Auteursrechterlijke overeenkomst

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

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

Titel: Algoritmen voor networkflow
Richting: 2de masterjaar in de informatica - databases Jaar: 2009

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

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

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

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

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

Ik ga akkoord,

GRAUWELS, Michiel

Datum: 14.12.2009

Algoritmen voor networkflow

Michiel Grauwels

promotor :
dr. Dries VAN DYCK

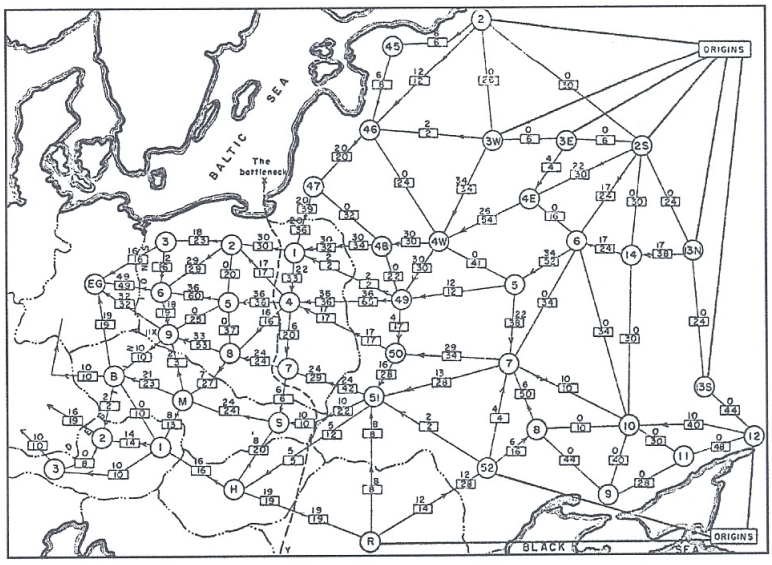
Eindverhandeling voorgedragen tot het bekomen van de graad
master in de informatica databases

universiteit hasselt

universiteit hasselt
Departement Wiskunde, Natuurkunde, Informatica
Vakgroep Informatica

Algoritmen voor networkflow

Prestroom-duw-algoritmen



door Michiel Grauwels
Promotor: dr. Dries Van Dyck
Begeleider: dr. Dries Van Dyck

MASTERPROEF VOORGEDRAGEN TOT HET BEHALEN VAN DE GRAAD VAN MASTER (2-JARIG) IN DE INFORMATICA/ICT/KENNISTECHNOLOGIE

Academiejaren 2007-2009

Inhoudstafel

Dankwoord

Geachte lezer,

U staat op het punt mijn masterproef te lezen. Het voltooien van deze masterproef zou niet mogelijk geweest zijn zonder de hulp, steun en aanmoediging van enkele mensen.

Allereerst gaat mijn dank uit naar mijn promotor en begeleider dr. Dries Van Dyck, voor het op regelmatige basis geven van de nodige feedback en extra informatie in verband met het onderwerp van deze masterproef en soms zelfs enkele minder relevante, doch interessante zaken.

Vervolgens wil ik ook mijn medestudenten aan de Universiteit Hasselt bedanken en in het bijzonder Jonny Daenen voor het aanbrengen van de nodige kennis in verband met het tekenen van grafen in LaTeX.

Als laatste wil ik een zeer bijzonder woord van dank richten aan mijn ouders, familie, vrienden en kennissen. Mijn ouders wil ik graag bedanken voor de kans die zij mij geboden hebben om mijn universitaire studies aan te vatten. Mijn vriendin Charlotte wist op gepaste tijden steeds voor de nodige ontspanning en afleiding te zorgen en haar ouders, Luc en Brigitte, gaven mij, samen met Charlotte, de kans om ook bij hun thuis aan deze masterproef te werken.

Bedankt voor het begrip, het geduld en de vele mogelijkheden!

Michiel Grauwels
10 juni 2009

Hoofdstuk 1

Inleiding

Hoeveel verkeer kan er passeren doorheen een drukke stad tijdens de spitsuren? Hoeveel olie kan er doorheen een gegeven netwerk van pijpleidingen stromen op een dag? Hoe kan een fruitboer uit Spanje zoveel mogelijk ton sinaasappelen verzenden per vrachtwagen vanuit zijn bedrijf via doorgeefluiken naar verscheidene andere bedrijven in bijvoorbeeld Rusland en dit binnen één week tijd? Elk van deze vragen komt neer op het probleem om zoveel mogelijk artikelen, grondstoffen, producten, ... van het ene station naar het andere binnen een gegeven tijd en eventueel langs allerlei tussenstations te sturen of te laten *stromen*. Al de stations samen vormen een netwerk met een begin- en eindstation. Er loopt hierbij een hoeveelheid stroom per tijdseenheid van het beginstation naar het eindstation. De hoeveelheid stroom per tijdseenheid doorheen het netwerk wordt begrensd door de capaciteiten van de verbindingen die het netwerk zelf bepalen. Het gestelde probleem oplossen, komt er dus op neer om de maximale hoeveelheid stroom te zoeken per tijdseenheid doorheen een dergelijk netwerk en dit zonder de begrenzingen, opgelegd door de verbindingen, te verbreken.

We noemen het gestelde probleem het maximumstroomprobleem en er bestaat een zeer efficiënte oplossingsmethode voor dit probleem, nl. de prestroom-duw-methode van Goldberg en Tarjan. Uit deze methode zijn vervolgens specifieke prestroom-duw-algoritmen ontstaan.

Deze masterproef is als volgt opgebouwd. Na dit inleidende hoofdstuk, introduceren we in Hoofdstuk 2, gebaseerd op [13] en [6], het wiskundige begrip graaf en bekijken enkele relevante begrippen en eigenschappen. Een netwerk waarop het maximumstroomprobleem wordt toegepast, stellen we immers voor met behulp van een type graaf, nl. een gerichte graaf.

Er moet echter stroom kunnen lopen doorheen een dergelijk netwerk. We breiden het daarom vervolgens, in Hoofdstuk 3, gebaseerd op [6] en [11], uit naar een stroomnetwerk. Dit is een netwerk dat in staat is om een stroom te beheren en er vervolgens een maximale stroom doorheen te zoeken, hetgeen ons brengt bij het maximumstroomprobleem, beschreven

in Hoofdstuk 4 en dat gebaseerd is op [6] en [11]. We bekijken eveneens concrete toepassingen uit de praktijk waar het maximumstroomprobleem natuurlijk in naar voorkomt.

De uitleg van de prestroom-duw-methode zelf wordt, in Hoofdstuk 5, gebaseerd op [6], [4] en [11], ingeleid door de studie van de eerste "redelijke" oplossingsmethode voor het maximumstroomprobleem, de toenemende-pad-methode van Ford-Fulkerson. Deze methode werd gebruikt als basis voor tal van andere oplossingsmethoden, maar zijn nadelen gaven uiteindelijk voer voor de ontwikkeling van de veel efficiëntere prestroom-duw-methode. We bespreken deze methode uitvoerig in Hoofdstuk 6, gebaseerd op [6] en [11], samen met enkele specifieke implementaties van deze methode, waarnaar we zullen verwijzen als prestroom-duw-algoritmen. Vervolgens bekijken we, in Hoofdstuk 7 heuristieken die de uitvoering van deze algoritmen in het algemeen versnellen.

Met behulp van de vergaarde theoretische kennis, bouwden we een applicatie die ons toelaat met de bestudeerde prestroom-duw-algoritmen in uitvoering, al dan niet in combinatie met bepaalde heuristieken, te experimenteren en/of hun uitvoering in 3D te visualiseren.

In Hoofdstuk 8 volgt een beschrijving van de resultaten van enkele experimenten die we hebben opgezet met betrekking tot de bestudeerde prestroom-duw-algoritmen en hun heuristieken.

De implementatie en werking van de applicatie wordt besproken in Hoofdstuk 10. Daarenboven geven we in dit hoofdstuk ook voorbeelden van stroomnetwerken waarmee we bepaalde effecten met betrekking tot prestroom-duw-algoritmen, beschreven doorheen de masterproef, met de applicatie, kunnen visualiseren.

Indien er nog andere referenties geraadpleegd werden, wordt dit expliciet aangegeven in de tekst op de plaats van gebruik.

Figuren 5.6, 6.6 en 6.7 zijn afkomstig uit [4], [10] en [5]. Alle andere figuren zijn gebaseerd op [11] en [6] of van eigen makelij.

Hoofdstuk 2

Relevante grafentheorie

Zoals reeds gezegd in Hoofdstuk 1 is het stroomnetwerk, waarop we het maximumstroomprobleem loslaten, een gerichte graaf. We willen de stroom doorheen deze graaf optimaliseren, het is dus een *graafoptimalisatieprobleem*. Het is dus noodzakelijk dat we eerst een kijkje nemen in de wereld van de grafentheorie om vertrouwd te raken met het begrip graaf en zijn eigenschappen. We bekijken dit laatste enkel binnen het kader van het probleem omdat een volledige studie ons te ver zou leiden.

2.1 Definitie graaf

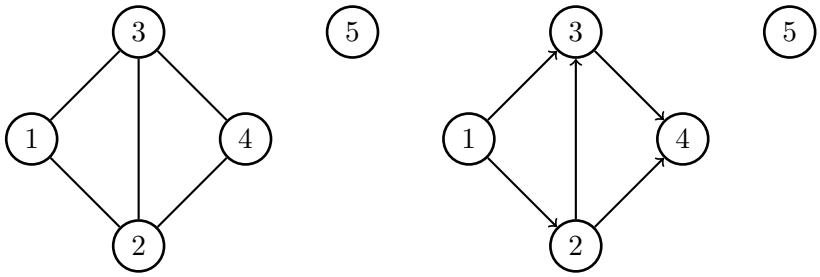
Een *graaf* $G = (V, E)$ bestaat uit een eindige verzameling *knopen* (vertices) V en een eindige verzameling *bogen* (edges) E. Deze bogen worden gevormd door koppels van knopen zodanig dat $E \subseteq V \times V$.

Indien de koppels in E ongeordend zijn, zeggen we dat E *symmetrisch* is en dat de bogen van G *ongericht* zijn. We noteren een boog van G dan als een ongeordend koppel van knopen $\{u, v\}$, met $u, v \in V$. We noemen G bijgevolg een *ongerichte graaf*. In Figuur 2.1(a) zien we een voorbeeld van een ongerichte graaf $G_1 = (V, E_1)$ met $V = \{1, 2, 3, 4, 5\}$ en $E_1 = \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\}$.

Wanneer de koppels in E wél geordend zijn, zeggen we dat E *asymmetrisch* is en dat de bogen van G *gericht* zijn. Deze bogen worden dan gevormd door geordende koppels van knopen (u, v), met $u, v \in V$. We noemen G bijgevolg een *gerichte graaf* of *digraaf*[1]. In Figuur 2.1(b) zien we een voorbeeld van een gerichte graaf $G_2 = (V, E_2)$ met $V = \{1, 2, 3, 4, 5\}$ en $E_2 = \{(1, 2), (1, 3), (2, 3), (2, 4), (3, 4)\}$.

We beschouwen de termen graaf en *netwerk* als synoniemen van elkaar en stellen, in het verdere verloop van dit werk, een netwerk voor als een digraaf G. We gebruiken steeds n voor het aantal knopen en m voor het aantal bogen van G. We definiëren in het verdere verloop van dit hoofdstuk

[1] directed graph



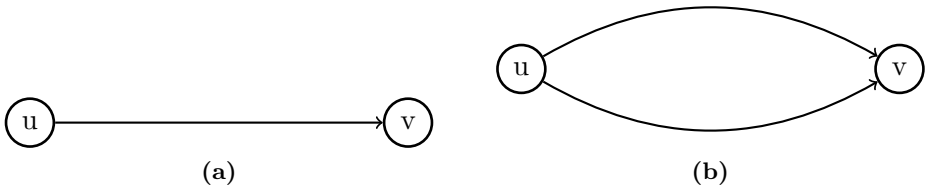
(a) ongerichte graaf G_1 **(b)** gerichte graaf G_2

Figuur 2.1: Illustratie bij de definitie van een graaf met in (a) de tekening van een ongerichte graaf G_1 en in (b) de tekening van een gerichte graaf of digraaf G_2.

enkele relevante eigenschappen met betrekking tot G en gebruiken een dergelijk netwerk als basis voor de formele definitie van een stroomnetwerk in Hoofdstuk 3.

2.2 Kop en staart

De twee knopen u en v die, als geordend koppel, samen een gerichte boog (u, v) vormen, noemen we de *staart* en resp. de *kop* van boog (u, v), zie Figuur 2.2(a) ter illustratie. Deze boog is een *uitgaande boog* voor de staart u en een *inkomende boog* voor de kop v.



Figuur 2.2: Illustratie van kop en staart, uitgaande- en inkomende boog in (a) en de illustratie van multibogen in (b).

Twee of meer bogen die opgebouwd zijn uit dezelfde staart en dezelfde kop, noemen we *multibogen*, zie Figuur 2.2(b) ter illustratie. We merken op dat geen enkele boog in E een multiboog is aangezien E geen multiverzameling[2] is van $V \times V$.

Een boog waarvan staart en kop dezelfde knoop voorstellen, noemen we een *lus*, zie Figuur 2.3 ter illustratie.

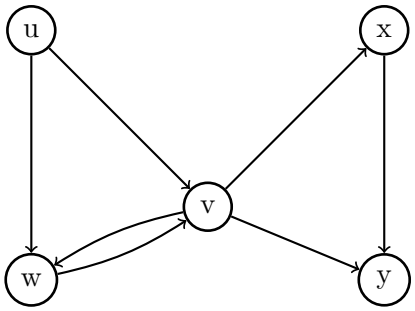
[2]Een multiverzameling is een verzameling waarvan de elementen meer dan één keer mogen voorkomen.

Figuur 2.3: Illustratie van een lus.

2.3 Graad

We definiëren het *aantal inkomende bogen van een knoop* v als $deg^+(v)$ en het *aantal uitgaande bogen van een knoop* v als $deg^-(v)$. De *graad van een knoop* v definiëren we als volgt: $deg(v) = deg^+(v) + deg^-(v)$. Het is verder heel makkelijk in te zien dat het aantal bogen van een graaf, m, gelijk is aan de som van het aantal inkomende bogen van alle knopen van de graaf en tevens gelijk is aan de som van het aantal uitgaande bogen van alle knopen van de graaf. De *graad van een graaf* zelf is gelijk aan $max\{deg(v) : v \in V\}$, met andere woorden de maximale graad onder zijn knopen.

Wanneer we dit principe toepassen op graaf G_3 uit Figuur 2.4, geldt er voor knoop v: $deg^+(v) = 2$, $deg^-(v) = 3$ en $deg(v) = 5$. De graad van G_3 is gelijk aan $max\{2, 3, 5, 2, 2\} = 5$.



Figuur 2.4: Illustratie van graad.

2.4 Adjacentielijst

Wanneer een boog (u, v) bestaat, dan zeggen we dat knoop v een *buur* is van knoop u. De *adjacentielijst* $A(u) = \{v \in V : (u, v) \in E\}$ stelt de verzameling knopen voor die buren zijn van knoop u. Met een dergelijke lijst, voor elke knoop, wordt het mogelijk om de topologische structuur van een graaf te representeren.

Ter illustratie geven we even de adjacentielijsten voor de knopen van

graaf G_3 uit Figuur 2.4: $A(u) = \{v, w\}$; $A(v) = \{w, x, y\}$; $A(w) = \{v\}$; $A(x) = \{y\}$; $A(y) = \emptyset$.

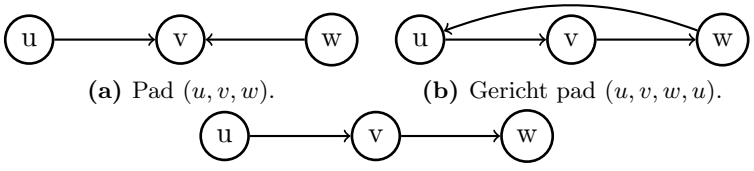
2.5 Adjacentiematrix

De topologische structuur van een graaf G kan ook gerepresenteerd worden met behulp van een matrix, $A(G)$, over de verzameling $\{0, 1\}$. We noemen een dergelijke matrix de *adjacentiematrix* van G. De rijen en kolommen van $A(G)$ zijn geïndexeerd door de knopen van G. Op die manier bestaat de matrix uit $n \times n$ cellen a_{uv} waarvan de waarde aangeeft of v een buur is van u. Indien v een buur of geen buur is van u geldt er dat $a_{uv} = 1$, resp. $a_{uv} = 0$.

2.6 Paden

Een *wandeling* is een opeenvolging van knopen $(v_0, \ldots, v_n)$ zodat $(v_i, v_{i+1}) \in E$ of $(v_{i+1}, v_i) \in E$ met $0 \leq i < n$. De knoop v_0 noemen de *startknoop*, v_n de *eindknoop* en v_i, $0 < i < n$, *interne* knopen van de wandeling.

Een *pad* is een wandeling zodat geen interne knoop herhaalt. Indien het pad voor een boog (u, v) eerst zijn staart u bezoekt en daarna zijn kop v, dan noemen we deze boog een *voorwaartse boog* van het pad. Wanneer het pad hierbij eerst zijn kop v bezoekt en daarna pas zijn staart u, spreken we van een *achterwaartse boog* van het pad. In Figuur 2.5(a) zien we het pad (u, v, w), waarbij boog (u, v) een voorwaartse boog en (w, v) een achterwaartse boog is van het pad.



(a) Pad (u, v, w). **(b)** Gericht pad (u, v, w, u).

(c) Simpel pad (u, v, w).

Figuur 2.5: Illustratie bij de definitie van een pad, gericht pad, cykel en simpel pad.

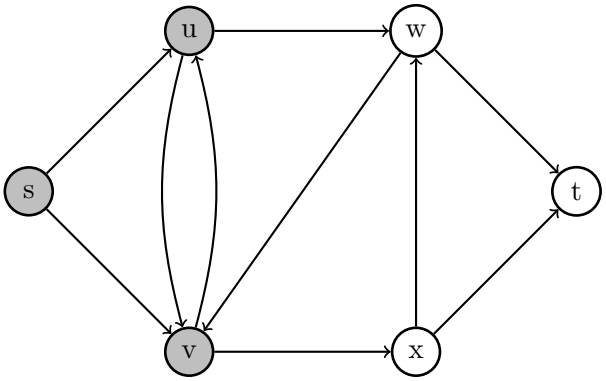
Indien het pad enkel is opgebouwd uit voorwaartse bogen, dan noemen we dit pad een *gericht pad*, zie Figuur 2.5(b) ter illustratie. Een *cykel* is een pad waarvan start- en eindknoop gelijk zijn. Een gericht pad dat geen cykel is, noemen we een *simpel pad*, zie Figuur 2.5(c) ter illustratie. Een simpel pad van u naar v noteren we als volgt: $u \to v$. Een *acyclische* graaf is een graaf die enkel simpele paden bevat.

2.7 Boogsnede

Een *boogsnede* $[S, \bar{S}]$, met $S \subset V$ en $\bar{S} = V \setminus S$, bestaat uit de verzameling bogen $\{(u, v) \in E \mid (u \in S \wedge v \in \bar{S}) \vee (u \in \bar{S} \wedge v \in S)\}$.

Een boog (u, v) met $u \in S$ en $v \in \bar{S}$ noemen we een *voorwaartse boog* van $[S, \bar{S}]$. De *verzameling voorwaartse bogen* van $[S, \bar{S}]$ noteren we als $(S, \bar{S})$. Een boog (u, v) met $v \in S$ en $u \in \bar{S}$ noemen we een *achterwaartse boog* van $[S, \bar{S}]$. De *verzameling achterwaartse bogen* van $[S, \bar{S}]$ noteren we als $(\bar{S}, S)$.

