat $rechts(b'_k) < rechts(b_l)$ kan gelden als $b_l = v_n$ en $b'_k = b'_{|B'|}$ een cutvertex van G is met cutedge (b'_k, b_l).

Neem nu aan dat $|N(S)| < |S|$, oftewel, er geldt $l < k$. We bekijken eerst het geval dat $rechts(b'_l) \geq rechts(b_l)$. Voor $b'_i \in S$ met $l < i \leq k$ geldt dan dat $links(b'_i) > rechts(b'_l) \geq rechts(b_l)$. Er geldt dus dat $|N(b'_i)| = 0$, wat een tegenspraak oplevert. Nu bekijken we het geval dat $b_l = v_n$ en b'_k een cutvertex van G is met cutedge (b'_k, b_l) en dat $rechts(b'_k) < rechts(b_l)$. Deze situatie schetsen we in Figuur 10.

Wederom bekijken we een $b'_i \in S$ met $l < i \leq k$. Daarvoor geldt dus dat $links(b'_i) > rechts(b'_l) \geq rechts(b_l)$, maar ook dat $links(b'_i) \leq rechts(b'_i) \leq rechts(b_l)$, want $b_l = v_n$. Dit is een tegenspraak.

Hiermee hebben we aangetoond dat $|N(S)| \geq |S|$ voor iedere deelverzameling $S \subset B'$ en daarmee volgens het Theorema van Hall dat er een koppeling bestaat van B' in $H = (B \cup B', E_{|B \cup B'})$. $\square$



Figuur 10

Met Propositie 8 hebben we bewezen dat indien T' een independency tree is, Algoritme 5 inderdaad de gewenste Max Leaf IT T oplevert. Nu rest ons te laten zien dat het algoritme ook werkt als T' een Hamiltonpad is, die bevat is in een Hamiltoncykel $T' + \{e\}$ voor een $e \in E$.

Propositie 9. Zij $G = (V, E)$ een intervalgraaf, B de gevonden bladverzameling, T' een Hamiltonpad van $G' = (V - B, E|_{V-B})$ bevat in Hamiltoncykel $T' + \{e\}$. Dan bestaan er $v_1, v_2 \in V - B$ en $b_1, b_2 \in B$ zodat $(v_1, b_1), (v_2, b_2) \in E$ en $(v_1, v_2) \in T' + \{e\}$ voor $v_1 \neq v_2$ en $b_1 \neq b_2$.

Bewijs. Voor iedere $b \in B$ geldt dat $b \cap v \neq \emptyset$ voor een $v \in V - B$, vanwege de samenhang van G. Daarnaast zijn knopen in $V - B$ onderdeel van een Hamiltoncykel van G', dus geen cutvertex van G'. Stel dat er een knoop $v \in V - B$ bestaat zonder kant naar B. Dan is v dus een cutvertex van G, anders is $B \cup \{v\}$ een grotere bladverzameling dan B, wat in tegenspraak is met Propositie 6. Echter, v is geen cutvertex van G', dus verbindt v knopen in $V - B$ met knopen in B en hebben we een tegenspraak. Iedere $v \in V - B$ is dus verbonden met minstens één blad in B.

We nemen aan dat de stelling niet geldt. Dan geldt voor ieder tweetal $v_1, v_2 \in V - B$, waarbij $(v_1, v_2) \in T' + \{e\}$, dat $(v_1, b), (v_2, b) \in E$ voor een $b \in B$ de enige verbindingen met B zijn. Dit geldt voor ieder tweetal buren in Hamiltoncykel $T' + \{e\}$, dus alle knopen in $V - B$ zijn verbonden met b en niet-verbonden met iedere andere knoop in B. Echter, iedere boom heeft tenminste twee bladeren, dus $|B| \geq 2$. Er is dus een $b^* \in B$ met $b^* \neq b$, waarvoor ook geldt dat $(b^*, v) \in E$ voor een $v \in V - B$, wat een tegenspraak oplevert. □

Hiermee is bewezen dat iedere stap die het algoritme uitvoert mogelijk is en de gewenste Max Leaf IT T oplevert.