In Figuur 2.6 zien we een voorbeeld van een boogsnede $[S, T]$, met $S = \{s, u, v\}$ (op de figuur aangegeven in het grijs) en $T = \{w, x, t\}$. Wanneer we de verzameling voorwaartse- en achterwaartse bogen berekenen, krijgen we het volgende: $(S, T) = \{(u, w), (v, x)\}$ en $(T, S) = \{(w, v)\}$.



Figuur 2.6: Illustratie bij de definitie van een boogsnede.

Hoofdstuk 3

Stroomnetwerken

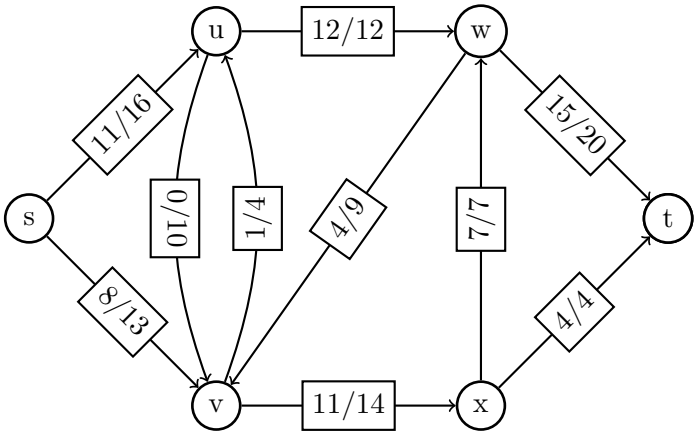
We stellen een *stroomnetwerk* formeel voor als een speciale vorm van een netwerk G, waarbij E géén lussen bevat. De *bron* en de *afvoer* van G zijn beide knopen van het netwerk en definiëren we als s, met $deg^+(s) = 0$, en resp. t, met $deg^-(t) = 0$. We veronderstellen dat elke knoop $v \in V$ op een simpel pad ligt van s naar t en bijgevolg is $m \geq n - 1$. Indien m naar het maximaal aantal bogen in het stroomnetwerk toegaat, spreken we van een *dicht* stroomnetwerk. Met andere woorden, indien $m \approx n^2$, dan heeft het stroomnetwerk een zéér hoge *dichtheid*. Alle knopen $v \in V \setminus \{s, t\}$, die dus gelegen zijn tussen s en t, noemen we *intermediaire* knopen.

Elke boog $(u, v) \in E$ heeft een *capaciteit* $c(u, v) > 0$ met c de *capaciteitsfunctie* $c : V \times V \to [0, +\infty[$. Voor elke boog $(u, v) \in E$ waarvoor er geen boog $(v, u) \in E$ bestaat, voegen we de boog (v, u) toe aan E en zetten $c(v, u) = 0$. Merk op dat we bogen met een capaciteit gelijk aan nul niet laten bijdragen tot het totaal aantal bogen m van het stroomnetwerk. We weten reeds dat E géén multibogen bevat. Multibogen in een stroomnetwerk toestaan heeft geen nut en maakt het stroomnetwerk onnodig complex. We kunnen multibogen immers samenvoegen tot één boog en zijn capaciteit gelijkstellen aan de som van de capaciteiten van de multibogen waaruit de boog werd samengesteld. Verder definiëren we de maximale capaciteit van een stroomnetwerk als $c_{max} = max\{c(u, v) : (u, v) \in E\}$.

Over elke boog $(u, v) \in E$ loopt er een *stroom*, en dus van knoop u naar knoop v. We definiëren deze als $f(u, v)$ met f de *stroomfunctie* $f : V \times V \to]-\infty, +\infty[$. f is de representatie voor de hele stroom van s naar t doorheen G. We noemen stroom f *geldig* indien hij voldoet aan de volgende drie eigenschappen:

1. *beperking op de capaciteit*,
2. *het annuleringsprincipe* en
3. *behoud van stroom*.

In Figuur 3.1 zien we een voorbeeld van een stroomnetwerk. De labels op de bogen van de graaf interpreteren we als volgt: het getal voor de schuine streep komt overeen met de hoeveelheid stroom over de boog en het getal achter de schuine streep stelt de capaciteit voor van de boog. We zien deze notatie als een manier om uit te drukken hoe de stroom over een boog zich verhoudt ten opzichte van de capaciteit van deze boog.



Figuur 3.1: Illustratie van een stroomnetwerk.

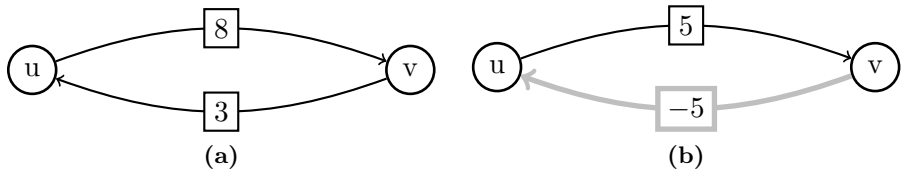
3.1 Eigenschappen van de stroomfunctie

Eigenschap 3.1 (Beperking op de capaciteit)**.** *De stroom over een boog mag de capaciteit van de boog niet overschrijden. We drukken dit formeel uit als volgt:*

$$\forall u, v \in V : f(u, v) \leq c(u, v). \tag{3.1}$$

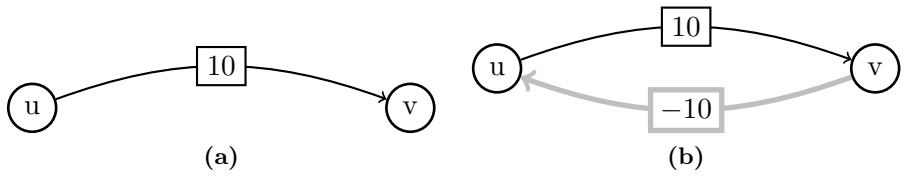
Een boog (u, v) waarvan de stroom $f(u, v)$ gelijk is aan zijn capaciteit $c(u, v)$ noemen we *verzadigd*. In het andere geval noemen we de boog *onverzadigd*.

Wanneer we naar de situatie in Figuur 3.2(a) kijken, dan zien we dat $f(u, v) = 8$ en $f(v, u) = 3$. Als we de vergelijking maken met een netwerk van pijpen waardoor water stroomt, dan wordt er 8 liter water per seconde van u naar v gezonden en 3 liter water per seconde van v naar u. Het is intuïtief heel makkelijk in te zien dat hetzelfde effect kan bekomen worden door enkel het zenden van 5 liter water per seconde van u naar v. Deze 5 liter bekom je door de 3 liter in de ene richting af te trekken van de 8 liter in de andere richting. Je kan stellen dat 3 van de 8 liter water per seconde tussen u en v geannuleerd wordt door 3 liter water per seconde van v naar u. We passen dit principe ook omgekeerd toe en bekomen dan de situatie zoals geïllustreerd in Figuur 3.2(b).



Figuur 3.2: Illustratie van de annulering van positieve stromen ten opzichte van elkaar. (a) Een positieve stroom van knoop u naar knoop v: $f(u, v) = 8$ en een positieve stroom van knoop v naar knoop u: $f(v, u) = 3$. (b) Het resultaat van annulering van de positieve stromen, uit (a), ten opzichte van elkaar.

Een ander voorbeeld zien we in Figuur 3.3, waarbij de stroom van v naar u, in Figuur 3.3(a), gelijk is aan nul.



Figuur 3.3: Illustratie van de annulering van positieve stromen ten opzichte van elkaar. (a) Een positieve stroom van knoop u naar knoop v: $f(u, v) = 10$ en een positieve stroom van knoop v naar knoop u: $f(v, u) = 0$. (b) Het resultaat van annulering van de positieve stromen, uit (a), ten opzichte van elkaar.

Het principe, geïllustreerd in voorgaande voorbeelden, kunnen we formeel als volgt formuleren:

Eigenschap 3.2 (Het annuleringsprincipe)**.** *We kunnen positieve stromen in tegengestelde richting tussen twee knopen van het stroomnetwerk, annuleren ten opzichte van elkaar door deze onderling van elkaar af te trekken. De stromen in beide richtingen zijn bijgevolg elkaars tegengestelde en we drukken dit formeel uit als volgt:*

$$\forall u, v \in V : f(u, v) = -f(v, u). \tag{3.2}$$

We kunnen met andere woorden positieve tegengestelde stromen, tussen twee knopen van het stroomnetwerk, omvormen tot één strikt positieve stroom in één enkele richting. Verder is er géén stroom tussen knoop u en v, $f(u, v) = f(v, u) = 0$, enkel en alleen indien de stromen in beide richtingen gelijk zijn of voor beide bogen (u, v) en (v, u) geldt dat ze niet tot E behoren. In de secties die volgen, maken we regelmatig impliciet gebruik van het annuleringsprincipe, in het bijzonder bij het formuleren van de wet van behoud van stroom en bij de definitie van het begrip residuele capaciteit.

We grijpen later, in Sectie 3.3, terug naar het annuleringsprincipe bij het opstellen van vergelijking (3.9) en steunen hier ook op bij het opstellen van vergelijking (3.3) en (3.4).

Eigenschap 3.3 (Behoud van stroom)**.** *De totale stroom die een intermediaire knoop verlaat, is gelijk aan nul en we drukken dit formeel uit als volgt:*

$$\forall v \in V \setminus \{s,t\} : \sum_{u \in V} f(v,u) = 0. \tag{3.3}$$

De totale stroom die in een intermediaire knoop toekomt, is gelijk aan nul en we drukken dit formeel uit als volgt:

$$\forall v \in V \setminus \{s,t\} : \sum_{u \in V} f(u,v) = 0. \tag{3.4}$$

We definiëren de totale positieve stroom die in een knoop v toekomt als:

$$\sum_{\substack{u \in V \\ f(u,v)>0}} f(u,v). \tag{3.5}$$

We definiëren de totale positieve stroom die een knoop v verlaat als:

$$\sum_{\substack{u \in V \\ f(v,u)>0}} f(v,u). \tag{3.6}$$

Het verschil tussen (3.5) en (3.6) definiëren we als de totale stroom in knoop v en schrijven we formeel neer als volgt:

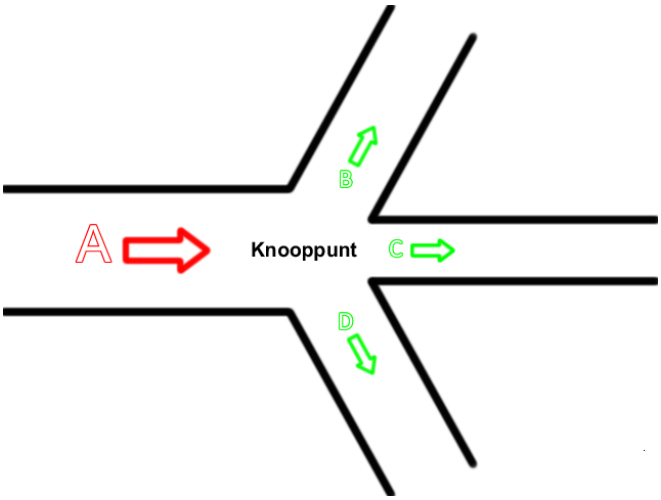
$$\sum_{\substack{u \in V \\ f(u,v)>0}} f(u,v) - \sum_{\substack{u \in V \\ f(v,u)>0}} f(v,u). \tag{3.7}$$

De totale stroom in elke intermediaire knoop v is gelijk aan nul indien er voldaan is aan (3.3) en (3.4).

De totale positieve stroom die in een intermediaire knoop toekomt is dus gelijk aan de totale positieve stroom die dezelfde intermediaire knoop verlaat. De totale stroom in een intermediaire knoop is met andere woorden steeds gelijk aan nul. Alles wat toekomt in een intermediaire knoop, moet er ook weer uit! Dit is naar analogie met de elektrische stroomwet van Kirchhoff [1] waarvan we een illustratie terugvinden in Figuur 3.4. Hierbij geldt er dat $A = B + C + D \Rightarrow (B + C + D) - A = 0$ en bijgevolg is er dus geen stroom aanwezig in het knooppunt.

Definitie 3.4 (Totale stroom)**.** *De totale stroom die afkomstig is uit bron s definiëren we als*

$$|f| = \sum_{v \in V} f(s,v). \tag{3.8}$$



Figuur 3.4: Illustratie van de elektrische stroomwet van Kirchhoff.

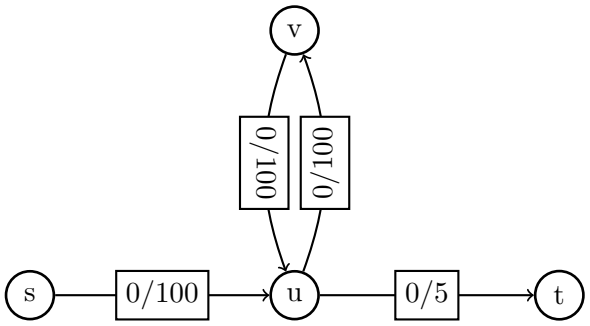
Deze is steeds gelijk aan de totale stroom die in t aankomt. Dit zien we in vanwege het feit dat s deze stroom produceert en de stroom in elke intermediaire knoop nul is vanwege (3.3) en (3.4). Bijgevolg moet de geproduceerde stroom in s volledig geconsumeerd worden door t en stelt (3.8) eigenlijk de *waarde* van de hele stroom f van s naar t doorheen G voor.

3.2 Van netwerk tot stroomnetwerk

Een netwerk, met gegeven bron s en afvoer t, dienen we om te vormen tot een stroomnetwerk wanneer we het wensen te gebruiken om vragen te beantwoorden over stromen die er doorheen lopen. Deze transformatie houdt de volgende stappen in:

1. Verwijder alle inkomende bogen van s en alle uitgaande bogen van t;
2. Verwijder alle lussen;
3. Ken een capaciteit toe aan elke boog;
4. Verwijder de bogen met een capaciteit gelijk aan nul;
5. Elimineer de knopen die niet op minstens één simpel pad van s naar t gelegen zijn (m.b.v. een DFS[1]). In Figuur 3.5 zien we een voorbeeld van een knoop v die op geen enkel simpel pad van s naar t gelegen is in het stroomnetwerk. Alle stroom die van u naar v gaat, kan v enkel verlaten via u. Zijn aanwezigheid in dit stroomnetwerk is dus duidelijk overbodig!

[1] Depth-First-Search



Figuur 3.5: Illustratie van een te elimineren knoop v.

We komen in het verdere verloop van dit werk niet meer terug op deze transformatie van netwerk naar stroomnetwerk. *We gaan er vanaf nu vanuit dat we steeds met een stroomnetwerk G werken.*

3.3 Residueel stroomnetwerk

We kunnen ons stroomnetwerk G en zijn bijhorende stroom f ook beschouwen in termen van resterende stroom, in plaats van gebruikte stroom, over de bogen. Men spreekt dan van een *residueel stroomnetwerk* dat uit bogen bestaat met een *residuele capaciteit*.

De residuele capaciteit van een boog $(u, v) \in E$, $c_f(u, v) \geq 0$, is gelijk aan de maximale hoeveelheid extra stroom die over deze boog kan gestuurd worden zonder de capaciteitsbeperking (Eigenschap 3.1) te schenden en definiëren deze, met toepassing van het annuleringsprincipe, als volgt:

$$\forall u, v \in V : c_f(u, v) = c(u, v) - f(u, v). \tag{3.9}$$

Merk op dat indien we binnen ons stroomnetwerk, G, géén annuleringsprincipe beschouwen, bovenstaande residuele capaciteit gedefiniëerd zou worden als:

$$\forall u, v \in V : c_f(u, v) = c(u, v) - f(u, v) + f(v, u).$$

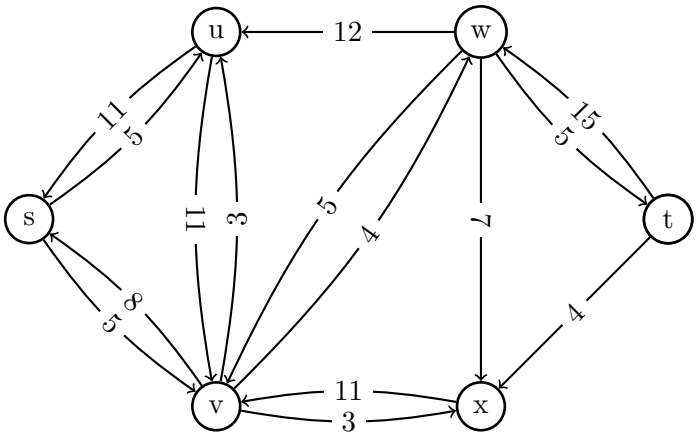
We definiëren een residueel stroomnetwerk van het stroomnetwerk G, ten opzichte van een stroom f, als $G_f = (V, E_f)$, waarbij

$$E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}. \tag{3.10}$$

Een boog (u, v), in een residueel stroomnetwerk, noemen we een *residuele boog*. Merk op dat een dergelijke boog enkel bestaat indien tenminste één van de twee bogen, (u, v) of (v, u), voorkomt in het stroomnetwerk. Wanneer de boog (u, v) verzadigd is in het stroomnetwerk, is er slechts één boog tussen knoop u en knoop v aanwezig in het residueel stroomnetwerk, nl. de boog (v, u).

Gegeven de residuele capaciteiten van de bogen in het residueel stroomnetwerk en de capaciteiten van de bogen in het gerelateerde stroomnetwerk, kunnen we de hoeveelheid stroom over elke boog in dit stroomnetwerk bepalen. Gegeven de hoeveelheid stroom over elke boog in het stroomnetwerk en de residuele capaciteiten van de bogen in het residueel stroomnetwerk, kunnen we de capaciteiten van de bogen in het stroomnetwerk bepalen. De twee voorgaande redeneringen volgen beide uit (3.9) en mede vanwege deze laatste, zien we in dat we de residuele capaciteit van een boog, $c_f(u, v)$, in constante tijd kunnen berekenen uit $c(u, v)$ en $f(u, v)$. We veronderstellen, in het vervolg van dit werk, dan ook impliciet het bestaan van een residueel stroomnetwerk.

In Figuur 3.6 zien we een voorbeeld van een residueel stroomnetwerk. Dit residueel stroomnetwerk werd berekend uit het stroomnetwerk uit Figuur 3.1. De labels van de bogen komen overeen met de residuele capaciteit van de bogen en zijn, visueel gezien, verwerkt in de boog. Dit laatste is verschillend ten opzichte van de visualisatie van het gewone stroomnetwerk en zorgt er voor dat de lezer in het verdere verloop van dit werk een duidelijk onderscheid kan maken tussen een stroomnetwerk en een residueel stroomnetwerk.



Figuur 3.6: Illustratie van een residueel stroomnetwerk.

Residuele stroomnetwerken spelen een centrale rol in de oplossingsmethoden voor het maximumstroomprobleem, die we later zullen bestuderen, en maken het voor de lezer ook makkelijker te begrijpen.

3.4 s-t-boogsnede

We voeren nu een speciale notie van een boogsnede in een stroomnetwerk in, nl. een *s-t-boogsnede*. Een s-t-boogsnede is een boogsnede $[S, T]$, met

$T = V \setminus S$, zodanig dat voor de bron s en de afvoer t van het stroomnetwerk geldt dat $s \in S$ en $t \in T$.

In het vervolg van dit werk bedoelen we met een boogsnede steeds een s-t-boogsnede, tenzij expliciet anders vermeld.

We noteren de *capaciteit van een boogsnede* van een stroomnetwerk als $c\,[S,T]$ en definiëren deze als volgt:

$$c\,[S,T] = \sum_{(u,v)\in(S,T)} c(u,v). \tag{3.11}$$

$c\,[S,T]$ is dus gelijk aan de totale capaciteit van de voorwaartse bogen van $[S,T]$. Een *minimale boogsnede* van een stroomnetwerk heeft een capaciteit die de minimum capaciteit is onder alle boogsnedes van het stroomnetwerk. Merk op dat een minimale boogsnede niet noodzakelijk uniek is!

We noteren de *stroom over een boogsnede* van een stroomnetwerk als $f\,[S,T]$ en definiëren deze als volgt:

$$f\,[S,T] = \sum_{\substack{(u,v)\in(S,T)\\ f(u,v)>0}} f(u,v) - \sum_{\substack{(v,u)\in(T,S)\\ f(v,u)>0}} f(v,u). \tag{3.12}$$

$f\,[S,T]$ is dus in beide richtingen opgebouwd uit strikt positieve stroom tussen S en T: het verschil tussen de totale strikt positieve stroom over de voorwaartse bogen van $[S,T]$ en de totale strikt positieve stroom over de achterwaartse bogen van $[S,T]$. In wat volgt, bekijken we nog twee stellingen in verband met boogsnedes van een stroomnetwerk.

Stelling 3.5. *Indien f een stroom is in een stroomnetwerk G en $[S,T]$ is een boogsnede van G, dan geldt er dat $|f| = f\,[S,T]$.*

Stelling 3.5 zegt met andere woorden dat, voor een gegeven stroom doorheen een stroomnetwerk, de stroom over elke mogelijke boogsnede van dat stroomnetwerk hetzelfde is. Deze stelling bevestigt nogmaals het feit dat de waarde van een stroom gelijk is aan de totale stroom die aankomt in de afvoer t. We zien dit in door de volgende boogsnedes in G te beschouwen: $[S_1,T_1]$, met $S_1 = \{s\}$ en $T_1 = V\setminus\{s\}$ en $[S_2,T_2]$, met $S_2 = V\setminus\{t\}$ en $T_2 = t$. De stroom over $[S_1,T_1]$ is gelijk aan de totale stroom die vertrekt vanuit de bron. De stroom over $[S_2,T_2]$ is gelijk aan de totale stroom die aankomt in de afvoer. Vanwege Stelling 3.5 zijn de stromen over beide boogsnedes gelijk. Wanneer we ook nog (3.8) erbij betrekken, kunnen we concluderen dat de totale stroom die aankomt in de afvoer wel degelijk gelijk is aan de waarde van de stroom binnen G. De volgende stelling is een rechtstreeks en duidelijk gevolg van Stelling 3.5:

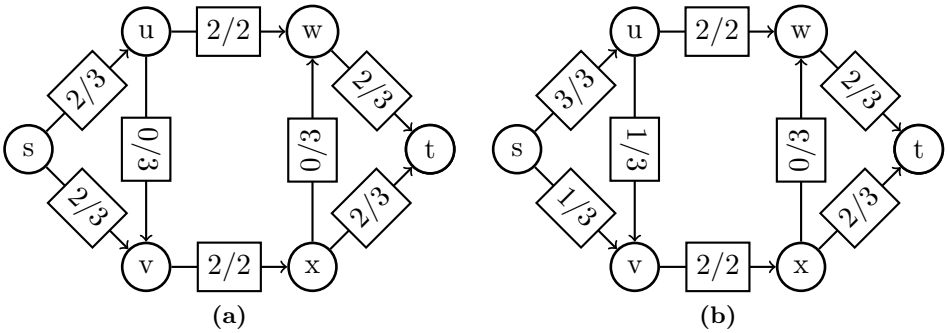
Stelling 3.6. *De waarde van eender welke stroom f in een stroomnetwerk G is kleiner dan of gelijk aan de capaciteit van eender welke boogsnede in G.*

Stelling 3.6 zegt met andere woorden dat de waarde van eender welke stroom niet groter kan zijn dan de capaciteit van een minimale boogsnede. Stel immers dat de waarde van een stroom groter is dan de capaciteit van een minimale boogsnede, dan ervaren we een tegenstrijdigheid met Stelling 3.5 aangezien we, om hieraan te voldoen, de capaciteitsbeperking (Eigenschap 3.1) moeten verbreken.

Hoofdstuk 4

Het maximumstroomprobleem

Het *maximumstroomprobleem* is een probleem met betrekking tot stroomnetwerken en dus, zoals reeds aangegeven in Hoofdstuk 2, een graafoptimalisatieprobleem. Men formuleert dit probleem in het algemeen als volgt: *bepaal de maximale hoeveelheid stroom die doorheen een bepaald stroomnetwerk kan gezonden worden per eenheid van tijd en tevens de manier waarop deze stroom over dit hele stroomnetwerk verdeeld kan worden.* De manier waarop de maximale stroom precies verdeeld is, kan mogelijk verschillend zijn van oplossing tot oplossing. In Figuur 4.1 zien we een voorbeeld van een stroomnetwerk met een maximale stroom die op twee manieren verdeeld is.



Figuur 4.1: Illustratie van een stroomnetwerk met een maximale stroom. In (a) is de maximale stroom anders verdeeld dan in (b).

4.1 In de praktijk

Voorbeelden van het maximumstroomprobleem, uit het dagelijkse leven, vinden we terug in het bepalen van de maximale hoeveelheid stroom van:

- petroleumproducten in een netwerk van pijpleidingen,
- auto's in een wegennetwerk,
- berichten in een telecommunicatienetwerk,
- elektriciteit in een elektrisch netwerk,
- ...

per tijdseenheid.

Uit de voorbeelden van toepassingen die we hier geven, blijkt dat het gestelde probleem zich voornamelijk voordoet binnen scenario's die van toepassing zijn op fysieke netwerken en meer bepaald op *transportnetwerken*. De knopen, bogen, capaciteiten en de stroom uit het gemodelleerde stroomnetwerk vormen hierbij een representatie van de verschillende type fysieke entiteiten binnen het desbetreffende fysieke netwerk. We nemen als voorbeeld de fysieke entiteiten uit een wegennetwerk:

- knopen $\approx$ kruispunten,
- bogen $\approx$ wegen,
- capaciteiten $\approx$ combinatie aantal vakken en toegelaten snelheid van de wegen, met andere woorden het maximaal aantal auto's dat kan passeren per tijdseenheid.
- stroom $\approx$ aantal auto's per tijdseenheid.

Een andere voorbeeld dat te maken heeft met transport vinden we terug in het Harris-Ross verslag waarin de spoorwegen van de Sovjet Unie en Oost-Europa gemodelleerd werden als een maximumstroomprobleem. Het stroomnetwerk had 44 knopen en 105 bogen. De maximale stroom, die uiteindelijk gevonden werd, bedroeg 163.000 ton goederen. Harris was daarenboven de persoon die Ford en Fulkerson geïnspireerd heeft tot de ontwikkeling van hun bekende Maximumstroom-minimumboogsnede-stelling [17]. We zullen op deze stelling terugkomen in Hoofdstuk 5.

Het probleem kan zich echter ook voordoen als een optimalisatieprobleem dat op het eerste zicht helemaal geen betrekking heeft tot grafen of netwerken. Het probleem van *scheduling* is hier een belangrijk voorbeeld van.

Stel bijvoorbeeld een aantal uniforme, parallel geschakelde machines, een aantal uit te voeren taken en een beschikbaar tijdsinterval per taak. We kunnen dan bekijken of er een verdeling van deze taken over de machines en het totaal beschikbaar tijdsinterval bestaat zodanig dat de machines de taken allemaal volledig kunnen uitvoeren binnen de beschikbare tijd. We illustreren dit aan de hand van het volgende voorbeeld: veronderstel drie machines

($M = 3$), vier taken met bijhorende uitvoeringstijd (p_j), het tijdstip vanaf wanneer de uitvoering mag starten (r_j) en het tijdstip vanaf wanneer de taak voltooid moet zijn (d_j), zie Tabel 4.1.

Taak j	1	2	3	4
Uitvoeringstijd p_j	1.5	1.25	2.1	3.6
Starttijd r_j	3	1	3	5
Deadline d_j	5	4	7	9

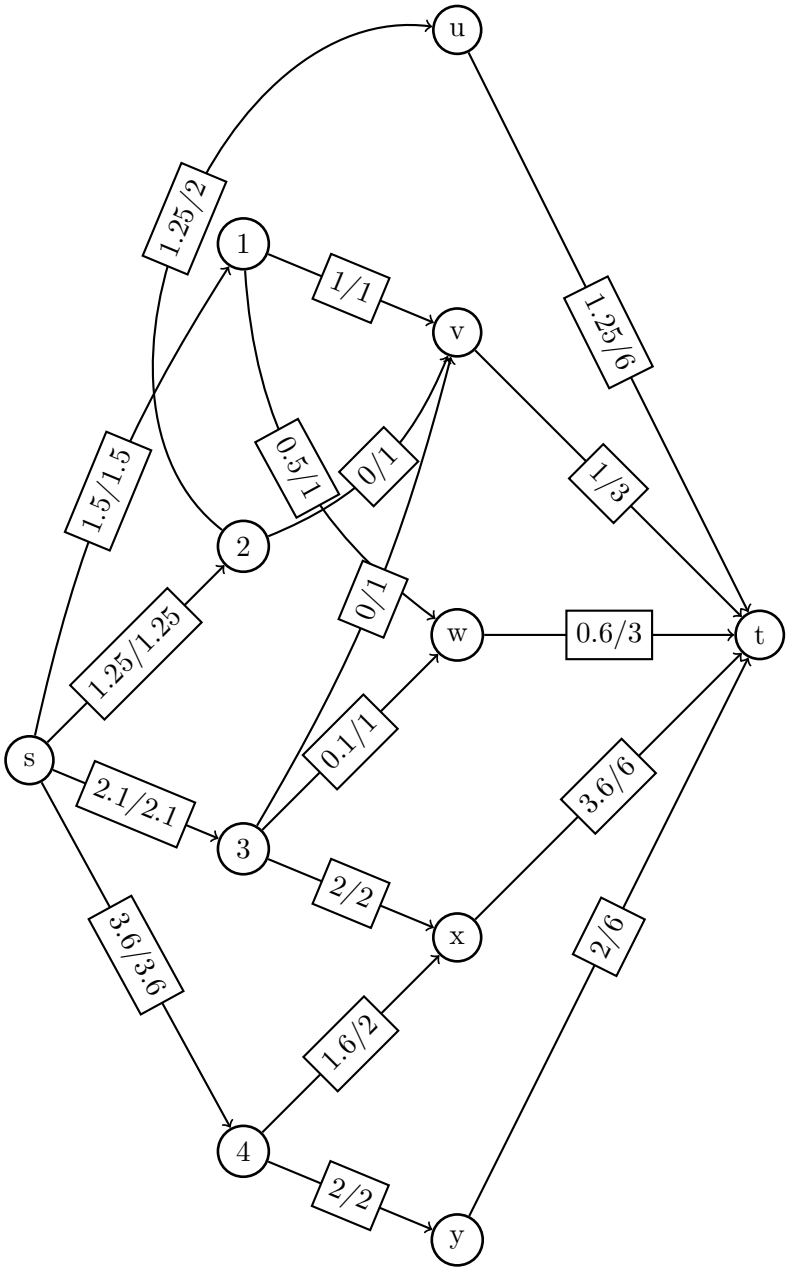
Tabel 4.1: Voorbeeld van een scheduling probleem met betrekking tot de verdeling van taken over drie uniforme, parallel geschakelde machines.

We kunnen nu de starttijden en deadlines in volgorde opschrijven: 1, 3, 4, 5, 7, 9 en er vervolgens de volgende tijdsintervallen uit samenstellen: $T_{1,3}$, $T_{3,4}$, $T_{4,5}$, $T_{5,7}$ en $T_{7,9}$. We merken nu op dat we voor elk tijdsinterval $T_{k,l}$ de taken j kunnen uitvoeren waarvoor geldt dat $r_j \leq k$ en $d_j \geq l$. We formuleren nu dit voorbeeld als een maximumstroomprobleem.

We hebben een bron s, een afvoer t, een knoop voor elke taak j en een knoop voor elk tijdsinterval $T_{k,l}$. We verbinden s met elke taak j en dit met capaciteit p_j, hetgeen de totale uitvoeringstijd is van taak j. Vervolgens verbinden we elk tijdsinterval $T_{k,l}$ met t door middel van een boog met capaciteit $(l - k) \times M$. Deze laatste capaciteit stelt de totale beschikbare uitvoeringstijd voor op alle machines te samen van tijdstip k tot tijdstip l. Tot slot verbinden we elke taak j met een knoop $T_{k,l}$ indien $r_j \leq k$ en $d_j \geq l$ en dit door middel van een boog met capaciteit $(l-k)$, hetgeen de maximale uitvoeringstijd voorstelt die kan worden toegekend aan taak j van tijdstip k tot tijdstip l. In Figuur 4.2 op pagina 20 zien we een illustratie van het hierboven beschreven stroomnetwerk.

Wanneer we nu het maximumstroomprobleem oplossen voor dit stroomnetwerk weten we dat de vier taken uitvoerbaar zijn indien de waarde van de maximale stroom gelijk is aan de som van de uitvoeringstijden van alle taken. In het andere geval is deze kleiner dan de som van de uitvoeringstijden van alle taken en weten we dat we de vier taken, gegeven de drie machines en de beschikbare tijd, niet allemaal volledig kunnen uitvoeren. In Figuur 4.2 op pagina 20 zien we dat de waarde van de maximale stroom gelijk is aan de som van de uitvoeringstijden van alle taken. Bijgevolg zijn de vier taken volledig uitvoerbaar binnen de opgegeven tijdsgrenzen.

De voorbeelden van toepassingen die we tot nu toe vermeld hebben, geven slechts een beknopt overzicht van het belang van het maximumstroomprobleem in de praktijk. Voor meer voorbeelden verwijzen we naar de literatuur, bijvoorbeeld [6].



Figuur 4.2: Illustratie van scheduling stroomnetwerk met knoop: $u = T_{1,3}$, $v = T_{3,4}$, $w = T_{4,5}$, $x = T_{5,7}$ en $y = T_{7,9}$.

4.2 Meerdere bronnen en/of afvoeren

Het maximumstroomprobleem kan ook betrekking hebben op stroomnetwerken met *meerdere bronnen* $\{s_1, \ldots, s_k\}$ *en/of afvoeren* $\{t_1, \ldots, t_l\}$. Om een dergelijke vorm van het probleem op te lossen, kunnen we het probleem reduceren naar een maximumstroomprobleem dat betrekking heeft op een stroomnetwerk met één bron s en één afvoer t. We zetten hiervoor het stroomnetwerk met meerdere bronnen en/of afvoeren om naar een stroomnetwerk met één bron s en één afvoer t. Deze omzetting houdt in dat we:

- een nieuwe bron s en/of een nieuwe afvoer t invoegen,
- gerichte bogen (s, s_i) invoegen met $c(s, s_i) = +\infty$ voor iedere $i = 1, \ldots, k$ en/of,
- gerichte bogen (t_i, t) invoegen met $c(t_i, t) = +\infty$ voor iedere $i = 1, \ldots, l$.

Het is nu duidelijk dat s de nodige stroom voor alle s_i kan leveren en dat t altijd genoeg capaciteit heeft om de stroom afkomstig van alle t_i op te nemen. Door het toepassen van de beschreven omzetting, zorgen we er voor dat het oplossen van het maximumstroomprobleem op een stroomnetwerk met meerdere bronnen en/of afvoeren gereduceerd wordt tot het maximumstroomprobleem met één bron en één afvoer.

Merk op dat het zelfs voldoende zou zijn om de gerichte bogen (s, s_i) te voorzien van een capaciteit die gelijk is aan de som van de capaciteiten van de bogen vertrekkende vanuit s_i. De gerichte bogen (t_i, t) kunnen we dan analoog voorzien van een capaciteit die gelijk is aan de som van de capaciteiten van de bogen die aankomen in t_i.

4.3 Oplossingsmethoden

Voor een gegeven stroomnetwerk bestaan er slechts een eindig aantal mogelijke, geldige stromen. We veronderstellen hiervoor dat alle simpele paden van de bron s naar de afvoer t bestaan uit minstens één boog (u, v) met $c(u, v) \neq +\infty$. Een naïve oplossing zou erin bestaan alle mogelijke stromen op te sommen en diegene met de hoogste stroomwaarde terug te geven. Dit is in praktijk echter niet haalbaar aangezien het aantal mogelijke alternatieven zéér hoog kan oplopen.

Men is echter redelijk snel tot het inzicht gekomen dat het maximumstroomprobleem efficiënt oplosbaar is. Er zijn twee belangrijke oplossingsmethoden gekend:

1. **Toenemende-pad-methode**[1],

[1] In het Engels: Augmenting Path method

2. **Prestroom-duw-methode**[2], ook wel de **Duw-verhoog-methode**[3] genoemd.

De toenemende-pad-methode werd ontworpen door *Ford* en *Fulkerson*. De methode stuurt steeds stroom over simpele paden van s naar t totdat er geen simpele paden van s naar t overblijven in het residueel stroomnetwerk.

Goldberg en *Tarjan* ontwierpen de prestroom-duw-methode. Er wordt door deze methode eerst stroom vanuit de bron in het stroomnetwerk gestuurd en laat tijdens zijn verdere uitvoering toe dat knopen de eigenschap van behoud van stroom schenden. De overtollige stroom die knopen hebben, wordt geleidelijk aan naar de afvoer gestuurd en indien dit niet mogelijk is, terug naar de bron. De resulterende stroom, op het einde van de uitvoering, voldoet dus wel aan de eigenschap van behoud van stroom.

We bespreken de twee methoden gedetailleerd in Hoofdstuk 5 en Hoofdstuk 6.

[2]In het Engels: Preflow Push method
[3]In het Engels: Push Relabel method

Hoofdstuk 5

Toenemende-pad-methode

Bij de toenemende-pad-methode van Ford-Fulkerson beschouwen we het maximumstroomprobleem op een stroomnetwerk G (met bijhorend residueel stroomnetwerk G_f). De werking en correctheid van de toenemende-pad-methode zijn gebaseerd op drie belangrijke begrippen waarvan we de eerste twee reeds vermeld hebben in Secties 3.3 en 3.4: residueel stroomnetwerk, s-t-boogsnede en *toenemend pad*.

5.1 Toenemend pad

We definiëren een toenemend pad p in een stroomnetwerk G als een simpel pad, van de bron s naar de afvoer t, in het bijhorende residueel stroomnetwerk G_f. Elke boog van p heeft een residuele capaciteit en dus kan er in G een extra stroom gestuurd worden over elke boog waarvoor er een residuele boog is die deel uitmaakt van p, en dit zonder de capaciteiten van deze bogen te overschrijven.