5.3 Complexiteitsanalyse

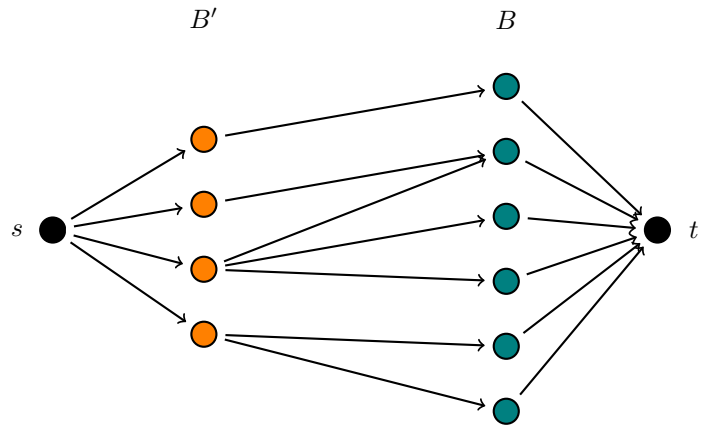
Theorema 3. Zij $G = (V, E)$ een intervalgraaf. Dan is het probleem Max Leaf Independency Tree oplosbaar op G in $O(nm)$ tijd.

Bewijs. De correctheid van Algoritme 4 volgt uit Propositie 5 en Propositie 6. Daarnaast volgt de correctheid van Algoritme 5 uit Propositie 7, Propositie 8 en Propositie 9.

De looptijd van Algoritme 4 is $O(nm)$. We gaan er vanuit dat de intervallen en in het bijzonder de rechterpunten die horen bij de knopen bekend zijn. Indien dit niet het geval is, kan de intervalrepresentatie van een intervalgraaf in lineaire tijd ($O(n)$) worden bepaald [17]. Hierbij wordt gebruik gemaakt van *lexicografische breadth-first-search*, wat we in het volgende hoofdstuk zullen behandelen.

Bepalen welke knopen graad 1 hebben, kan in lineaire tijd door de graden van alle knopen te bekijken. We noteren wederom n voor het aantal knopen en m voor het aantal kanten in de graaf. Sorteren van de intervallen op rechterpunten is mogelijk met ieder bekend sorteeralgoritme en kan dus in $O(n \log(n))$ tijd. Vervolgens gaan we voor maximaal n knopen na of het een cutvertex representeert en verwijderen we eventueel zijn buren. Controleren of een knoop een cutvertex is, kan in $O(n + m)$ tijd door middel van bijvoorbeeld breadth-first-search. Het verwijderen van de buren van een knoop kost slechts lineaire tijd. De totale looptijd van Algoritme 4 is zo $O(nm)$.

Voor Algoritme 5 bepalen we vervolgens een opspannende boom van G'. Dit kan door middel van BFS in $O(|V - B| + |E|_{V-B}|)$ tijd. Zoals we eerder zagen kost het toepassen van Algoritme 1 op deze boom $O(n^2)$ tijd. Als het toepassen van Algoritme 1 een independency tree oplevert, willen we een koppeling van B' in $H = (B \cup B', E|_{B \cup B'})$ vinden. Dit kunnen we doen door H iets aan te passen [18]. We voegen namelijk een knoop s (bron) toe, met pijlen naar de knopen in B'. Vervolgens veranderen we de ongerichte kanten in $E|_{B \cup B'}$ in gerichte pijlen van de knopen in B' naar de knopen in B. Ten slotte voegen we knoop t (put) toe met pijlen van alle knopen in B naar t. De gevormde graaf ziet er dan uit zoals in Figuur 11. We kennen voor iedere pijl gewicht 1 toe en vinden met het Ford-Fulkerson algoritme de maximale stroming. Op deze manier vinden we een maximale bipartiete matching in H, wat vanwege Propositie 12 een koppeling van B' in H is. Het Ford-Fulkerson algoritme bepaalt telkens een pad van s naar t in de restgraaf, zolang een dergelijk pad nog bestaat, totdat het de maximale stroming heeft bepaald. Oftewel, in ons geval, totdat het iedere knoop in B' heeft gekoppeld aan een knoop in B. Een dergelijk pad (als deze bestaat) kan in $O(m)$ tijd worden bepaald, dus de looptijd is $O(|B'| \cdot m)$.



Figuur 11

Nadat we deze koppeling hebben gevonden, voegen we voor de overige bladeren in B een willekeurige kant naar B' toe en hebben we de uiteindelijke Max Leaf Tree T bepaald.