De *residuele capaciteit van een toenemend pad* p, $c_f(p)$, is het minimum van de residuele capaciteit van de bogen van p:

$$c_f(p) = min\{c_f(u,v) : (u,v) \textit{ ligt op het toenemend pad } p\}. \tag{5.1}$$

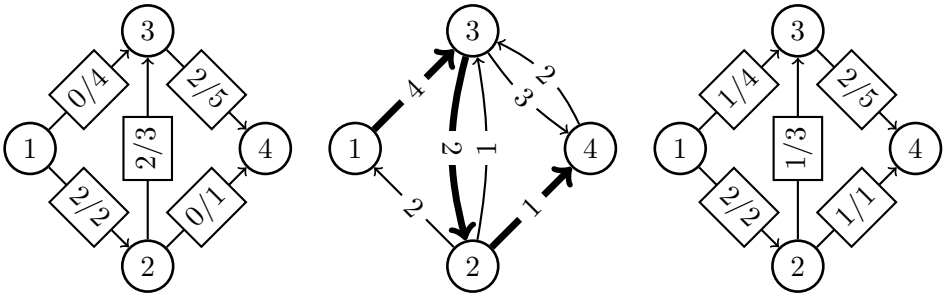
Stelling 5.1. *Veronderstel dat p een toenemend pad is in G_f, dan definiëren we een functie $f_p : V \times V \to]-\infty, +\infty[$ waarbij $f_p(u,v)$ gelijk is aan:*

- *$c_f(p)$ als (u,v) tot p behoort,*
- *$-c_f(p)$ als (v,u) tot p behoort,*
- *nul wanneer noch (u,v), noch (v,u) tot p behoren.*

We noemen deze functie een stroom in G_f met als waarde $|f_p| = c_f(p) > 0$ en is de extra stroom, over of via p, die we aan stroom f in G kunnen toevoegen. We zeggen dan dat we een extra stroom over p sturen. Deze extra

stroom kan nooit nul zijn, aangezien bogen met een residuele capaciteit gelijk aan nul niet tot het residueel stroomnetwerk behoren.

We illustreren dit alles aan de hand van het stroomnetwerk in Figuur 5.1. In Figuur 5.1(a) heeft het stroomnetwerk een stroom waarvan de waarde gelijk is aan twee. Wanneer we nu het bijhorende residueel stroomnetwerk bekijken in Figuur 5.1(b) en de stroom binnen het stroomnetwerk verhogen volgens de residuele capaciteit van het aangeduide pad p, die gelijk is aan één, krijgen we een stroom doorheen het stroomnetwerk zoals aangegeven in Figuur 5.1(c). We zien nu duidelijk in dat $f_p(1,3) = 1$, $f_p(2,3) = -1$ en $f_p(2,4) = 1$, aangezien de bogen $(1,3)$, $(3,2)$ en $(2,4)$ tot p behoren.



(a) Stroomnetwerk met stroomwaarde gelijk aan 2

(b) Overeenstemmend residueel stroomnetwerk van het stroomnetwerk in (a)

(c) Resultaat van het toevoegen van stroom volgens het toenemend pad uit (b)

Figuur 5.1: Stroom over een toenemend pad.

Wanneer we f_p toevoegen aan f dan krijgen we een nieuwe stroom, f'. We definiëren deze nieuwe stroom als een functie: $f' : V \times V \to]-\infty, +\infty[$ waarvoor geldt dat $f' = f + f_p$ en waarvan de waarde, $|f'| = |f| + |f_p| > |f|$. Als we dit principe herhaaldelijk toepassen, zal de stroom in G noodzakelijkerwijs convergeren naar een maximale stroom.

We besluiten met een logische observatie die volgt uit deze sectie en waar de werking van de toenemende-pad-methode essentieel op gebaseerd is: *zolang G_f toenemende paden bevat, kunnen we extra stroom sturen van bron s naar afvoer t.*

5.2 Werking

De methode start met een initiële stroom gelijk aan nul. De toenemende-pad-methode werkt daarna op een iteratieve manier. In elke iteratie zoeken we een toenemend pad. Zolang er een toenemend pad kan gevonden worden, sturen we een extra stroom over dit toenemend pad. Vanaf het moment dat

er geen toenemend pad meer kan gevonden worden, is er een maximale stroom in het stroomnetwerk gevonden. Algoritme 1 toont de toenemende-pad-methode in pseudocode.

Algoritme 1 Pseudocode van de toenemende-pad-methode.

```
 1: function FordFulkerson(G)
 2:     for each (u, v) ∈ E do
 3:         f(u, v) ← 0
                ▷ We veronderstellen nog steeds: (u, v) ∈ E ⇒ (v, u) ∈ E.
 4:     end for

 5:     while ∃ een toenemend pad p in G_f do
 6:         c_f(p) ← min{c_f(u, v) : (u, v) ligt op p}
 7:         for each (u, v) op p do
 8:             f(u, v) ← f(u, v) + c_f(p)
 9:             f(v, u) ← −f(u, v)
10:         end for
11:     end while

12:     return f
13: end function
```

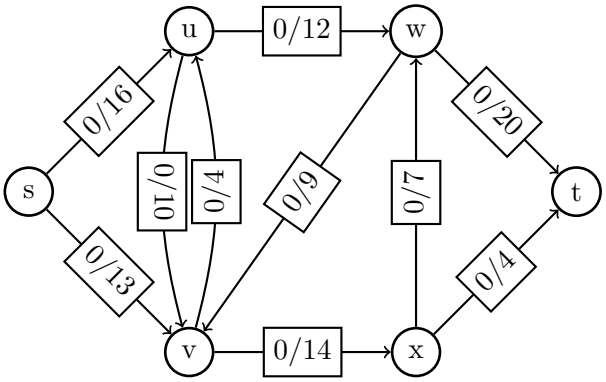
We stellen de stroom over iedere boog $(u, v) \in E$, $f(u, v)$, initieel gelijk aan nul om de hele stroom doorheen G te initialiseren op nul. Dit gebeurt in lijnen 2-4. In lijnen 5-11 wordt er telkens een toenemend pad geselecteerd, zijn residuele capaciteit bepaald en de stroom in G verhoogd aan de hand van deze residuele capaciteit. Dit wordt herhaald tot er geen toenemende paden meer overblijven. Merk op dat de toenemende-pad-methode de specifieke implementatie voor de zoektocht van een toenemend pad, lijn 5, openlaat.

We illustreren nu de werking van de toenemende-pad-methode op het stroomnetwerk uit Figuur 5.2. In Figuur 5.3(a) zien we het bijhorende residueel stroomnetwerk waarop we het algoritme uitvoeren. We duiden een toenemend pad aan door deze in het residueel stroomnetwerk in het vet te zetten. Hoe het stroomnetwerk zelf er tijdens de verschillende iteraties uitziet, wordt als oefening overgelaten aan de lezer.

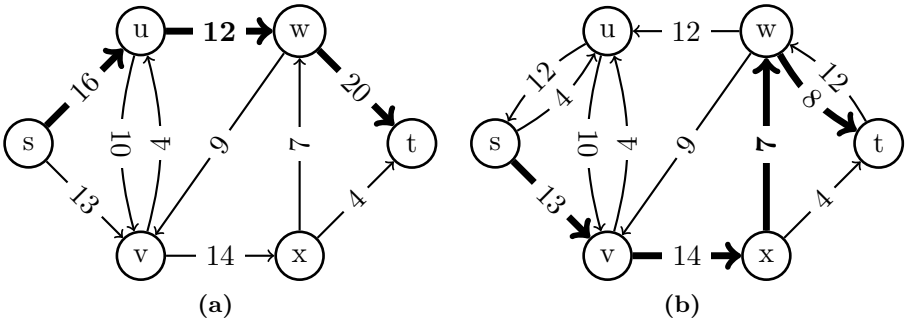
Wanneer we nu in de uitvoering van de methode het toenemende pad nemen, zoals aangeduid in Figuur 5.3(a), kunnen we de stroom in het stroomnetwerk verhogen met de residuele capaciteit van dit pad. Deze residuele capaciteit is in dit geval gelijk aan 12.

We krijgen dan als resultaat het residueel stroomnetwerk zoals aangegeven in Figuur 5.3(b). Wanneer we het zonet beschreven principe nog eens tweemaal toepassen, krijgen we als resultaat de residuele stroomnetwerken uit Figuren 5.4(a), 5.4(b).

Uit het residueel stroomnetwerk dat weergegeven wordt in Figuur 5.4(b)



Figuur 5.2: Voorbeeld uitvoering van de toenemende-pad-methode.

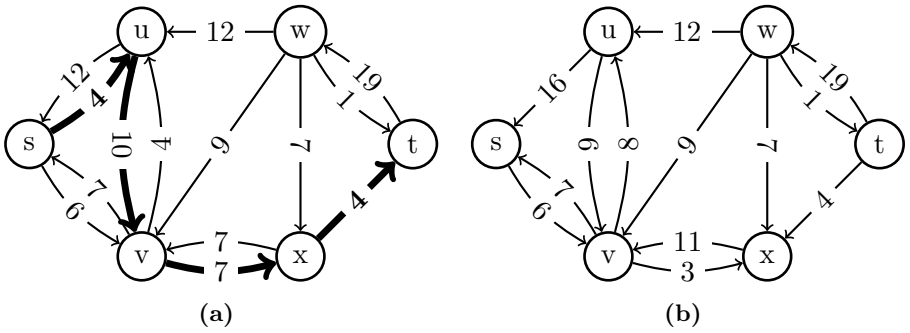


Figuur 5.3: Voorbeeld uitvoering van de toenemende-pad-methode.

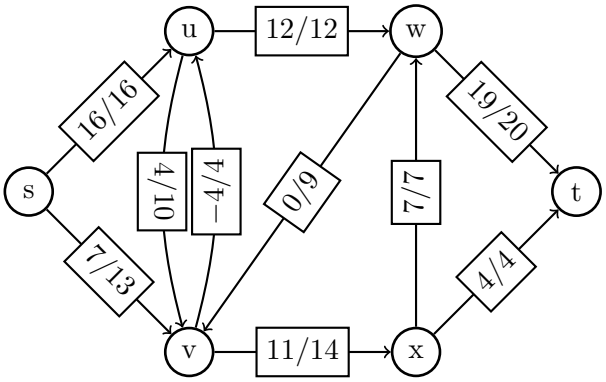
blijkt dat we geen toenemend pad meer kunnen selecteren aangezien er geen meer is. Bijgevolg eindigt de uitvoering van de methode en hebben we een maximale stroom gevonden waarvan de waarde gelijk is aan 23, zie Figuur 5.5 voor de resulterende stroom doorheen het stroomnetwerk waarop we de methode hebben losgelaten.

5.3 Correctheid

Uit Stelling 3.6 weten we dat de waarde van een maximale stroom doorheen een stroomnetwerk gelijk is aan de capaciteit van een minimale boogsnede van het stroomnetwerk. We kunnen dit verduidelijken via de vergelijking met een flessenhals in een fysiek netwerk. In een wegennetwerk zijn de wegen waar files ontstaan de flessenhals van het netwerk. Automobilisten zullen automatisch proberen een alternatieve route te zoeken zodat de capaciteit van de minimale boogsnede tussen bijvoorbeeld Hasselt en Brussel uiteindelijk, tijdens de spits, volledig benut wordt.



Figuur 5.4: Voorbeeld uitvoering van de toenemende-pad-methode.

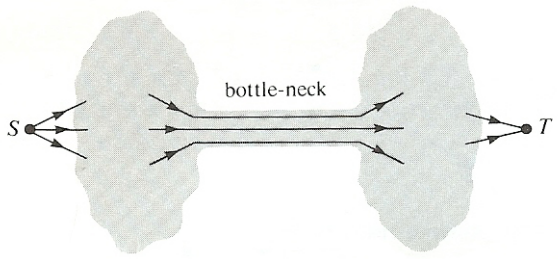


Figuur 5.5: Voorbeeld uitvoering van de toenemende-pad-methode.

Het is intuïtief duidelijk dat de hoeveelheid stroom doorheen deze netwerken begrensd is door de hoeveelheid stroom die er per tijdseenheid doorheen de flessenhals kan (zie Figuur 5.6). Een minimale boogsnede in een stroomnetwerk is dus een flessenhals van het stroomnetwerk. We formuleren het vastgestelde feit in de vorm van Stelling 5.2 en gebruiken deze tevens om de correctheid van de toenemende-pad-methode van Ford-Fulkerson aan te tonen, namelijk dat f een maximale stroom is in G wanneer er geen toenemend pad meer bestaat in G_f.

Stelling 5.2 (Maximumstroom-minimumboogsnede-stelling)**.** *Volgende uitspraken zijn equivalent:*

1. *f is een maximale stroom in stroomnetwerk G;*
2. *Het residueel stroomnetwerk G_f bevat geen toenemende paden;*
3. *$|f| = c\,[S,T]$ waarbij $[S,T]$ een minimale boogsnede is van het stroomnetwerk G.*



Figuur 5.6: Illustratie van een flessenhals.

Bewijs. Laten we veronderstellen dat f een maximale stroom is en dat G_f een toenemend pad p bevat. Aangezien elke boog van p residuele capaciteit heeft, kan er dus extra stroom gestuurd worden over p, wat in tegenstrijd is met de maximaliteit van f. Bijgevolg kan G_f geen toenemende paden bevatten.

Veronderstel dat G_f geen toenemende paden bevat. We definiëren dan vervolgens een boogsnede $[S, T]$ met $S = \{v \in V : \exists$ een simpel pad van s naar v in $G_f\}$ en $T = V \setminus S$. Bij constructie geldt dat $s \in S$ en $t \notin S$ aangezien er geen simpel pad bestaat van s naar t in G_f, vermits deze geen toenemende paden bevat. Voor ieder paar knopen $u \in S$ en $v \in T$ geldt er dat $f(u, v) = c(u, v)$. Indien dit niet zo zou zijn, zou er gelden dat boog $(u, v) \in E_f$ en is dit in strijd met het feit dat v in T zit. Bijgevolg is het zo dat $f[S, T] = c[S, T] = |f|$, vanwege Stelling 3.5.

Uit dit laatste volgt onmiddellijk dat $|f|$ maximaal is aangezien Stelling 3.6 zegt dat $|f| \leq c[S, T]$ voor alle boogsnedes $[S, T]$. □

5.4 Analyse

In lijnen 2-4, in Algoritme 1, doorlopen we alle bogen van het stroomnetwerk en initialiseren de bijhorende stroom over elke boog (en zijn tegengestelde) op nul. We hebben dus $O(m)$ tijd nodig om deze drie lijnen uit te voeren in de veronderstelling dat we de stroomwaarden over de bogen in constante tijd kunnen aanpassen, met behulp van een adjacentiematrix, en we gebruik maken van een adjacentielijst voor de topologie van het stroomnetwerk.

Hoeveel keer worden lijnen 5-11 uitgevoerd? – Wel, de lus wordt uitgevoerd zolang er een toenemend pad bestaat in het residueel stroomnetwerk of met andere woorden zolang we de stroom in het stroomnetwerk kunnen verhogen omdat de waarde van een maximale stroom nog niet bereikt is. Zij f^* een maximale stroom in een stroomnetwerk en $|f^*|$ zijn waarde. We voeren onze complexiteitsanalyse vanaf nu, naast het aantal bogen m en het aantal knopen n, ook uit in termen van $|f^*|$.

Als elke boog een gehele capaciteit heeft, zal de stroom in elke iteratie

toenemen met een gehele waarde. In het ergste geval wordt de waarde van de stroom in elke iteratie met exact één gehele eenheid verhoogd en zal de lus $|f^*|$ keer worden uitgevoerd. We leiden hier uit af dat de methode stopt voor stroomnetwerken met gehele capaciteiten.

Merk op dat het algoritme ook eindigt indien de capaciteiten rationaal zijn. We kunnen rationale capaciteiten immers terugbrengen naar gehele capaciteiten door de capaciteiten allemaal naar dezelfde noemer te brengen en deze noemer vervolgens te laten vallen.

Binnen de lus moeten telkens de volgende zaken worden uitgevoerd met een totale tijdscomplexiteit van $O(m+n)$:

1. Zoeken van een toenemend pad neemt $O(m)$ tijd in beslag aangezien elke boog ten hoogste één keer onderzocht wordt;

2. Minimale capaciteit van het toenemend pad bepalen om zo de residuele capaciteit van dit pad te bepalen. De kost hiervoor bedraagt $O(n)$ tijd, aangezien we op een toenemend pad hoogstens $n-1$ bogen kunnen hebben;

3. Vervolgens gaan we iedere boog van het toenemend pad af om de stroom volgens die boog in het originele stroomnetwerk bij te werken. De kost hiervoor bedraagt tevens $O(n)$ tijd vanwege dezelfde reden als in het vorige puntje.

Lijnen 5-11 kunnen dus uitgevoerd worden in $O((m+n)|f^*|)$ tijd. De tijd nodig om lijnen 2-4 uit te voeren, is verwaarloosbaar ten opzichte van deze tijd.

Stelling 5.3. *Indien de capaciteiten geheel zijn, kan Algoritme 1 uitgevoerd worden in tijd* $O((m+n)|f^*|)$.

De specifieke implementatie voor de zoektocht van een toenemend pad hebben we tot op heden opengelaten. *Edmonds* en *Karp* pasten twee strategieën toe om toenemende paden te zoeken: het zoeken van het kortste pad en het zoeken van het pad met de grootste residuele capaciteit. Ze bewezen dat beide implementaties van Ford-Fulkerson's methode uitvoerbaar zijn in $O(nm^2)$ en resp. $O(m^2 \log m \log |f^*|)$ tijd. Voor meer informatie hieromtrent verwijzen we naar [15] en [19].

5.5 Nadelen

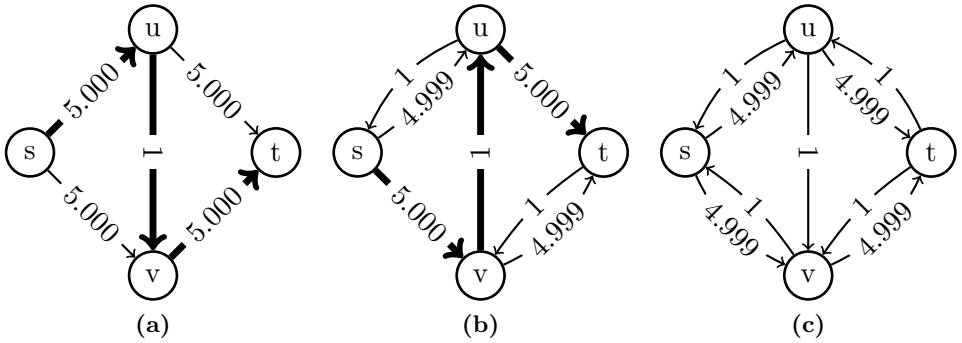
Het aantal iteraties dat de toenemende-pad-methode van Ford-Fulkerson uitvoert, zijn, zoals gebleken is uit de complexiteitsanalyse, begrensd door de waarde van de maximale stroom. Deze waarde kan echter arbitrair groot worden, bijvoorbeeld $|f^*| = 2^n$, waardoor de uitvoeringstijd exponentiëel kan zijn. Wanneer $|f^*|$ klein is, is de uitvoeringstijd van de methode goed.

Laten we verder een eenvoudig stroomnetwerk beschouwen waarvoor $|f^*|$ heel groot is, namelijk 10.000 eenheden stroom, zie Figuur 5.7. Het probleem bij dit residueel stroomnetwerk is het feit dat, in het slechtste geval, steeds een toenemend pad over de boog tussen u en v zal gekozen worden en de residuele boog met capaciteit 1 tussen knoop u en v steeds zal omflippen. De waarde van de stroom zal in dat geval 10.000 keer geïncrementeerd worden met waarde 1. We zien in Figuren 5.7(a), 5.7(b) en 5.7(c) het resultaat van de uitvoering van twee mogelijke eerste stappen in een dergelijke situatie.