In het geval van een Hamiltonpad kunnen we voor iedere knoop in $V - B$ nagaan of deze samen met zijn buur voldoet aan de gewenste eigenschappen. In beide gevallen blijft de looptijd van Algoritme 5 dus $O(nm)$. □

6 Chordale graaf

De laatste graafklasse die we bekijken is die van de *chordale graaf*. Dit is een graaf waarin iedere cykel van lengte minstens 4 een koorde bevat. Het is eenvoudig in te zien dat een splitgraaf chordaal is. Als er namelijk een partitie $\{U, V\}$ van de knopen bestaat, zodat U een onafhankelijke verzameling knopen is en V een kliek, dan heeft iedere cykel maximaal één knoop in U. Een cykel van lengte vier of groter bevat dus altijd een koorde.

De intervalgrafen vormen ook een deelverzameling van de chordale grafen [19; 20]. Als er namelijk wel een cykel $v_1, \ldots v_k$ van minstens vier knopen zonder koorde in een intervalgraaf zou bestaan, dan kunnen we voor iedere doorsnijding $v_i \cap v_{i+1}$ een punt nemen. Aangezien er geen koorden in de cykel voorkomen, wordt dit een strikt stijgende of strikt dalende rij punten. Echter, dan is de doorsnijding $v_1 \cap v_k$ leeg, dus is $v_1, \ldots v_k$ geen cykel.

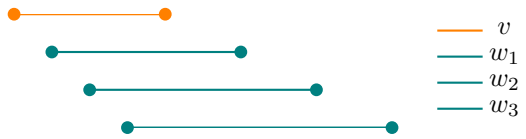
Een co-bipartiete graaf hoeft daarentegen niet chordaal te zijn. Als we namelijk een co-bipartiete graaf bekijken met partitie $\{U, V\}$, waarin twee bruggen $(u_1, v_1), (u_2, v_2) \in E$ bestaan met $u_1 \neq u_2$ en $v_1 \neq v_2$ en waarin $(u_1, v_2), (u_2, v_1) \notin E$, bevat deze graaf cykel u_1, u_2, v_2, v_1, u_1 zonder koorde. Uiteraard kunnen co-bipartiete grafen wel chordaal zijn.

6.1 Perfect elimination ordering

Omdat intervalgrafen chordaal zijn, is het logisch om te onderzoeken of het algoritme uit het vorige hoofdstuk veralgemeniseerd kan worden, zodat het werkt voor alle chordale grafen. De natuurlijke volgorde die volgt uit intervallen op een getallenlijn bestaat echter niet voor chordale grafen.

Een andere definitie voor chordale grafen geeft wel een manier om knopen te ordenen. Een graaf is namelijk chordaal precies dan als het een *perfect elimination ordering* (PEO) heeft. Een PEO is een volgorde van knopen zodanig dat iedere knoop v met zijn buren die ná v voorkomen in de PEO een kliek vormen.

Als we deze manier van het sorteren van knopen nu vergelijken met sorteren op rechterpunten bij de intervalgrafen, valt ons iets op. Als we namelijk een knoop v van een intervalgraaf bekijken, samen met de buren van v waarvan de rechterpunten verder liggen dan die van v, dan zien we dat deze een kliek vormen. Het sorteren van intervallen op basis van de rechterpunten van de intervallen, resulteert dus in een PEO. Om dit duidelijker te maken, illustreren we dit in de volgende figuur, waarin knopen w_1, w_2 en w_3 buren van v zijn, waarvoor geldt $rechts(w_i) > rechts(v)$ voor $i = 1, 2, 3$:



Figuur 12

De PEO-sortering blijkt een goed werkbare veralgemenisering van de sortering op rechterpunten en zullen we gebruiken in ons algoritme. Door middel van lexicografische breadth-first-search kan de PEO in lineaire tijd worden bepaald [21]. Goed om hierbij op te merken is dat de PEO van een graaf niet uniek is. Deze is namelijk afhankelijk van bij welke knoop men begint. Algoritme 6 beschrijft het algoritme. We zullen ook een voorbeeld behandelen om inzicht te krijgen in hoe het werkt.

Als we voor de graaf in Figuur 13 een PEO willen bepalen, initialiseren we dus eerst een verzameling $X = \{V\}$. We kiezen vervolgens een willekeurige startknoop, zeg knoop f. De buren van f zijn c en g, dus krijgen we $X = \{\{c, g\}, \{a, b, d, e\}\}$. Daarna beginnen we weer bij stap 2 en kiezen we c of g. We nemen c, die buren a, b, d en g heeft. Verzameling X ziet er dan als volgt uit: $X = \{\{g\}, \{d\}, \{a, b, e\}\}$. Het blijven herhalen van deze stappen levert uiteindelijke PEO $\{e, b, a, d, g, c, f\}$ op.

Zij $G = (V, E)$ een chordale graaf.

Stap 1: Definieer een verzameling $X = \{V\}$, die één element bevat: de hele verzameling knopen.

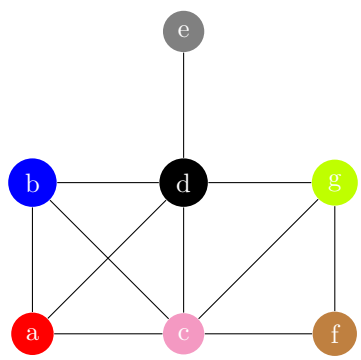
Stap 2: Kies een knoop v uit de eerste verzameling in X die nog niet gekozen knopen bevat en verwijder v uit X.

Stap 3: Verdeel iedere verzameling $S \in X$ in twee deelverzamelingen: de buren van v in S en de niet-buren van v in S.

Stap 4: Herhaal stap 2 en stap 3 tot X leeg is.

Stap 5: Output PEO = de omgekeerde volgorde van de gekozen knopen.

Algoritme 6: Bepalen van een PEO in een chordale graaf (Rose et al.)



Figuur 13: Chordale graaf

6.2 Bepalen van de optimale bladverzameling

Aan de hand van de PEO-sortering kunnen we dezelfde techniek die we gebruikten bij de intervalgrafen toepassen om de optimale bladverzameling B te bepalen. Het resultaat is te bekijken in Algoritme 7. Om te bewijzen dat de bladverzameling die we bepalen met Algoritme 7 de grootst mogelijke bladverzameling is, noteren we wederom $N(v)$ van een knoop v voor alle buren van v in G. Verder noemen we $N^-(v)$ en $N^+(v)$ de verzamelingen buren van v die respectievelijk voor en na v voorkomen in de PEO.

Propositie 10. Zij $G = (V, E)$ een chordale graaf waarbij $V = \{v_1, \ldots, v_n\}$ gesorteerd is op PEO en $B = \{b_1, \ldots, b_k\}$ de door Algoritme 7 bepaalde bladverzameling. Dan heeft G geen independency tree met meer dan k bladeren.

Bewijs. We maken gebruik van inductie over $|V|$. Als $|V| = 3$, dan is G een driehoek en bevat het geen independency tree of het bevat twee kanten en heeft het twee knopen van graad 1 en één knoop van graad 2. Het algoritme vindt dan de twee knopen van graad 1 als bladeren, wat de optimale bladverzameling is.

Nu nemen we aan dat er een optimale bladverzameling O bestaat voor de knopen $\{v_1, \ldots, v_i\} \subset V$ voor een bepaalde $3 \leq i < n$, die overeenkomt met de door het algoritme bepaalde B. Vervolgens bekijken we de optimale bladverzameling O voor $\{v_1, \ldots, v_{i+1}\}$. Daarvoor onderscheiden we twee gevallen: $v_{i+1} \notin B$ en $v_{i+1} \in B$.

Als $v_{i+1} \notin B$, dan is v_{i+1} een cutvertex van G of er is een $v \in N^-(v_{i+1})$ die al is toegevoegd aan B. Als v_{i+1} een cutvertex is, kan het geen blad zijn van een opspannende boom en geldt dus ook $v_{i+1} \notin O$.

Zij $G = (V, E)$ een chordale graaf.

Stap 1: Gebruik Algoritme 6 om een PEO $v_1, \ldots, v_n$ van de knopen in V te bepalen.

Stap 2: Bepaal welke knopen graad 1 hebben en voeg deze toe aan een lege verzameling J.

Stap 3: Voeg de overige knopen van V, achter de al toegevoegde knopen, op volgorde van de PEO toe aan J. Deze verzameling noemen we J_0.