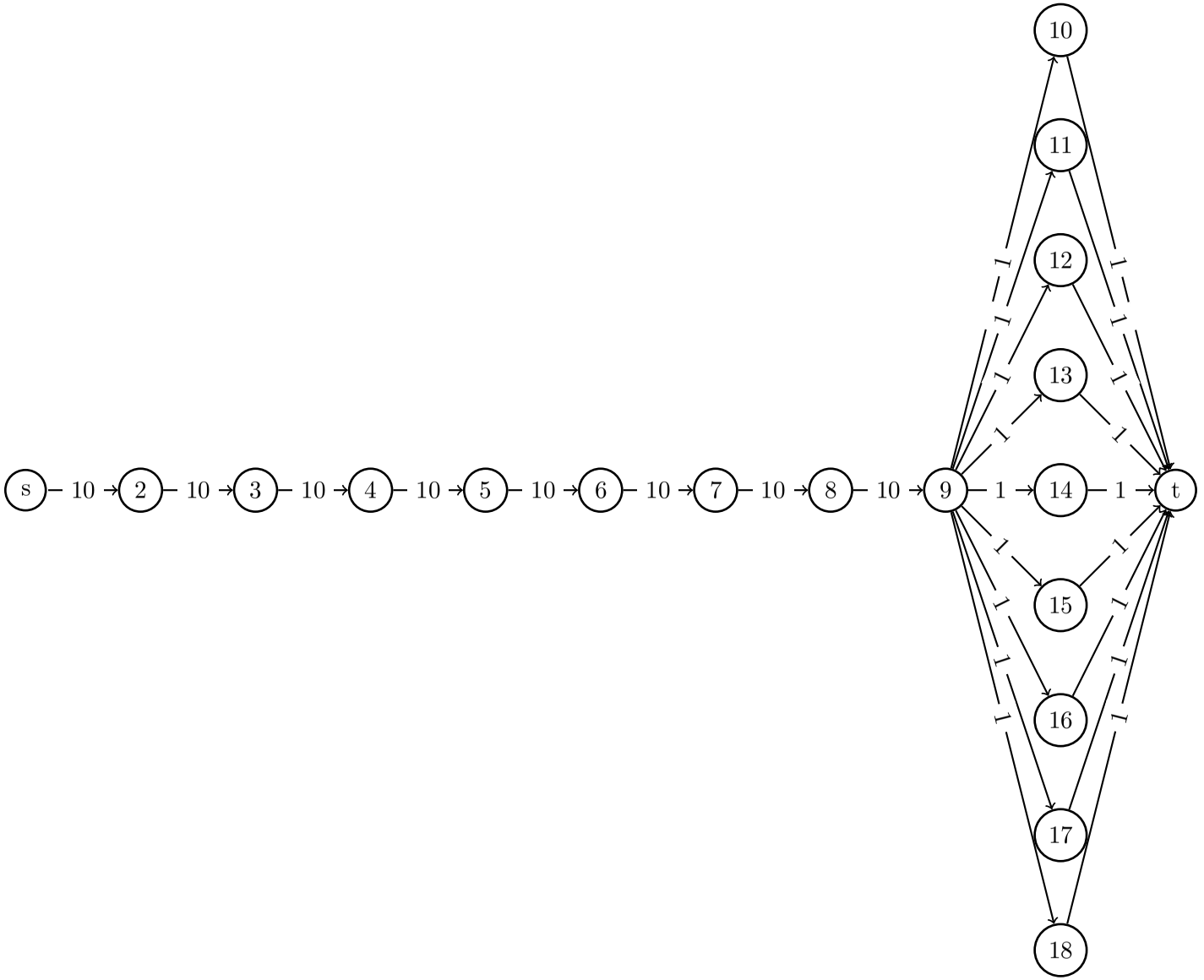
Verder is het zo dat in elke iteratie stroom moet gezonden worden over het hele gevonden toenemend pad. Dit vereist in het slechtse geval telkens $O(n)$ tijd. Dit kan er tevens voor zorgen dat er tijdens de uitvoering van de methode herhaaldelijk kleine hoeveelheden stroom gestuurd worden over gemeenschappelijke padsegmenten, hetgeen het **inherente nadeel** vormt van de toenemende-pad-methode. Een duidelijk voorbeeld hiervan vinden we terug in Figuur 5.8. In elke iteratie wordt er een extra hoeveelheid stroom met waarde 1 toegestaan. Het zou echter beter zijn om eerst 9 eenheden stroom tot in knoop 9 te sturen en dan vervolgens deze 9 eenheden te verdelen over zijn uitgaande bogen.

Een derde nadeel is dat de methode mogelijk niet eindigt wanneer de toenemende paden slecht gekozen worden. Dit in de zin dat er in elke iteratie toenemende paden optreden met een steeds kleiner wordende residuele capaciteit. We krijgen zo de mogelijkheid dat de stroom in het betrokken stroomnetwerk niet convergeert naar een maximale waarde. Dit kan echter alleen maar gebeuren indien er onder de capaciteiten van de bogen irrationale getallen voorkomen. Dit effect werd reeds door Ford en Fulkerson met een voorbeeld aangetoond, maar het kleinst mogelijke voorbeeld van dit effect werd beschreven door *Uri Zwick*. We achten dit derde nadeel verder niet zo belangrijk aangezien computers enkel getallen exact kunnen voorstellen die geheel zijn of die dyadisch[1] rationaal zijn. We gaan daarom ook niet

[1] met als noemer een macht van twee



Figuur 5.7: Illustratie van flippen bij uitvoering van de toenemende-pad-methode.



Figuur 5.8: Illustratie inherente nadeel toenemende-pad-methode.

dieper in op de vermelde voorbeelden, maar verwijzen hiervoor naar [19] en [20].

5.6 Andere oplossingsmethoden

Dinic ontwikkelde later de *blokkerende-stroom-methode*, [3] en [7], die gerelateerd is aan de toenemende-pad-methode. Hij stelde voor om de toenemende paden in fasen te zoeken. In fase i wordt er telkens een stroom gestuurd over kortste toenemende paden van lengte i. Fase i eindigt wanneer er zo geen toenemend pad overblijft en men zegt dan dat de bekomen stroom alle toenemende paden van lengte i *blokkeert*. Deze methode voert hoogstens $n-1$ fasen uit aangezien dit de maximale lengte is van zo'n toenemend pad en i na afloop van elke fase sowieso groter wordt. Aangezien een snellere implementatie van de blokkerende-stroom-methode neerkomt op het sneller vinden van een blokkerende stroom, introduceerden Dinic en *Karzanov* hiervoor later betere algoritmen.

Karzanov [7] introduceerde tevens het concept *prestroom*. Een prestroom is een stroom in een stroomnetwerk die toelaat dat intermediaire knopen overstromen. Dit concept gaf aanleiding tot de ontwikkeling van *Cherkassky*'s *blokkerende-prestroom-methode*. Deze methode tracht per fase een blokkerende-prestroom, in plaats van een blokkerende stroom, te zoeken. Deze methode kan uiteindelijk gezien worden als een overgang van de toenemende-pad-gebaseerde methoden naar de prestroom-duw- of duw-verhoog-methode van Goldberg en Tarjan. De prestroom-duw-methode overtreft alle andere methoden in de praktijk en wordt dan ook uitvoerig besproken in het volgende hoofdstuk.

Hoofdstuk 6

Prestroom-duw-methode

Vanwege de nadelen van de toenemende-pad-gebaseerde methoden, is het nodig dat we op zoek gaan naar een nieuwe, betere, snellere, flexibelere manier om het maximumstroomprobleem op te lossen. We bekijken daarom de *prestroom-duw-methode* van *Goldberg* en *Tarjan*, ook wel de *duw-verhoog-methode* genoemd. Het is namelijk zo dat de snelste algoritmen, die er te vinden zijn voor het oplossen van het maximumstroomprobleem, gebaseerd zijn op deze aanpak. Deze aanpak stuurt, als het ware, stroom over verschillende paden tegelijkertijd. Er wordt meer bepaald stroom gestuurd over individuele bogen, verspreid over het hele stroomnetwerk, in plaats van over toenemende paden.

6.1 Intuïtie

Laten we een fysiek netwerk beschouwen waardoor we een maximale hoeveelheid water willen laten stromen. Het fysiek netwerk bestaat uit buizen waardoor er een bepaalde hoeveelheid water kan stromen. De buizen worden onderling verbonden door middel van knooppunten. We onderscheiden hierbij twee speciale knooppunten: een hoog gelegen bron en een lager gelegen afvoer. Vanuit de bron wordt al het water voor het netwerk aangevoerd en in de lager gelegen afvoer wordt al dit water opgevangen. Elk ander knooppunt is voorzien van een reservoir voor de opvang van water en is samen met dit reservoir en zijn verbindingen met buizen gelegen op een platform. De hoogte van een dergelijk platform is initieel gelijk aan nul en kan toenemen in verloop van tijd. De hoogtes van de platformen bepalen hoe er water van het ene naar het andere, lager gelegen, platform stroomt.

We laten initieel zoveel mogelijk water vanuit de bron doorheen zijn buizen naar beneden stromen. Wanneer er water aankomt in een knooppunt wordt dit opgeslagen in zijn reservoir. Het water in deze reservoirs laten we, knooppunt per knooppunt, telkens verder stromen, doorheen buizen, naar lager gelegen knooppunten. Indien er voor een knooppunt toch nog water in

zijn reservoir aanwezig blijft, omdat de capaciteit van zijn buizen naar lager gelegen knooppunten is opgebruikt, verhogen we diens platform zodanig dat er minstens één nieuwe extra buis naar beneden is.

Dit proces blijft zich herhalen tot uiteindelijk een maximale hoeveelheid water naar de afvoer stroomt. Indien er zich op dat moment nog water bevindt in de reservoirs van bepaalde knooppunten, proberen we dit water weg te krijgen door de betrokken platformen verder te verhogen, hoger dan de gefixeerde hoogte van het platform van de bron. Op die manier stroomt al het teveel aan water terug in de richting van de bron en bekomen we een configuratie van het netwerk die een maximale hoeveelheid water van de bron naar de afvoer laat stromen.

We vertalen dit concept nu naar een theoretisch stroomnetwerk G.

6.2 Prestroom

De knooppunten die voorzien zijn van een reservoir komen overeen met de intermediaire knopen van G. Er wordt in deze reservoirs steeds een bepaalde hoeveelheid water opgeslagen. We leiden hieruit af dat er in intermediaire knopen een teveel aan stroom kan optreden. Dit is in strijd met de wet van behoud van stroom. Het is namelijk niet meer zo dat alles wat toekomt in een intermediaire knoop er ook weer onmiddellijk uitgaat: de totale positieve stroom die de knoop verlaat, heft de totale positieve stroom die in de knoop toekomt niet noodzakelijk volledig op. Pas op het einde zal het overschot aan stroom in de intermediaire knopen verdwenen zijn en de stroom doorheen het stroomnetwerk een geldige stroom zijn.

De prestroom-duw-methode houdt daarom tijdens zijn uitvoering een *prestroom* f_{pre} bij. Deze prestroom f_{pre} is net zoals een gewone stroom doorheen een stroomnetwerk een functie $f_{pre} : V \times V \to]-\infty, +\infty[$ die voldoet aan beperkingen op de capaciteit en het annuleringsprincipe, maar die zich mogelijk **niet** houdt aan de wet van behoud van stroom en bijgevolg **geen** geldige stroom is.

Het teveel aan stroom dat zich in de knopen kan opstapelen, tijdens de uitvoering van de methode, noemen we de *reststroom*. De reststroom in een knoop v drukken we uit als volgt:

$$e(v) = \sum_{\substack{u \in V \\ f(u,v)>0}} f(u,v) - \sum_{\substack{u \in V \\ f(v,u)>0}} f(v,u). \tag{6.1}$$

Indien $e(v)$ groter is dan nul voor een knoop $v \in V$, dan zeggen we dat knoop v *overstroomt* en noemen we v een *actieve* knoop. In het andere geval noemen we v een *inactieve* knoop. De afvoer t overstroomt bij constructie altijd, hetgeen net zijn functie is binnen een stroomnetwerk. De bron is daarenboven de enige knoop met negatieve reststroom.

Afspraak 6.1. *De bron s en de afvoer t zijn nooit actief.*

We zullen soms ook spreken over de reststroom $e(U)$ van een verzameling knopen U, met:

$$e(U) = \sum_{u \in U} e(u). \tag{6.2}$$

Het doel van de prestroom-duw-methode is om de prestroom, die hij onderhoudt tijdens zijn uitvoering, om te zetten tot een geldige stroom. De methode stopt dus wanneer het stroomnetwerk geen actieve knopen meer bevat. Om tot deze situatie te komen, na de initialisatie van een prestroom f_{pre} doorheen het stroomnetwerk, gebruikt de methode twee basisoperaties die enkel van toepassing zijn op actieve knopen:

1. duwoperatie: *duwen van stroom* $\approx$ ledigen van een reservoir,
2. verhoogoperatie: *verhogen van een actieve knoop* $\approx$ verhogen van een platform.

Vooraleer we de initialisatie van de prestroom en beide basisoperaties formaliseren, definiëren we de hoogte van een knoop in het stroomnetwerk. De hoogtes van knopen bepalen immers wanneer we beide basisoperaties kunnen uitvoeren.

6.3 Hoogte van een knoop

We drukken de hoogte van een knoop uit aan de hand van de volgende definitie:

Definitie 6.2 (Hoogte van een knoop)**.** *De hoogte van een knoop wordt gegeven door de hoogtefunctie $h : V \to \mathbb{N}$ waarvoor het volgende geldt:*

- $h(s) = n$,
- $h(t) = 0$,
- $\forall (u, v) \in E_f : h(u) \leq h(v) + 1$.

Uit de zonet gegeven definitie blijkt dat $(u, v) \notin E_f$ wanneer $h(u) > h(v) + 1$. Belangrijk om op te merken is het feit dat de hoogte van de bron, $h(s) = n$, en de hoogte van de afvoer, $h(t) = 0$, nooit mogen wijzigen om te voldoen aan de hoogtefunctie. De reden waarom $h(s)$ gelijk moet zijn aan n zal later duidelijk worden uit het bewijs van Stelling 6.8.

6.4 Initialisatie van de prestroom

De initialisatiefase van de prestroom-duw-methode creëert een initiële prestroom in het stroomnetwerk G. We bekijken dit even in detail in Algoritme 2. In lijnen 2-7 initialiseren we de hoogte van s op n, de hoogte van alle andere knopen op nul en stellen tevens de reststroom voor alle knopen gelijk aan nul. In lijnen 8-10 initialiseren we de prestroom f_{pre} doorheen het hele stroomnetwerk, zodanig dat $|f_{pre}| = 0$. Daarna sturen we vanuit bron s een stroom zodanig dat elke uitgaande boog volledig verzadigd is. De grootst mogelijke hoeveelheid stroom die we kunnen bekomen, is immers gelijk aan de volledige capaciteit van de boogsnede $[S, T]$ met $S = \{s\}$ en $T = V \setminus S$. De buren van s worden nu allemaal actieve knopen zodanig dat er kan gestart worden met de uitvoering van duw- of verhoogoperaties. We vinden de werking van dit principe terug in lijnen 11-16. Deze laatste actie is toegestaan, aangezien al hetgeen teveel is, later toch terug naar de bron vloeit wanneer we tot de situatie komen dat de hoogtes van intermediaire knopen de vastgelegde hoogte van s overstijgen.

Algoritme 2 Pseudocode van de initialisatie van prestroom f_{pre}.

```
1:  procedure INITIALISEER_PRESTROOM(G)
2:      h(s) ← n
3:      e(s) ← 0
4:      for each v ∈ V \ {s} do
5:          h(v) ← 0
6:          e(v) ← 0
7:      end for

8:      for each (u, v) ∈ E do
9:          f_pre(u, v) ← 0
                ▷ We veronderstellen nog steeds: (u, v) ∈ E ⇒ (v, u) ∈ E.
10:     end for

11:     for each v ∈ A(s) do
12:         f_pre(s, v) ← c(s, v)
13:         f_pre(v, s) ← −c(s, v)
14:         e(v) ← c(s, v)
15:         e(s) ← e(s) − c(s, v)
16:     end for
17: end procedure
```

6.5 Duwen van stroom

We willen telkens de reststroom in een intermediaire knoop kwijtspelen. We gaan daarom een actieve knoop beschouwen en enkel kijken naar zijn buren in het residueel stroomnetwerk G_f. Het zijn net de bogen naar deze buren die van belang zijn. Over deze bogen sturen, *duwen*, we de reststroom vanuit de actieve knoop naar zijn buren. We mogen echter niet zomaar over eender welke boog duwen.

Zoals uit de intuïtie achter de prestroom-duw-methode gebleken is, gaan we stroom steeds van boven naar beneden laten vloeien. Vanwege dit principe en de eigenschap van de hoogtefunctie waaraan elke hoogte van een knoop voldoet, is het duidelijk dat er in de prestroom-duw-methode slechts geduwd wordt van een actieve knoop naar deze zijn buren indien deze exact één niveau lager gelegen zijn. De bogen waarover er mag geduwd worden, noemen we *toegelaten* bogen.

De hoeveelheid stroom die we in elke duwoperatie van u naar één van zijn buren v duwen, schrijven we formeel neer als $\delta_{u\to v}$ en noemen we de *duwstroom*. Een duw van $\delta_{u\to v}$ eenheden stroom van knoop u naar knoop v verlaagt $e(u)$ en $c_f(u,v)$ met $\delta_{u\to v}$ en verhoogt $e(v)$ en $c_f(v,u)$ met $\delta_{u\to v}$.

In Algoritme 3 vinden we de pseudocode terug van de duwoperatie uit de prestroom-duw-methode. In lijn 2 berekenen we de hoeveelheid stroom die er geduwd mag worden van knoop u naar knoop v. Deze komt overeen met het minimum van de reststroom van u, $e(u)$, en de residuele capaciteit over de boog (u,v) in het residueel stroomnetwerk en kennen dit aan $\delta_{u\to v}$ toe. Door dit minimum te nemen, zal $e(u)$ nooit negatief worden en de beperking op de capaciteit van de boog (u,v) in het gerelateerde stroomnetwerk zal steeds gerespecteerd blijven. Vervolgens passen we in lijnen 3-4 de prestroom over bogen (u,v) en (v,u) aan volgens de pas berekende duwstroom. Ook de reststroom in beide knopen verandert door de duwoperatie en we passen deze waarden aan in lijnen 5-6.

Algoritme 3 Pseudocode van de duwoperatie over een boog (u,v).

```
1: procedure DUW(u,v)
2:     δ_{u→v} ← min(e(u), c_f(u,v))
3:     f_pre(u,v) ← f_pre(u,v) + δ_{u→v}
4:     f_pre(v,u) ← −f_pre(u,v)
5:     e(u) ← e(u) − δ_{u→v}
6:     e(v) ← e(v) + δ_{u→v}
7: end procedure
```

We spreken van een *verzadigde duwoperatie* als de boog (u,v) verzadigd wordt ($c_f(u,v)=0$) en we spreken van een *onverzadigde duwoperatie* in het andere geval. Na een onverzadigde duwoperatie zal de knoop van waaruit geduwd werd sowieso geen reststroom meer bevatten. Na een verzadigde

duwoperatie zal de knoop van waaruit geduwd werd geen reststroom meer bevatten wanneer deze gelijk was aan de residuele capaciteit van de boog waarover geduwd werd.

6.6 Verhogen van een actieve knoop

Wanneer alle buren v van een actieve knoop u, in het residueel stroomnetwerk, op dezelfde hoogte of hoger gelegen zijn, $h(u) \leq h(v)$, kunnen we geen duwoperatie meer toepassen. We verhogen in dat geval knoop u met behulp van de verhoogoperatie.

In Algoritme 4 vinden we de pseudocode terug van de verhoogoperatie binnen de prestroom-duw-methode. De verhoogoperatie zorgt ervoor dat er opnieuw geduwd kan worden door de knoop v te verhogen tot één niveau hoger dan zijn laagste buur in het residueel stroomnetwerk.