Stap 4: Neem x = de eerste knoop in J.
Als $G' = (V - \{x\}, E|_{V-\{x\}})$ samenhangend is: voeg x toe aan bladverzameling B en verwijder x en alle buren van x uit J.

Als G' niet samenhangend is: verwijder x uit J.

Stap 5: Herhaal stap 4 totdat J leeg is.

Stap 6: Output B.

Algoritme 7: Het bepalen van optimale bladverzameling B voor een chordale graaf

Als er al een $v \in N^-(v_{i+1})$ is toegevoegd aan B, is v volgens de inductiehypothese ook een element van O. Dan geldt dus ook $v_{i+1} \notin O$.

Als $v_{i+1} \in B$, dan geldt dat v_{i+1} geen cutvertex is en $N^-(v_{i+1}) \cap B = \emptyset$. Volgens de inductiehypothese geldt dan ook dat $N^-(v_{i+1}) \cap O = \emptyset$. Als $v_{i+1} \in O$, dan zijn we klaar, dus laten we $v_{i+1} \notin O$. Dan moet er een $v \in N^+(v_{i+1})$ wél in O zitten, want anders is $O \cup \{v_{i+1}\}$ een grotere bladverzameling dan O. De knopen zijn gesorteerd op PEO, dus $N^+(v_{i+1})$ is een kliek. Er zit dan dus precies één element van $N^+(v_{i+1})$ in O. Dit stelt ons in staat om v in O te vervangen door v_{i+1}, wat ons een andere optimale bladverzameling geeft. □

6.3 Bladverzameling uitbreiden naar Max Leaf IT

Na het bepalen van de optimale bladverzameling B willen we de daadwerkelijke Max Leaf IT T vormen. Dit doen we in Algoritme 8. Hiervoor gebruiken we dezelfde strategie als bij de intervalgrafen: we bepalen aan de hand van Algoritme 1 een independency tree T' of Hamiltoncykel $T' + \{e\}$ in de graaf $G' = (V - B, E|_{V-B})$, waaraan we de bladeren uit B verbinden, zodat we de gewenste T verkrijgen. We passen de PEO-volgorde daarvoor iets aan. We gebruiken namelijk de in Algoritme 7 geïntroduceerde J_0, wat niets anders is dan een PEO, waarbij de knopen van graad 1 naar voren zijn gehaald. Voor de buren van een knoop v die voor of na v voorkomen in J_0 noteren we respectievelijk $N^-_{J_0}(v)$ en $N^+_{J_0}(v)$.

Wat ons rest, is het laten zien dat stap 4 van Algoritme 8 altijd kan worden uitgevoerd. Vergelijkbaar met ons bewijs voor de intervalgrafen gebruiken we het Theorema van Hall, om aan te tonen dat er een koppeling van de bladeren van independency tree T' bestaat in $H = (B \cup B', E_{|B \cup B'})$. Vervolgens laten we zien dat er wederom buren in Hamiltoncykel $T' + \{e\}$ bestaan die aan verschillende bladeren uit B verbonden zijn. We bekijken eerst het geval dat T' een independency tree is met bladverzameling B'. In Propositie 11 bewijzen we een eigenschap, die we later kunnen uitbreiden naar een bewijs voor het bestaan van de gewenste koppeling.

Propositie 11. Zij $G = (V, E)$ een chordale graaf, B de door Algoritme 7 bepaalde bladverzameling, T' een independency tree van $G' = (V - B, E|_{V-B})$ met bladverzameling B' en $J_0 = \{v_1, \ldots, v_n\}$ de eerder behandelde sortering van V. Dan geldt: voor iedere $b' \in B'$ bestaat er een $b \in B$ zodanig dat $b \in N^-_{J_0}(b')$.

Zij $G = (V, E)$ een chordale graaf.

Stap 1: Vind de optimale bladverzameling B en J_0 door middel van Algoritme 7.

Stap 2: Bepaal een willekeurige opspannende boom T' van $G' = (V - B, E|_{V-B})$.

Stap 3: Voer Algoritme 1 uit op T', zodat T' een independency tree of een Hamiltonpad bevat in een Hamiltoncykel van G' is.

Stap 4: Als T' een independency tree is: Voeg T' toe aan T en voeg vervolgens voor ieder blad van T' een kant toe naar een blad in B.

Als T' een Hamiltonpad, bevat in Hamiltoncykel $T' + \{e\}$ is: Bepaal het Hamiltonpad in $V - B$ waarbij de twee uiteinden beide een kant gemeenschappelijk hebben met twee verschillende bladeren in B en voeg het Hamiltonpad en deze kanten toe aan T.

Stap 5: Voeg kanten van de overgebleven bladeren in B naar T' toe aan T.

Stap 6: Output T.

Algoritme 8: Het bepalen van Max Leaf IT T in chordale graaf G

Bewijs. Zij $b' \in B'$ willekeurig. Er geldt dat $b' \notin B$, dus $deg(b') > 1$. Verder bestaat er vanwege de vormgeving van Algoritme 7 een $b \in B$ zodanig dat $b \in N^-_{J_0}(b')$ of b' is een cutvertex van G. We willen dus laten zien dat de stelling ook geldt als b' een cutvertex van G is. Neem hiervoor het omgekeerde aan: b' is een cutvertex van G en $N^-_{J_0}(b') \cap B = \emptyset$. Knoop b' is een blad van opspannende boom T' van G', dus b' is geen cutvertex van G'. Er is dus een blad $b \in B$ dat verbonden is met b', waarvoor $(b', b) \in E$ een cutedge is. Aangezien $N^-_{J_0}(b') \cap B = \emptyset$, geldt dat $b \in N^+_{J_0}(b')$.

Stel nu dat $b \in N(v)$ voor een $v \in V - \{b'\}$. Dan is (b', b) geen cutedge en b' dus geen cutvertex en hebben we een tegenspraak.

Blad b heeft dus graad 1 in G. Echter, dan bevindt b zich in J_0 voor b', want de graad van b' is minstens 2, wat een tegenspraak is met dat $b \in N^+_{J_0}(b')$. □

Dit is de cruciale eigenschap die we gebruiken om te laten zien dat er een koppeling van B' bestaat in de $H = (B \cup B', E_{|B \cup B'})$.

Propositie 12. Zij $G = (V, E)$ een chordale graaf, B de door Algoritme 7 bepaalde bladverzameling, T' een independency tree van $G' = (V - B, E|_{V-B})$ met bladverzameling B' en $J_0 = \{v_1, \ldots, v_n\}$ de eerder behandelde sortering van V. Dan bestaat er een koppeling van B' in bipartiete graaf $H = (B \cup B', E_{|B \cup B'})$.

Bewijs. Voor iedere $b' \in B'$ geldt dat $|N(b')| \geq 1$ in H. Dit volgt direct uit Propositie 11. Voor alle $S \subset B'$ met $|S| = 1$ geldt dus dat $|N(S)| \geq |S|$.

Zij nu $S = \{b'_1, \ldots, b'_k\} \subset B'$ voor $2 \leq k \leq |B'|$. Volgens Propositie 11 geldt dan voor iedere $b'_i \in S$ dat er een $b \in B$ bestaat zodat $b \in N^-_{J_0}(b'_i)$. We gaan laten zien dat voor iedere b'_i zelfs een unieke $b \in B$ bestaat met deze eigenschap.

Neem aan dat $b \in N^-_{J_0}(b'_i)$ en tegelijkertijd $b \in N^-_{J_0}(b'_j)$ voor $1 \leq i < j \leq k$. De graad van zowel b'_i als b'_j in G is minstens 2, aangezien het geen bladeren van T zijn. Blad b heeft buren b'_i en b'_j en heeft dus ook graad minstens 2. Knoop b komt in de PEO dus voor zowel b'_i als b'_j, dus de drie knopen vormen een kliek. In het bijzonder geldt: $(b'_i, b'_j) \in E$, dus T' is geen independency tree en we hebben een tegenspraak.

Er geldt dus dat voor iedere $b'_i \in S$ een unieke $b_j \in N^-_{J_0}(b'_i)$ bestaat, dus er geldt dat $|N(S)| \geq |S|$ voor alle $S \subset B'$. □

Nu we het geval dat T' een independency tree is, hebben behandeld, gaan we verder met het geval dat T' een Hamiltonpad is, bevat in Hamiltoncykel $T' + \{e\}$. Daarvoor laten we zien dat we altijd twee buren in de Hamiltoncykel kunnen nemen als uiteinden van een Hamiltonpad. Deze moeten we dan verbinden met twee verschillende bladeren uit B, anders is één van de uiteinden een blad van onze uiteindelijke T. In de onderstaande propositie bewijzen we dat we altijd twee buren kunnen nemen, die verbonden zijn met verschillende bladeren.