Eigenschap 6.3. *Er is steeds minstens één toegelaten boog vanuit v na een verhoogoperatie.*

Algoritme 4 Pseudocode van de verhoogoperatie op een actieve knoop u.

```
1: procedure VERHOOG(u)
2:     h(u) ← 1 + min{h(v) : (u, v) ∈ E_f}
3: end procedure
```

We weten uit Afspraak 6.1 dat de bron s en de afvoer t van een stroomnetwerk nooit actieve knopen zijn:

Opmerking 6.4. *Er kan tijdens de uitvoering van de prestroom-duw-methode nooit een duw- of verhoogoperatie plaatsvinden op de bron s of de afvoer t van het stroomnetwerk G waarvan we de maximale stroom wensen te bepalen.*

6.7 Generieke prestroom-duw-algoritme

We formaliseren nu de prestroom-duw-methode, in Algoritme 5, in de vorm van het generieke prestroom-duw-algoritme, aangezien we de volgorde waarin we actieve knopen selecteren openlaten. Zolang er een toepasbare duw- of verhoogoperatie bestaat na de initialisatie van de prestroom, selecteren we een actieve knoop en voeren er de correcte operatie op uit. Het is niet mogelijk dat we op een actieve knoop geen operatie kunnen toepassen, zie Stelling 6.5 in Sectie 6.8. Na de uitvoering van de lus in lijnen 3-5 wordt onze prestroom f_{pre} automatisch omgevormd tot een geldige stroom f aangezien er vanaf dat moment geen actieve knopen meer zijn in het stroomnetwerk en er dus ook geen operaties meer mogelijk zijn.

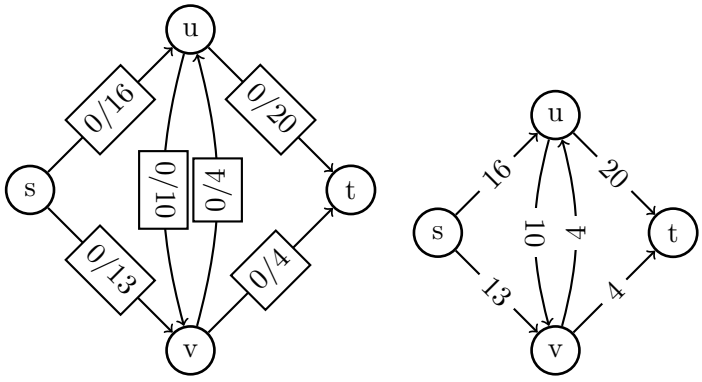
Algoritme 5 Pseudocode van het generieke prestroom-duw-algoritme.

```
1:  function PRESTROOM-DUW(G)
2:      INITIALISEER_PRESTROOM(G)

3:      while ∃ een actieve knoop u do
4:          if ∃v waarvoor (u, v) ∈ E_f ∧ h(u) = h(v) + 1 then
5:              DUW(u,v)
6:          else
7:              VERHOOG(u)
8:          end if
9:      end while

10:     return f
11: end function
```

We illustreren nu de werking van het generieke prestroom-duw-algoritme op het stroomnetwerk uit Figuur 6.1(a). In Figuur 6.1(b) zien we het bijhorende residueel stroomnetwerk waarop we het prestroom-duw-algoritme volledig uitvoeren.



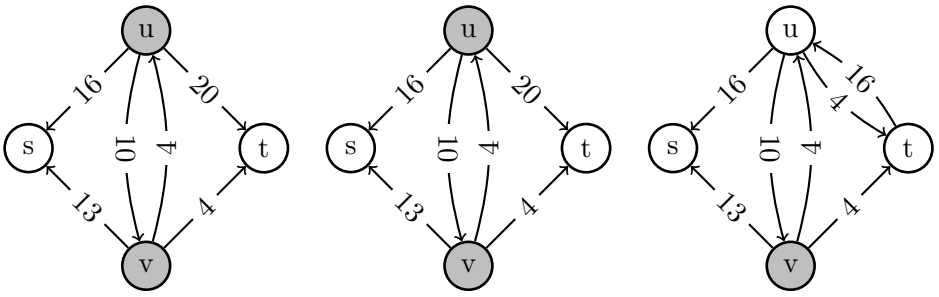
(a)

(b) $h(s) = 0$ $e(s) = 0$
$h(u) = 0$ $e(u) = 0$
$h(v) = 0$ $e(v) = 0$
$h(t) = 0$ $e(t) = 0$

Figuur 6.1: Voorbeeld uitvoering van het prestroom-duw-algoritme.

Wanneer we de prestroom initialiseren in het stroomnetwerk, krijgen we het residueel stroomnetwerk uit Figuur 6.2(a). We zien hierbij ook dat knopen u en v actief geworden zijn en we duiden dit in de figuren steeds aan met een ingekleurd bolletje.

Vervolgens nemen we bijvoorbeeld knoop u en verhogen deze naar $h(u) = 1$, zie Figuur 6.2(b). Bijgevolg sturen we vanuit knoop u zestien eenheden



(a) **h(s) = 4** **e(s) = -29**
$h(u) = 0$ **e(u) = 16**
$h(v) = 0$ **e(v) = 13**
$h(t) = 0$ $e(t) = 0$

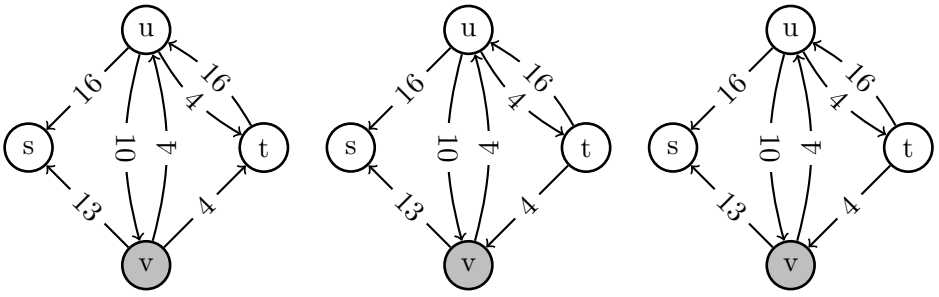
(b) $h(s) = 4$ $e(s) = -29$
h(u) = 1 $e(u) = 16$
$h(v) = 0$ $e(v) = 13$
$h(t) = 0$ $e(t) = 0$

(c) $h(s) = 4$ $e(s) = -29$
$h(u) = 1$ **e(u) = 0**
$h(v) = 0$ $e(v) = 13$
$h(t) = 0$ **e(t) = 16**

Figuur 6.2: Voorbeeld uitvoering van het prestroom-duw-algoritme.

stroom naar t. Het resultaat hiervan is het residueel stroomnetwerk uit Figuur 6.2(c) met slechts één actieve knoop meer, nl. knoop v.

Om nu vanuit knoop v stroom te kunnen duwen, dienen we deze te verhogen naar een hoogte van $h(v) = 1$, zie Figuur 6.3(a). Na de verhoogoperatie op v wordt het mogelijk om vanuit deze knoop stroom te duwen. Vanwege de beperkte residuele capaciteit van boog (v, t) is knoop v nog steeds actief na een duwoperatie vanuit v naar t. Het resultaat van de duwoperatie is zichtbaar in Figuur 6.3(b). Vervolgens kunnen we pas een duwoperatie doen vanuit v naar u na een verhoging van v naar $h(v) = 2$. Het resultaat van de verhoging is zichtbaar in Figuur 6.3(c) en het resultaat van de duwoperatie is zichtbaar in Figuur 6.4(a). Knoop v blijft actief en knoop u wordt, als gevolg van de zonet uitgevoerde duwoperatie, terug actief.



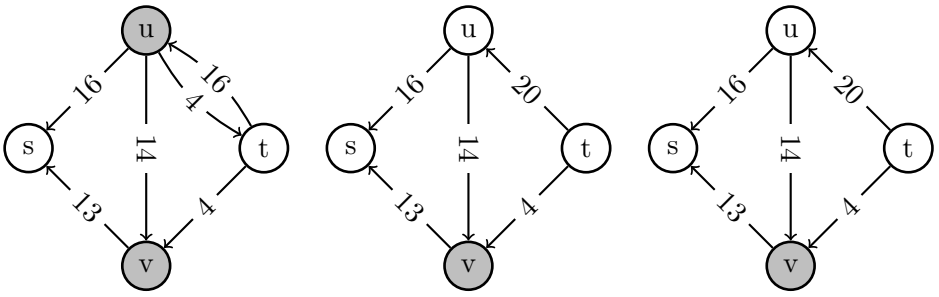
(a) $h(s) = 4$ $e(s) = -29$
$h(u) = 1$ $e(u) = 0$
h(v) = 1 $e(v) = 13$
$h(t) = 0$ $e(t) = 16$

(b) $h(s) = 4$ $e(s) = -29$
$h(u) = 1$ $e(u) = 0$
$h(v) = 1$ **e(v) = 9**
$h(t) = 0$ **e(t) = 20**

(c) $h(s) = 4$ $e(s) = -29$
$h(u) = 1$ $e(u) = 0$
h(v) = 2 $e(v) = 9$
$h(t) = 0$ $e(t) = 20$

Figuur 6.3: Voorbeeld uitvoering van het prestroom-duw-algoritme.

De vier eenheden stroom die door de laatst uitgevoerde duwoperatie als reststroom in u zijn aangekomen, duwen we vervolgens weer naar afvoer t.



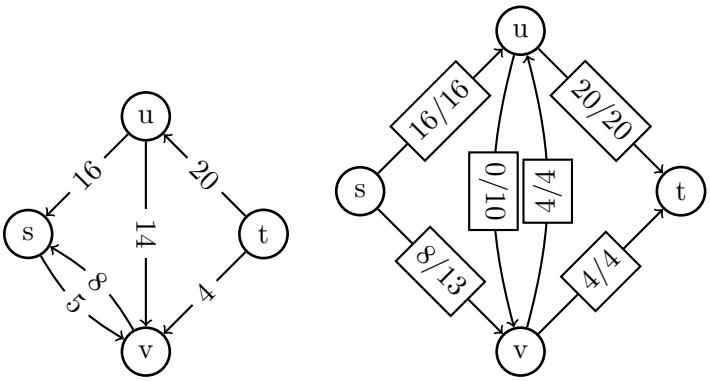
(a) $h(s) = 4$ $e(s) = -29$
$h(u) = 1$ $\mathbf{e(u) = 4}$
$h(v) = 2$ $\mathbf{e(v) = 5}$
$h(t) = 0$ $e(t) = 20$

(b) $h(s) = 4$ $e(s) = -29$
$h(u) = 1$ $\mathbf{e(u) = 0}$
$h(v) = 2$ $e(v) = 5$
$h(t) = 0$ $\mathbf{e(t) = 24}$

(c) $h(s) = 4$ $e(s) = -29$
$h(u) = 1$ $e(u) = 0$
$\mathbf{h(v) = 5}$ $e(v) = 5$
$h(t) = 0$ $e(t) = 24$

Figuur 6.4: Voorbeeld uitvoering van het prestroom-duw-algoritme.

Het resultaat hiervan is zichtbaar in Figuur 6.4(b). Knoop u is nu niet langer actief. Knoop v is momenteel de enige actieve knoop, maar is helemaal niet vatbaar voor een duwoperatie aangezien deze knoop enkel s als buur heeft binnen het residueel stroomnetwerk en $h(s) = 4 > h(v) = 2$. We voeren vervolgens een verhoogoperatie uit op v zodanig dat het teveel aan stroom dat er blijkbaar is, terug naar de bron kan geduwd worden. Het resultaat van beide operaties is zichtbaar in Figuur 6.4(c) en 6.5(a).



(a) $h(s) = 4$ $\mathbf{e(s) = -24}$
$h(u) = 1$ $e(u) = 0$
$h(v) = 5$ $\mathbf{e(v) = 0}$
$h(t) = 0$ $e(t) = 24$

(b)

Figuur 6.5: Voorbeeld uitvoering van het prestroom-duw-algoritme.

Er zijn nu geen actieve knopen meer in het residueel stroomnetwerk en dus weten we dat de uitvoering van het algoritme ten einde is en we een maximale stroom gevonden hebben met een waarde van 24, zie Figuur 6.5(b).

6.8 Correctheid

We willen nu aantonen dat het generieke prestroom-duw-algoritme bij beëindiging een maximale stroom f in G heeft gegenereerd. We bekijken hiervoor eerst een aantal stellingen en hun bewijzen. Dat het algoritme effectief eindigt, tonen we pas aan in Sectie 6.9. Pas na afloop van de laatst vernoemde sectie kunnen we een oordeel vellen over het feit of het generieke prestroom-duw-algoritme het maximumstroomprobleem effectief oplost.

Stelling 6.5. *Wanneer knoop u een actieve knoop is, met andere woorden $e(u) > 0$, kunnen we op deze knoop ofwel een duwoperatie ofwel, indien dit niet gaat, een verhoogoperatie toepassen.*

Bewijs. We weten dat h een hoogtefunctie is en er bijgevolg voor iedere boog (u, v) in het residueel stroomnetwerk geldt dat $h(u) \leq h(v) + 1$. We weten verder, zoals reeds vermeld, dat we enkel mogen duwen vanuit u indien $h(u) = h(v) + 1$. Stel nu dat we niet kunnen duwen vanuit u, dan geldt er voor alle bogen (u, v) in het residueel stroomnetwerk dat $h(u) < h(v) + 1$ en bijgevolg $h(u) \leq h(v)$. Vervolgens kan er, per definitie, een verhoogoperatie worden toegepast op u. Indien er geen verhoogoperatie kan worden uitgevoerd op u, moet er gelden dat $h(u) > h(v)$ voor alle buren v van u in het residueel stroomnetwerk. Vanwege het feit dat h een hoogtefunctie is, kan $h(u)$ niet anders dan gelijk zijn aan $h(v)+1$ aangezien we anders de eigenschap $h(u) \leq h(v) + 1$ schenden.

We hebben in dit bewijs impliciet gebruik gemaakt van het feit dat u steeds buren heeft in het residueel stroomnetwerk. Waarom is dit zo? – Wel, wanneer een knoop actief is, wil dit zeggen dat er stroom naar deze knoop geduwd werd en niet is weggeraakt. Er kan echter steeds stroom teruggeduwd worden naar de knopen waarvan de stroom kwam. De knoop heeft dus sowieso minstens één buur in het residueel stroomnetwerk. Merk op dat Eigenschap 6.3 ook steunt op dit feit. □

Stelling 6.6. *Tijdens de uitvoering van het algoritme geldt er voor iedere knoop $u \in V$ dat zijn hoogte $h(u)$ nooit afneemt. De hoogte $h(u)$ neemt met tenminste één eenheid toe nadat er een verhoogoperatie op is toegepast.*

Bewijs. De hoogte van een knoop kan enkel wijzigen wanneer we er een verhoogoperatie op toepassen. Wanneer we een knoop u verhogen, weten we voor u en elke buur v in het residueel stroomnetwerk dat $h(u) \leq h(v)$. Bijgevolg geldt er vóór de verhoogoperatie dat $h(u) < 1+min\{h(v) : (u, v) \in E_f\}$. In de verhoogoperatie wordt de hoogte van u gelijkgesteld aan het rechterlid van voorgaande vergelijking en dus stijgt de hoogte met minstens één eenheid. □

Stelling 6.7. *Tijdens de uitvoering van het algoritme gelden steeds de eigenschappen van de hoogtefunctie h.*

Bewijs. De verhoogoperatie zorgt ervoor dat de hoogtefunctie h na zijn uitvoering nog steeds een hoogtefunctie is. De verhoogoperatie gaat steeds de hoogte van een knoop aanpassen naar één eenheid hoger dan de hoogte van diens laagste buur in het residueel stroomnetwerk. Er is dus geen buur die twee niveau's lager ligt na een verhoogoperatie. De hoogte van de bron en de afvoer veranderen nooit, aangezien ze immers nooit actief zijn. Een duwoperatie verandert geen hoogtes van knopen en de enige bogen die een duwoperatie kan introduceren in het residueel stroomnetwerk gaan enkel van laag naar hoog. De hoogtefunctie h is dus invariant onder de duwoperatie.

Tijdens de uitvoering van het algoritme vinden enkel verhoog- en duwoperaties plaats. De hoogtefunctie h wordt gerespecteerd onder beide en dus ook doorheen de uitvoering van het algoritme. □

Stelling 6.8. *Er bestaat geen simpel pad van de bron s naar de afvoer t in het residueel stroomnetwerk G_f onder prestroom f_{pre}.*

Bewijs. Na de initialisatie van f_{pre} geldt er dat $deg^-(s) = 0$ in G_f. Er zijn op dat moment dus geen simpele paden vertrekkend uit s aangezien deze geen uitgaande bogen heeft. Dit is echter geen garantie dat er gedurende en bij een beëindiging van de uitvoering van het algoritme ook geen toenemende paden zijn in G_f.

Veronderstel dat er gedurende de uitvoering wél een simpel pad p bestaat van s naar t in G_f, met $p = (v_0, \ldots, v_k)$, $v_0 = s$ en $v_k = t$. Vanwege het feit dat p een simpel pad is, weten we dat $k \leq n - 1$ en bijgevolg $k < n$. Verder weten we nu ook dat $\forall i \in \{0, \ldots, k-1\} : (v_i, v_{i+1}) \in E_f$. We kunnen nu, vanwege de beperkingen die hoogtefunctie h oplegt aan G_f, heel eenvoudig bewijzen dat het veronderstelde een contradictie vormt. Vanwege het feit dat h een hoogtefunctie is, geldt het volgende: $\forall i = 0, \ldots, k-1 : h(v_i) \leq h(v_{i+1}) + 1$. Wanneer we dit nu iteratief toepassen over p, verkrijgen we de volgende ongelijkheid: $h(s) \leq h(t) + k$. Vanwege h geldt er dat $h(t) = 0$ en bijgevolg $h(s) \leq k < n$. Dit vormt echter een contradictie met het feit dat $h(s) = n$, volgens de regels van h. We besluiten dus dat er geen simpel pad van s naar t bestaat in G_f onder f_{pre}. □

Stelling 6.9. *Het generieke prestroom-duw-algoritme genereert, bij beëindiging, een maximale stroom in het betrokken stroomnetwerk.*

Bewijs. Wanneer het generieke prestroom-duw-algoritme eindigt, is elke intermediaire knoop inactief. Indien dit niet zo zou zijn, zou het algoritme nooit geëindigd hebben aangezien we dan vanwege Stelling 6.5 sowieso ofwel een duwoperatie ofwel een verhoogoperatie op een bepaalde intermediaire knoop kunnen toepassen en er een nieuwe iteratie zou worden opgestart. Vermits nu iedere intermediaire knoop inactief is, wordt f_{pre} een geldige stroom f. De eigenschappen van h zijn vanwege Stelling 6.7 bewaard gebleven tijdens de uitvoering van het algoritme. Vanwege Stelling 6.8 bestaat er

nu geen toenemend pad in het uiteindelijke residueel stroomnetwerk G_f. We kunnen nu de Maximumstroom-minimumboogsnede-stelling (Stelling 5.2) toepassen waaruit volgt dat f een maximale stroom is [2]. □

6.9 Analyse

We tonen in deze sectie aan dat het generieke prestroom-duw-algoritme effectief eindigt. We doen dit door het totaal aantal operaties dat het algoritme uitvoert te begrenzen, waarbij we de duwoperaties opsplitsen in verzadigde en onverzadigde. Kennis over het aantal keren dat de drie operaties afzonderlijk maximaal worden uitgevoerd, zal ons uiteindelijk een begrenzing opleveren voor het totaal aantal operaties.

Vooraleer we overgaan tot het begrenzen van de aantallen keren dat de drie operaties afzonderlijk kunnen worden uitgevoerd, introduceren we een aantal stellingen, nodig om de analyses die daarop volgen te kunnen uitvoeren.