Propositie 13. Zij $G = (V, E)$ een chordale graaf, B de gevonden bladverzameling, T' een Hamiltonpad van $G' = (V - B, E|_{V-B})$ bevat in Hamiltoncykel $T' + \{e\}$. Dan bestaan er $v_1, v_2 \in V - B$ en $b_1, b_2 \in B$ met $b_1 \neq b_2$ zodat $(v_1, b_1), (v_2, b_2) \in E$ en $(v_1, v_2) \in T' + \{e\}$ voor $v_1 \neq v_2$ en $b_1 \neq b_2$.

Bewijs. Dit bewijs is hetzelfde als het bewijs voor Propositie 9 en daarom weggelaten. □

6.4 Complexiteitsanalyse

Theorema 4. Zij $G = (V, E)$ een chordale graaf. Dan is het probleem Max Leaf Independency Tree oplosbaar op G in $O(nm)$ tijd.

Bewijs. De correctheid van Algoritme 7 volgt uit Propositie 10. Daarnaast volgt de correctheid van Algoritme 8 uit Propositie 11, Propositie 12 en Propositie 13.

Voor Algoritme 7 en Algoritme 8 kunnen we dezelfde implementatie gebruiken als voor Algoritme 4 en Algoritme 5. Het enige verschil is dat we de knopen nu sorteren aan de hand van een PEO in plaats van op basis van de rechterpunten van de intervallen. We zagen echter dat het bepalen van een PEO mogelijk is in lineaire tijd, dus dit verslechtert de looptijd niet. Zowel Algoritme 7 als Algoritme 8 heeft dus een looptijd van $O(nm)$. □

7 Conclusie

In deze scriptie hebben we independency trees behandeld door eerder gedane onderzoeken te bestuderen. Daarna zijn we op zoek gegaan naar algoritmen voor het Max Leaf Independency Tree-probleem op verschillende graafklasssen.

Voor co-bipartiete grafen bepaalden we een $O(|U| \cdot |V|)$-algoritme dat een brug tussen U en V vindt en deze verbindt met Hamiltonpaden in U en in V. Dit levert een independency tree met twee bladeren op, wat het maximale aantal is in een co-bipartiete graaf.

Het algoritme dat we opstelden voor splitgrafen met partitie $\{U, V\}$ voor een onafhankelijke verzameling knopen U en een kliek V heeft een lineaire looptijd. We bepalen een Hamiltonpad in V om vervolgens aan de uiteinden knopen in U te verbinden en ten slotte alle overige knopen uit U eraan toe te voegen. Als V een knoop bevat zonder brug naar U, vinden we zo een independency tree met $|U| + 1$ bladeren. Indien een dergelijke knoop niet bestaat, bevat de independency tree $|U|$ bladeren.

Vervolgens stelden we een algoritme samen voor intervalgrafen, dat we konden veralgemeniseren tot een algoritme voor chordale grafen. Ten eerste bepalen we de optimale bladverzameling door middel van een greedy algoritme. Het deel van de graaf dat zich niet in de bladverzameling bevindt, bouwen we dan om naar een independency tree of een Hamiltoncykel. De bladeren van deze independency tree of twee buren van de Hamiltoncykel verbinden we daarna met bladeren uit de bladverzameling, waarna we de overige bladeren willekeurig kunnen toevoegen. Deze algoritmen voor de intervalgrafen en chordale grafen hebben een looptijd van $O(nm)$.

Ons onderzoek zou nog verder kunnen worden uitgebreid. Ten eerste zouden we meer graafklassen kunnen bekijken en kunnen onderzoeken of vergelijkbare strategieën ook algoritmen voor deze graafklassen met een polynomiaal begrensde looptijd opleveren.

De gevonden algoritmen voor de intervalgrafen en de chordale grafen zijn daarnaast als het ware uitbreidingen van algoritmen met polynomiale looptijd voor Maximum Independent Set. Interessant zou zijn om te onderzoeken of voor iedere graafklasse waarvoor een algoritme met polynomiale looptijd voor Maximum Independent Set bestaat ook Max Leaf Independency Tree in polynomiale tijd oplosbaar is. Daarnaast zou kunnen worden onderzocht of een vergelijkbare aanpak werkt voor Min Leaf Independency Tree.

8 Bibliografie

[1] M. R. Garey, D. S. Johnson (1979). NP-Complete Problems. In *Computers and intractability: a guide to the theory of NP-completeness* (pp. 206). New York: W. H. Freeman and Company.

[2] M.S. Rahman, M. Kaykobad (2005). Complexities of some interesting problems on spanning trees. *Information Processing Letters, 94*, pp. 93–97.

[3] H. Fernau, J. Kneis, D. Kratsch, A. Langer, M. Liedloff, D. Raible, P. Rossmanith (2011). An exact algorithm for the Maximum Leaf Spanning Tree problem. *Theoretical Computer Science, 412*, pp. 6290-6302.

[4] K. Casel, J. Dreier, H. Fernau, M. Gobbert, P. Kuinke, F. S. Villaamil, M. L. Schmid, E. J. van Leeuwen (2018). Complexity of Independency and Cliquy Trees. *Discrete Applied Mathematics.* `https://doi.org/10.1016/j.dam.2018.08.011`.

[5] B. K. Bhattacharya, M. De, S. C. Nandy, S. Roy (2014). Maximum Independent Set for Interval Graphs and Trees in Space Efficient Models. *Canadian Conference on Computational Geometry.* Verkregen via: `http://www.cccg.ca/proceedings/2014/papers/paper33.pdf`.

[6] A. A. Diwan (2010). *Algorithmic problems on chordal graphs* [Lecture Notes]. Verkregen via `https://www.cse.iitb.ac.in/~aad/cs602/algorithmic_problems_chordal.pdf`.

[7] T. Böhme, H. Broersma, F. Göbel, A. V. Kostochka, M. Stiebitz (1996). Spanning trees with pairwise nonadjacent endvertices. *Discrete Mathematics, 170*(1-3), pp. 219-222.

[8] D. Eppstein, V. Le (2013). Graphs in which every spanning tree is an independency tree [Internet forum "mathoverflow"]. `https://mathoverflow.net/questions/141355/graphs-in-which-every-spanning-tree-is-an-independency-tree`.

[9] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, S. Saurabh (2016). Kernelization. In *Parameterized Algorithms* (pp. 17-39). Springer.

[10] J. Flum, M. Grohe (2006). Fixed-Parameter Tractability. In *Parameterized Complexity Theory* (pp. 1-30). Springer.

[11] W. Li, Y. Cao, J. Chen, J. Wang (2017). Deeper local search for parameterized and approximation algorithms for maximum internal spanning tree. *Information and Computation, 252*, pp. 187-200.

[12] J. Kleinberg, É. Tardos (2005). Graphs. In *Algorithm Design* (pp. 94-97). Pearson.

[13] P. L. Hammer, B. Simeone (1981). The splittance of a graph. *Combinatorica, 1(3)*, pp. 275-284.

[14] P. Heggernes, D. Kratsch (2006). Linear-time certifying algorithms for recognizing split graphs and related graph classes. *Reports in Informatics, 328*, pp. 1-21.

[15] M. C. Golumbic (1980). Split Graphs. In *Algorithmic Graph Theory and Perfect Graphs* (pp. 149-156). New York: Academic Press.

[16] R. Diestel (2000). Matching, Covering and Packing. In *Graph Theory* (pp. 31-33). New York: Springer-Verlag.

[17] D. G. Corneil, S. Olariu, L. Stewart (2009). The LBFS Structure and Recognition of Interval Graphs. *SIAM Journal on Discrete Mathematics, 23(4)*, pp. 1905-1953.

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein (2009). Maximum bipartite matching. In *Introduction to Algorithms (Third Edtion)* (pp. 732-736). The MIT Press.

[19] G. Hajos (1957). Über eine Art von Graphen. *Internationale Mathematische Nachrichten, 11.*

[20] G. Valiente (2001). *Advanced Graph Algorithms* [Lecture Notes]. Verkregen via `http://www.lsi.upc.es/~valiente/graph-00-01-e.pdf`.

[21] D. Rose, G. Lueker, R. E. Tarjan (1976). Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing, 5*, pp. 266–283.