Stelling 6.10. *Voor iedere actieve knoop u is er een simpel pad van u naar s in het residueel stroomnetwerk G_f.*

Bewijs. We weten dus dat knoop u een actieve knoop is en bijgevolg een reststroom, $e(u)$, bezit. Zij $U = \{u\} \cup \{v : \exists \text{ simpel pad van } u \text{ naar } v \text{ in } G_f\}$ en veronderstel dat $s \notin U$. Neem nu de verzameling van knopen $\overline{U} = V \setminus U$. We tonen aan dat als $w \in \overline{U}$ en $v \in U$, dat dan $f(w, v) \leq 0$.

Stel immers dat $f(w, v) > 0$, dan is $f(v, w) < 0$ vanwege de regel der annulering van stromen. Dit zorgt er op zijn beurt voor dat er een boog is van v naar w in het residueel stroomnetwerk, aangezien $c_f(v, w) = c(v, w) - f(v, w) > 0$. Uit $v \in U$ volgt dat er een simpel pad is van u naar v en dus ook een simpel pad van u naar w via de boog (v, w). Dit is echter een contradictie met $w \in \overline{U}$, dus geldt er dat $f(w, v) \leq 0$.

Wanneer we nu de volgende vergelijking bekijken:

$$f\left[\overline{U}, U\right] = \sum_{\substack{(w,v) \in (\overline{U}, U) \\ f(w,v) > 0}} f(w, v) - \sum_{\substack{(v,w) \in (U, \overline{U}) \\ f(v,w) > 0}} f(v, w), \tag{6.3}$$

weten we dat de eerste term van de vergelijking nul is, aangezien steeds geldt dat $f(w, v) \leq 0$ en dus niet in de sommatie van de eerste term zal worden opgenomen. Bijgevolg, en mede vanwege het feit dat ook $f(v, w) > 0$ voor de tweede term, geldt het volgende: $f\left[\overline{U}, U\right] \leq 0$.

Het vorige feit impliceert nu dat er evenveel of minder positieve stroom over de bogen van $\overline{U}$ naar U stroomt dan positieve stroom van U naar $\overline{U}$ en dus dat $e(U) \leq 0$.

Knopen die een element zijn van $V \setminus \{s\}$, hebben, per definitie, enkel niet-negatieve reststroom. We hebben echter aangenomen dat $U \subseteq V \setminus \{s\}$,

waaruit volgt dat $e(U) = 0$. Dit is echter onmogelijk aangezien $u \in U$ en $e(U) > 0$; u is immers actief. Bijgevolg moet er gelden dat $s \in U$. □

Stelling 6.11. *De hoogte van een knoop is hoogstens* $2n - 2$*:*

$$\forall u \in V : h(u) \leq 2n - 2.$$

Bewijs. De hoogte van s en t worden initieel op n, resp. 0 gezet en veranderen nooit aangezien s en t nooit actief worden.

Wanneer knoop $u \in V \setminus \{s, t\}$ verhoogd wordt, moet hij actief zijn. We weten dan uit Stelling 6.10 dat er een simpel pad $p = (v_0, \ldots, v_k)$ bestaat vanuit u naar s in G_f, met $u = v_0$ en $s = v_k$. Daarenboven weten we ook dat $k \leq n-2$ aangezien p een simpel pad is en bijgevolg niet meer bogen kan hebben dan $n - 2$. Voor de bogen $(v_i, v_{i+1}) \in G_f$ op p met $i = 0, \ldots, k - 1$ geldt er vanwege Definitie 6.2 dat $h(v_i) \leq h(v_{i+1}) + 1$ (*). De hoogtefunctie die we hiervoor gebruiken, blijft gerespecteerd doorheen de uitvoering van het algoritme, zie Stelling 6.7. Wanneer we nu al de ongelijkheden over p combineren, krijgen we de volgende ongelijkheid: $h(v_0) \leq h(v_k) + k \leq h(v_k) + n - 2 = h(s) + n - 2 = n + n - 2 = 2n - 2$. De term $h(v_k) + k$ in de ongelijkheid vloeit voort uit een iteratieve toepassing van (*). We weten nu reeds, per definitie, dat $h(v_0) = h(u)$ en dus weten we dat $h(u) \leq 2n - 2$. Met andere woorden, de hoogte van elke intermediaire knoop is steeds kleiner dan of gelijk aan $2n - 2$. □

6.9.1 Verhoogoperaties

De hoogte van een intermediaire knoop start initieel op nul en verhoogt met minstens één eenheid per verhoogoperatie die er op wordt uitgevoerd. Uit Stelling 6.11 volgt dus onmiddellijk dat er hoogstens $(n - 2)(2n - 2)$ verhoogoperaties kunnen uitgevoerd worden.

Stelling 6.12. *Het totaal aantal verhoogoperaties tijdens de uitvoering van het algoritme is* $O(n^2)$*.*

6.9.2 Verzadigde duwoperaties

We onderzoeken nu vervolgens hoeveel verzadigde duwoperaties er tussen elk paar knopen $u, v \in V$ kunnen plaatsvinden. Deze bestaan uit de som van het aantal duwoperaties van u naar v en van v naar u. Merk op dat er enkel duwoperaties tussen u en v kunnen plaatsvinden indien (u, v) of $(v, u) \in E_f$. Stel nu dat er een verzadigde duwoperatie heeft plaatsgevonden vanuit u naar v dan weten we dat $h(u) = h(v) + 1$. Door de verzadigde duwoperatie zal boog (u, v) natuurlijk uit G_f verdwijnen. Deze boog zal zijn intrede pas weer doen indien er een duwoperatie plaatsvindt vanuit v naar u. Dit kan pas gebeuren vanaf het moment dat $h(v) = h(u)+1$. Om dit te bekomen moet de hoogte van v toenemen met tenminste twee eenheden. We kunnen dit alles

ook omgekeerd bekijken en dan is het de hoogte van u die moet toenemen met tenminste twee eenheden. We starten steeds met hoogtes gelijk aan nul en we weten uit Stelling 6.11 dat deze nooit groter worden dan $2n - 2$. Een knoop kan dus bijgevolg maximaal net geen n keer zijn hoogte telkens verhogen met twee eenheden. Verzadigde duwoperaties wisselen af tussen knoop u en v en beide knopen verhogen hiervoor telkens afwisselend hun hoogte met twee. Dit laatste kunnen beide maximaal n keer doen en dus is het aantal verzadigde duwoperaties tussen beide knopen $O(n)$. Per boog in het originele stroomnetwerk kunnen er met andere woorden hoogstens $O(n)$ verzadigde duwoperaties plaatsvinden. Dit leidt ons tot de volgende stelling:

Stelling 6.13. *Het totaal aantal verzadigde duwoperaties tijdens de uitvoering van het algoritme is $O(nm)$.*

6.9.3 Onverzadigde duwoperaties

Er rest ons nog één ding te onderzoeken: een grens op het aantal onverzadigde duwoperaties.

We introduceren hiervoor de volgende functie:

$$H = \sum_{\{v \in V \setminus \{s,t\} : e(v) > 0\}} h(v).$$

We zullen nu aantonen dat de onverzadigde duwoperatie, met uitzondering van een specifieke verzadigde duwoperatie, de enige operatie is die H kan doen dalen. We weten dat een uitvoering van het algoritme start en eindigt met $H = 0$. Bijgevolg kunnen we het aantal onverzadigde duwoperaties begrenzen door de totale som van toenames van H te begrenzen.

Toenames van H worden enkel veroorzaakt door verhoogoperaties en verzadigde duwoperaties:

- Verhoogoperaties doen de hoogte van een knoop steeds stijgen en dit tot een maximale hoogte van $2n - 2$. Dit is bijgevolg tevens de maximale som van toenames van H door verhoogoperaties op één knoop, aangezien een actieve knoop v na een verhoogoperatie nog steeds actief is. We hebben in totaal $n-2$ knopen en bijgevolg is de totale som van toenames van H, veroorzaakt door verhoogoperaties, begrensd door $(2n - 2)(n - 2) = 2n^2 - 6n + 4$.

- Stel dat we een verzadigde duwoperatie doen vanuit knoop u naar knoop v. Enkel indien knoop v inactief was vóór de duwoperatie en $v \neq s \wedge v \neq t$, zal deze zijn hoogte, die maximaal $2n-3$ is, bijdragen tot de waarde van H. Eén verzadigde duwoperatie kan de waarde van H dus maximaal met $2n-3$ doen toenemen. We hebben echter hoogstens nm verzadigde duwoperaties, zie Stelling 6.13, en bijgevolg is de totale som van toenames van H, veroorzaakt door verzadigde duwoperaties, begrensd door $nm(2n - 3)$.

De totale som van toenames van H is bijgevolgd begrensd door $2n^2 - 6n + 4 + nm(2n-3) = 2n^2m - 3nm + 2n^2 - 6n + 4 = O(n^2m)$.

Een knoop u kan echter door een verzadigde duwoperatie ook inactief worden, waardoor de waarde van H afneemt, aangezien de bijdrage van u, die steeds groter is dan die van v, verdwijnt. Hoe sterk de waarde van H in dit geval afneemt, is vergelijkbaar met de afname van de waarde van H, veroorzaakt door een onverzadigde duwoperatie.

We tonen aan dat elke onverzadigde duwoperatie de waarde van H met minstens één eenheid doet afnemen.

Vóór de aanvang van een onverzadigde duwoperatie vanuit knoop u naar knoop v was u actief, anders kan er immers geen duwoperatie plaatsvinden. Knoop v kan zowel actief als inactief zijn. Wanneer we dan de onverzadigde duwoperatie uitvoeren, weten we dat u sowieso geen reststroom meer heeft en enkel de hoogte van v mogelijk een extra bijdrage zal leveren aan de waarde van H.

Laten we de nieuwe waarde van H definiëren als $H_{new} = H - h(u) + h(v)$. We weten nu ook dat de duwoperatie mogelijk werd aangezien $h(u) - h(v) = 1$. In het geval v inactief was en $v \neq s \wedge v \neq t$ neemt de waarde van H dus met exact één eenheid af. In het geval v reeds actief was of $v = s$ of $v = t$, geldt er: $H_{new} = H - h(u)$. We besluiten hieruit dat elke onverzadigde duwoperatie de waarde van H met minstens één eenheid doet afnemen.

We weten nu genoeg om het aantal onverzadigde duwoperaties langs boven te begrenzen. De waarde van H is initieel gelijk aan nul. Tijdens de uitvoering van het algoritme is $H > 0$ en kan stijgen of dalen. Op het einde van het algoritme moet H weer gelijk zijn aan nul. Alle toenames van de waarde van H moeten er bijgevolg ook weer af. We hebben reeds gezien dat de totale som van toenames van H begrensd is door $2n^2m - 3nm + 2n^2 - 6n + 4$. Een onverzadigde duwoperatie doet de waarde van H met minstens één eenheid dalen. Bijgevolg zijn er, in het slechtste geval, net zoveel onverzadigde duwoperaties nodig als de totale som van toenames van H, om de waarde van H tegen het einde van de uitvoering van het algoritme weer naar nul te brengen. Het totaal aantal onverzadigde duwoperaties, tijdens de uitvoering van het algoritme, is dus $O(n^2m)$.

Indien de waarde van H afneemt door verzadigde duwoperaties, zorgt dit er gewoon voor dat er voor elke verzadigde duwoperatie één onverzadigde duwoperatie minder nodig is. Dit verandert dus niets aan de bekomen bovengrens voor het aantal onverzadigde duwoperaties.

Stelling 6.14. *Het totaal aantal onverzadigde duwoperaties, tijdens de uitvoering van het generieke prestroom-duw-algoritme, is $O(n^2m)$.*

6.9.4 Begrenzing op het totaal aantal basisoperaties

We berekenen nu een bovengrens op het totaal aantal operaties aan de hand van het aantal basisoperaties per type. Voor een overzicht verwijzen we naar

Tabel 6.1.

Operatie	Aantal operaties
Verhoogoperatie	$O(n^2)$
Verzadigde duwoperatie	$O(nm)$
Onverzadigde duwoperatie	$O(n^2m)$

Tabel 6.1: Bovengrenzen voor het aantal keer elke operatie wordt uitgevoerd.

De totale som van al de aantallen basisoperaties is $O(n^2 + n^2m + nm)$.

Stelling 6.15. *Het totaal aantal operaties tijdens de uitvoering van het generieke prestroom-duw-algoritme is $O(n^2m)$ en wordt gedomineerd door het aantal onverzadigde duwoperaties die plaatsvinden.*

Uit Stelling 6.15 en Stelling 6.9 besluiten we tenslotte dat het generieke prestroom-duw-algoritme het maximumstroomprobleem oplost.

6.9.5 Ontladen van een actieve knoop

Het generieke prestroom-duw-algoritme selecteert gedurende zijn uitvoering een willekeurige actieve knoop en past vanuit deze knoop een onverzadigde- of verzadigde duwoperatie toe, of verhoogt de knoop. Na een verzadigde duwoperatie kan de knoop nog steeds actief zijn. Het algoritme is echter niet verplicht om deze knoop ook in een volgende iteratie te selecteren.

We kunnen er echter voor zorgen dat het algoritme, bij selectie van een actieve knoop v, stroom blijft sturen vanuit deze knoop tot op het moment dat zijn reststroom, $e(v)$, nul is, of de knoop verhoogd moet worden. Het algoritme voert hiervoor mogelijk meerdere verzadigde duwoperaties uit, gevolgd door een onverzadigde duwoperatie of een verhoogoperatie. Deze opeenvolging van operaties, op een actieve knoop, definiëren we als de *ontlading* van deze knoop. Indien $e(v) = 0$ na een ontlading van v, spreken we van een *volledige* ontlading.

De manier van ontladen, in de vorige paragraaf, levert ons ten hoogste één verhoging van v op per ontlading en is de *standaard strategie* voor het uitvoeren van een ontlading van een actieve knoop.

Naast deze standaard strategie, bestaan er nog twee andere strategieën om duw- en verhoogoperaties met elkaar te combineren tot de ontlading van een actieve knoop:

- We passen een duwoperatie toe op v tot $e(v) = 0$ of tot op het moment dat v verhoogd is en er minstens één keer stroom vanuit v geduwd werd. Toepassing van een dergelijke ontlading levert ons ten hoogste twee verhogingen van een knoop op per ontlading.

- We passen duw-en/of verhoogoperaties toe op v tot $e(v) = 0$. Wanneer we een dergelijke ontlading toepassen op een knoop, levert ons dit mogelijk meerdere verhogingen van deze knoop op per ontlading. Deze manier van ontladen kan zorgen voor een vroegtijdige identificatie van knopen die een reststroom bevatten die sowieso weer naar de bron moet afvloeien.

6.9.6 Tijdscomplexiteit van het generieke prestroom-duw-algoritme

Tot nu toe hebben we de basisoperaties begrensd, maar nog niet de tijdscomplexiteit van de basisoperaties zelf. In Algoritme 3 en 4 werd reeds pseudocode gegeven voor de duw- en verhoogoperatie.

Een duwoperatie (verzadigd/onverzadigd) vereist het bepalen van een minimum tussen reststroom en residuele capaciteit. De reststroom kan in constante tijd worden opgevraagd, via een tabel waarin we per knoop zijn reststroom bijhouden, en de residuele capaciteit kan in constante tijd berekend worden. Het berekenen van de residuele capaciteit gebeurt uit de stroom en capaciteit tussen de twee betrokken knopen. Beide gegevens voor alle paren knopen bewaren we in een adjacentiematrix, zodat ze opvraagbaar zijn in constante tijd. De reststromen kunnen vervolgens ook weer in constante tijd worden aangepast en ook de stroom tussen twee knopen.

Om de verhoogoperatie efficiënt te implementeren, definiëren we voor iedere knoop $v \in V \setminus \{s, t\}$ een lijst $L(v) = \{w \in V | (v, w) \in E \vee (w, v) \in E\}$. Deze lijst bevat de eventuele residuele buren w van v. De lengte van een lijst voor knoop v benoemen we met l_v. De totale lengte van alle lijsten samen benoemen we met $l = l_{v_1} + \cdots + l_{v_{n-2}}$, waarbij $\{v_1, \ldots, v_{n-2}\} = V \setminus \{s, t\}$.

l kan gelijk zijn aan $2m$ indien bogen slechts in één richting voorkomen tussen alle knopen die in het stroomnetwerk origineel met elkaar verbonden zijn. l kan ook gelijk zijn aan m indien bogen in de twee richtingen voorkomen tussen alle knopen die in het stroomnetwerk origineel met elkaar verbonden zijn. We zien bijgevolg het volgende in: $m \leq l \leq 2m$.

Bij het uitvoeren van een verhoogoperatie op een knoop v, lopen we lijst $L(v)$ helemaal af, op zoek naar de buren van v in het residueel stroomnetwerk. De lengte van een dergelijke lijst is gelijk aan l_v. Het opvragen en aanpassen van de hoogte van een knoop kost ons sowieso maar $O(1)$ tijd, aangezien we een tabel bijhouden met deze hun hoogtes, en bijgevolg geldt er:

Stelling 6.16. *Een duwoperatie (verzadigd/onverzadigd) is uitvoerbaar in $O(1)$ tijd. Een verhoogoperatie is uitvoerbaar in $O(l_v)$ tijd.*

Een implementatie van de standaard strategie van ontladen van een actieve knoop v is volledig uitgewerkt in Algoritme 6. Merk op dat we ook hier gebruik maken van de lijst $L(v)$ bij het zoeken van toegelaten bogen om

vervolgens stroom over te duwen. Indien er een onverzadigde duwoperatie plaatsvindt, wordt er bij een volgende ontlading van v verdergegaan met de residuele boog die aanleiding gaf tot deze onverzadigde duwoperatie en dit tot en met de laatste knoop van $L(v)$.

Algoritme 6 Pseudocode van de standaard strategie om een actieve knoop te ontladen.

```
1:  procedure ONTLADEN(v)
2:      w ← de huidige knoop van L(v)
3:      tijdOmTeVerhogen ← false

4:      while e(v) ≠ 0 and !tijdOmTeVerhogen do
5:          if (v, w) is een toegelaten boog then
6:              DUW(v,w)
7:          else
8:              if w is niet de laatste knoop van L(v) then
9:                  de volgende knoop van L(v) wordt diens huidige knoop
10:                 w ← de huidige knoop van L(v)
11:             else
12:                 de eerste knoop van L(v) wordt de huidige knoop w
13:                 tijdOmTeVerhogen ← true
14:             end if
15:         end if
16:     end while

17:     if tijdOmTeVerhogen then
18:         VERHOOG(v)
19:     end if
20: end procedure
```

Waarom wordt er in dit geval enkel doorgegaan tot en met de laatste knoop van $L(v)$, alvorens een verhoogoperatie uit te voeren, en niet weer volledig rond tot en met de knoop net vóór de knoop vanaf waar we begonnen zijn? Stel dat knoop u de kop is van de residuele boog die aanleiding gaf tot de onverzadigde duwoperatie. Er kan nu pas een residuele boog van v naar de knopen vóór u zijn indien er ondertussen een stroom vanuit deze knopen naar v gestuurd werd. Hiervoor moeten de hoogtes van deze knopen gelijk zijn aan $h(u) + 2$, waardoor v weer pas naar deze knopen kan duwen na een verhoogoperatie en waardoor het niet nodig is om vóór een verhoogoperatie binnen de ontlading deze knopen nog eerst te beschouwen.

Merk op dat de ontladingsoperatie enkel een knoop v verhoogt indien $e(v) > 0$ en voor alle buren w in het residueel stroomnetwerk geldt dat $h(v) \leq h(w)$.

In de ontlading van een actieve knoop v kan er een onderscheid gemaakt

worden tussen de uitvoering van:

1. (eventuele) verzadigde duwoperaties + sowieso een verhoogoperatie,
2. (eventuele) verzadigde duwoperaties + sowieso een onverzadigde duwoperatie.

We weten reeds dat er in totaal $O(nm) + O(n^2m)$ duwoperaties plaatsvinden. Vanwege Stelling 6.16 besluiten we dat het dus ook zoveel tijd in beslag neemt om deze duwoperaties uit te voeren.

Het aantal verhoogoperaties per knoop v is $O(n)$. De tijd, nodig om alle verhoogoperaties op een knoop v uit te voeren, is, mede vanwege Stelling 6.16, gelijk aan $O(nl_v)$. We sommeren deze tijd vervolgens over alle knopen $v \in V \setminus \{s, t\}$. Dit geeft als resultaat $O(nl)$. Om alle verhoogoperaties uit te voeren, hebben we bijgevolg $O(nm)$ tijd nodig.

De invoeging van de ontladingsoperatie in het generieke prestroom-duw-algoritme, leidt tot het generieke prestroom-duw-algoritme met standaard ontlaadstrategie zoals uitgewerkt in Algoritme 7. Wanneer we daarenboven veronderstellen dat het selecteren (willekeurig kiezen kan in $O(1)$ tijd) en onderhouden van de verzameling van actieve knopen $O(1)$ tijd kost, komen we tot de volgende stelling:

Stelling 6.17. *Het generieke prestroom-duw-algoritme met standaard ontlaadstrategie eindigt in $O(n^2m)$ tijd en lost het maximumstroomprobleem op.*

Algoritme 7 Pseudocode van het generieke prestroom-duw-algoritme met standaard ontlaadstrategie.

```
1: function PRESTROOM-DUW(G)
2:     INITIALISEER_PRESTROOM(G)

3:     while ∃ een actieve knoop v do
4:         ONTLADEN(v)
5:     end while

6:     return f
7: end function
```

6.10 Knoopselectiestrategieën

Merk op dat we de volgorde waarin we actieve knopen selecteren steeds hebben opengelaten. De strategie waarmee actieve knopen gekozen worden,

de zogenaamde *knoopselectiestrategie*, bepaalt een specifiek prestroom-duw-algoritme dat gebaseerd is op het generieke prestroom-duw-algoritme met standaard ontlaadstrategie.

Dergelijke selectiestrategieën proberen, door een goede keuze van actieve knoop, zo weinig mogelijk onverzadigde duwoperaties uit te voeren gedurende de uitvoering van het algoritme en zo de uitvoeringstijd van een prestroom-duw-algoritme naar beneden te halen. De totale uitvoeringstijd van onverzadigde duwoperaties is immers bepalend voor de tijdsgrens van een prestroom-duw-algoritme.

We bestuderen in dit werk drie specifieke prestroom-duw-algoritmen:

1. *FIFO prestroom-duw-algoritme*,
2. *Hoogste-hoogte-eerst prestroom-duw-algoritme*,
3. *Excess scaling prestroom-duw-algoritme.*

We bespreken voor elk prestroom-duw-algoritme zijn werking en tijdscomplexiteit. Merk op dat hun correctheid direct volgt uit de correctheid van het generieke prestroom-duw-algoritme met standaard ontlaadstrategie omdat ze er enkel van afwijken in de knoopselectiestrategie.

6.10.1 FIFO prestroom-duw-algoritme

Het FIFO prestroom-duw-algoritme [6][7] ontlaadt actieve knopen in een *First-In First-Out* volgorde. De actieve knopen worden hiertoe in een lijst Q[1] bijgehouden en het algoritme ontlaadt steeds de eerste knoop van Q. Indien de ontlading van een actieve knoop eindigt met een verhoogoperatie, is de knoop nog steeds actief en plaatsen we deze achteraan in Q. In het andere geval wordt de knoop inactief en verwijderen we deze uit Q. Knopen die tijdens de uitvoering van het algoritme actief worden, voegen we achteraan Q toe. Het algoritme eindigt wanneer Q leeg is.

Analyse

Bij het uitvoeren van een worst-case tijdscomplexiteitsanalyse van het FIFO prestroom-duw-algoritme delen we het totaal aantal ontladingen van actieve knopen op in *fasen*. Fase 1 bestaat uit de ontladingen toegepast op de actieve knopen die aan Q zijn toegevoegd bij de initialisatie van de prestroom. Fase $i+1$ met $i \geq 1$ bestaat uit de ontladingen toegepast op actieve knopen die (terug) toegevoegd zijn aan Q tijdens fase i.

Per fase kan er hoogstens één ontlading per actieve knoop uit Q gebeuren. Per ontlading kan er hoogstens één onverzadigde duwoperatie plaatsvinden. We besluiten dus dat er hoogstens $n-2$ onverzadigde duwoperaties per fase

[1] queue

plaatsvinden, aangezien er hoogstens $n-2$ actieve knopen in Q aanwezig zijn.

Om nu het totaal aantal onverzadigde duwoperaties te bepalen, en dus tevens de worst-case tijdscomplexiteit van het FIFO prestroom-duw-algoritme, begrenzen we het aantal fasen. We definiëren hiervoor een *potentiaalfunctie* $\Phi = \max\{h(v) : v \text{ is een actieve knoop}\}$, waarvan we de waarde bekijken na elke fase.

Bij de start van het algoritme geldt er $\Phi = 0$.

Tijdens de uitvoering van het algoritme kan er per fase ofwel geen, ofwel minstens één verhoogoperatie optreden.

We weten dat er $O(n^2)$ verhoogoperaties zijn en dus weten we dat er hoogstens zoveel fases zijn waarin minstens één verhoogoperatie optreedt.

In het geval dat er tijdens een fase geen enkele verhoogoperatie gebeurt, begeeft de reststroom van iedere knoop die actief was aan het begin van de fase zich naar knopen met kleinere hoogtes, geen verhoging betekent immers een volledige ontlading. Deze actieve knopen worden bijgevolg inactief en dus daalt Φ met minstens één eenheid. Het aantal keer dat Φ kan toenemen, kan nooit hoger zijn dan het aantal keer dat een knoop verhoogd wordt. De som van de toenamen in Φ is dus $O(n^2)$. Nu is het verder zo dat er op het einde van het algoritme geldt dat $\Phi = 0$. Iedere toename moet er dus ook weer af en dit kan enkel bereikt worden door de toepassing van $O(n^2)$ fasen zonder verhoging.

We komen zo aan een totaal van $O(n^2)$ fasen. Per fase hebben we zoals reeds gezegd hoogstens $n-2$ onverzadigde duwoperaties en in het totaal dus $O(n^3)$, die allen uitvoerbaar zijn in tijd $O(1)$. Om alle verhoogoperaties en verzadigde duwoperaties uit te voeren hebben we twee maal $O(nm)$ tijd nodig.

Stelling 6.18. *Het FIFO prestroom-duw-algoritme heeft een uitvoeringstijd van $O(n^3)$ en lost het maximumstroomprobleem op.*

6.10.2 Hoogste-hoogte-eerst prestroom-duw-algoritme

Het hoogste-hoogte-eerst prestroom-duw-algoritme duwt steeds stroom vanuit een actieve knoop met de hoogste hoogte. Ook voor dit algoritme kunnen we een $O(n^3)$ tijdsgrens afleiden. Hiervoor beschrijven we eerst hoe we deze vorm van actieve knoopselectie het efficiënste kunnen implementeren.

We onderhouden een *array van lijsten* B_i, met $0 \leq i \leq 2n-2$. Een lijst B_i bevat alle actieve knopen v met $h(v) = i$. Voor deze array houden we een index b bij die de grootste hoogte aangeeft van een actieve knoop en is dus met andere woorden de index van de hoogste niet-lege B_i. Indien er geen actieve knoop is, zetten we $b = -1$.

Tijdens de uitvoering van het algoritme gaan we in elke iteratie een willekeurige knoop v verwijderen uit B_b. Vervolgens ontladen we deze knoop en passen de waarde van b indien nodig aan. We onderscheiden twee gevallen:

1. Indien er een verhoogoperatie is gebeurd, verhoog dan b naar $h(v)$. Knoop v was immers reeds een hoogste knoop;

2. Indien alle reststroom in knoop v verwijderd is, verlaag dan b als $B_b = \emptyset$. We onderzoeken hiervoor de lijsten B_{b-1}, B_{b-2}, ..., tot we een niet-lege lijst tegenkomen, of $b = -1$ bereiken.

Merk op dat de totale som van toenames van b $O(n^2)$ is. De totale som van afnames van b is dus ook $O(n^2)$. Bijgevolg is het onderzoek naar de eerste lager gelegen niet-lege lijst geen dominante operatie. Wanneer b na een bepaalde iteratie negatief wordt, betekent dit dat er geen actieve knopen meer zijn en stopt het algoritme.

Analyse

In principe kunnen we met dezelfde potentiaalfunctie Φ uit de analyse van het FIFO prestroom-duw-algoritme een $O(n^3)$ tijdsgrens bekomen. Hiervoor delen we de uitvoering opnieuw op in fasen en beschouwen een fase als een reeks opeenvolgende ontladingen die b onveranderd laten.

Er is echter een betere begrenzing bekend. Cheriyan en Maheshwari [6] zijn er in geslaagd om de grens voor de onverzadigde duwoperaties van dit algoritme te verlagen naar $O(n^2\sqrt{m})$. Ze zijn deze grens bekomen met behulp van een uitbalancering van het aantal onverzadigde duwoperaties voor een "goedkope" en een "dure" fase.

Stelling 6.19. *Het hoogste-hoogte-eerst prestroom-duw-algoritme heeft een uitvoeringstijd van $O(n^2\sqrt{m})$ en lost het maximumstroomprobleem op.*

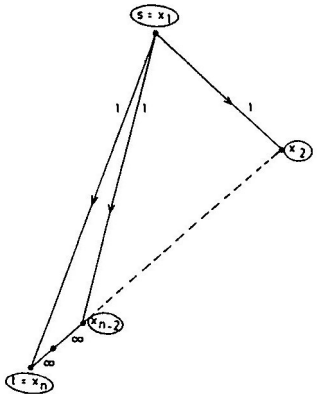
We sparen de lezer van verdere details vanwege het technische karakter van het bewijs, maar willen de lezer echter wel de intuïtie meegeven waarom deze knoopselectiestrategie tot minder onverzadigde duwoperaties leidt. We moeten voor ogen houden dat het FIFO prestroom-duw-algoritme een actieve knoop selecteert en zijn reststroom gewoon in de richting van de afvoer duwt. Het algoritme doet dit voor elke actieve knoop tot wanneer elke knoop inactief geworden is. Het hoogste-hoogte-eerst prestroom-duw-algoritme, daarentegen, voert steeds een selectie uit op de hoogste hoogte en duwt alle reststroom op deze hoogte naar een niveau lager en dit voor alle knopen op de hoogste hoogte. Alle reststroom stapelt zich dus in een niveau op en wordt steeds volledig naar een lager gelegen niveau, richting afvoer, weggewerkt. Bijgevolg vermijdt het algoritme herhaaldelijke duwoperaties met een kleine duwstroom. Dit vertaalt zich in de scherpere bovengrens voor het aantal onverzadigde duwoperaties.

6.10.3 FIFO vs. hoogste-hoogte-eerst

We zien dat de theoretische uitvoeringstijd van het hoogste-hoogte-eerst prestroom-duw-algoritme beter is dan de FIFO-implementatie. Uit empi-

risch onderzoek is gebleken dat een hoogste-hoogte-eerst implementatie in het algemeen dan ook het snelst een oplossing geeft. We merken echter het volgende op. Indien $m = \Theta(n^2)$, dan heeft het hoogste-hoogte-eerst prestroom-duw-algoritme een zelfde worst-case bovengrens als het FIFO prestroom-duw-algoritme.

We geven, gebaseerd op [10], ter illustratie, een voorbeeld van een stroomnetwerk waarop het hoogste-hoogte-eerst prestroom-duw-algoritme superieur is aan het FIFO prestroom-duw-algoritme met betrekking tot het aantal duwoperaties. Een dergelijk stroomnetwerk kunnen we construeren zoals aangegeven in Figuur 6.6.



Figuur 6.6: Voorbeeld van een stroomnetwerk waarop het hoogste-hoogte-eerst prestroom-duw-algoritme superieur is aan het FIFO prestroom-duw-algoritme.

Het stroomnetwerk bestaat uit een simpel pad van lengte $n-2$, in de richting van de afvoer. De afvoer is dus de laatste knoop op dit simpel pad. De bron komt overeen met de knoop met de laagste index, x_1, in het stroomnetwerk, de afvoer met de knoop met de hoogste index, x_n. Des te verder de knopen op het simpel pad verwijderd zijn van de afvoer, des te lager hun index. Elke boog op het simpel pad heeft een oneindige capaciteit en is gericht in de richting van de afvoer. De bron is verbonden met elke knoop op het simpel pad en dit met een capaciteit gelijk aan één.

Wanneer we nu beide algoritmen uitvoeren op dit geconstrueerde stroomnetwerk, merken we een groot verschil op met betrekking tot het aantal duwoperaties dat tijdens hun uitvoering plaatsvindt. Merk op dat dit verschil echter enkel wordt bekomen indien, bij het initialiseren van de prestroom, de knopen $x_2, \ldots, x_{n-1}$ volgens hun indexering als actieve knoop worden toegevoegd.

In het geval van het FIFO prestroom-duw-algoritme, vinden er per eenheid stroom, afkomstig uit de bron, $(1, \ldots, n-2)$ duwoperaties plaats. Er zijn $n-1$ eenheden stroom en dus bijgevolg $(n-1)(1+\cdots+n-2)$ of

$O(n^2)$ duwoperaties. In het geval van het hoogste-hoogte-eerst prestroom-duw-algoritme, vindt er per eenheid stroom, afkomstig uit de bron, één duwoperatie plaats. De reden, achter het verschil met de $O(n^2)$ duwoperaties, zit, zoals reeds besproken op het einde van Sectie 6.10.2, in het feit dat eenheden stroom niet telkens per één eenheid worden doorgegeven als duwstroom, maar telkens geaccumuleerd met de reststroom van de knoop waarnaar bij een vorige duwoperatie geduwd werd en van waaruit op dat moment geduwd wordt.

In Hoofdstuk 10 gebruiken we onder andere dit stroomnetwerk om het drastische verschil in aantal operaties tussen verschillende prestroom-duw-algoritmen visueel te illustreren.

Ondanks de uitvoeringstijden van beide prestroom-duw-algoritmen en het hierboven beschreven effect in het aantal duwoperaties, is het echter experimenteel vastgesteld dat een FIFO-implementatie veel robuuster is, met andere woorden dat deze nooit grote schommelingen in uitvoeringstijd oplevert [5].

6.10.4 Excess scaling prestroom-duw-algoritme

In deze sectie, gebaseerd op [7] en [6], bekijken we een variant van het generieke prestroom-duw-algoritme met standaard ontlaadstrategie, met een uitvoeringstijd die naast de grootte van het stroomnetwerk ook afhankelijk is van de grootte van de maximale boogcapaciteit c_{max}.

We beschouwen hiertoe eerst de *maximale reststroom* van een actieve knoop, e_{max}. Deze is een maat voor de afwijking van een prestroom ten opzichte van de bijhorende legale stroom. In het generieke prestroom-duw-algoritme met standaard ontlaadstrategie gaat de waarde voor e_{max} uiteindelijk reduceren tot nul. Het excess scaling prestroom-duw-algoritme doet dit op een systematische manier.

Het reduceren tot nul, dus het eindigen van het algoritme, hangt hier af van een parameter Δ die een bovengrens is voor e_{max} en initieel gelijk is aan $2^{\lceil \log_2 c_{max} \rceil}$. Het algoritme verloopt vanaf dan in zogenaamde Δ-*fasen*, waarbij Δ na elke fase gehalveerd wordt. Wanneer $\Delta < 1$, zijn er geen actieve knopen meer, eindigt het algoritme en dit na ten hoogste $\lceil \log_2 c_{max} \rceil + 1$ fasen.

Aangezien onverzadigde duwoperaties, die te weinig stroom verplaatsen, de uitvoeringstijd domineren, probeert het algoritme steeds een *relatief grote hoeveelheid stroom*, minstens $\frac{\Delta}{2}$, te duwen. Dit heeft immers een lager aantal onverzadigde duwoperaties tot gevolg. De relatief grote hoeveelheid slaat op het feit dat we ook *niet te veel stroom*, hoogstens Δ, mogen sturen, aangezien overtollige stroom teruggestuurd wordt en we vervolgens zo nog meer, en daarenboven nutteloos, werk verrichten. Beide doelstellingen kunnen we nu formaliseren:

1. Elke onverzadigde duwoperatie zendt ten minste $\frac{\Delta}{2}$ eenheden stroom.

2. Geen enkele reststroom zal ooit Δ overstijgen: $e_{max} \leq \Delta$.

Om deze doelstellingen te realiseren, maken we gebruik van de volgende selectiestrategie. *Onder alle knopen met een grote reststroom ($\geq \frac{\Delta}{2}$), selecteer een knoop met de kleinste hoogte.* Indien voor iedere knoop v geldt dat $e(v) < \frac{\Delta}{2}$ en dus $e_{max} < \frac{\Delta}{2}$, wordt Δ gehalveerd en begint er een nieuwe Δ-fase, zodanig dat we een nieuwe reeks actieve knopen krijgen om te selecteren. Opdat de reststroom van geen enkele actieve knoop ooit Δ zou overstijgen, moeten we de te duwen stroom op een aangepast manier bepalen. De stroom die we duwen van een knoop v naar een knoop $w \notin \{s, t\}$ bepalen we als volgt: $\delta_{v \to w} = \min\{e(v), c_f(v, w), \Delta - e(w)\}$. In het geval $w = s$ of $w = t$ gebruiken we de originele formule.

De knoopselectiestrategie en de aangepaste duwoperatie laten ons nu toe de effectieve realisatie van de doelstellingen door het excess scaling prestroom-duw-algoritme, Algoritme 8, te bewijzen.

Algoritme 8 Pseudocode van het excess scaling prestroom-duw-algoritme.

```
1:  function ExcessScaling(G)
2:      initialiseer_prestroom(G)
3:      Δ ← 2^⌈log_2 c_max⌉

4:      while Δ ≥ 1 do
5:          while ∃v ∈ V : e(v) ≥ Δ/2 do
6:              selecteer v met h(v) minimaal
7:              Aangepast_ontladen(v)
8:          end while
9:          Δ ← Δ/2
10:     end while

11:     return f
12: end function
```

Algoritme 9 Pseudocode van de standaard strategie om een actieve knoop te ontladen binnen het excess scaling prestroom-duw-algoritme.

```
1: procedure AANGEPAST_ONTLADEN(v)
2:     w ← de huidige knoop van L(v)
3:     tijdOmTeVerhogen ← false

4:     while e(v) ≠ 0 and !tijdOmTeVerhogen do
5:         if (v, w) toegelaten boog en AANGEPASTE_DUW(v, w) ≠ 0 then
6:         else
7:             if w is niet de laatste knoop van L(v) then
8:                 de volgende knoop van L(v) wordt diens huidige knoop
9:                 w ← de huidige knoop van L(v)
10:            else
11:                de eerste knoop van L(v) wordt de huidige knoop w
12:                tijdOmTeVerhogen ← true
13:            end if
14:        end if
15:    end while

16:    if tijdOmTeVerhogen then
17:        VERHOOG(v)
18:    end if
19: end procedure
```

Algoritme 10 Pseudocode van de aangepaste duwoperatie over een boog (u, v).

```
1: function AANGEPASTE_DUW(u,v)
2:     if v ≠ s en v ≠ t then
3:         δ_{u→v} ← min{e(u), c_f(u, v), Δ − e(v)}
4:     else
5:         δ_{u→v} ← min(e(u), c_f(u, v))
6:     end if

7:     f_pre(u, v) ← f_pre(u, v) + δ_{u→v}
8:     f_pre(v, u) ← −f_pre(u, v)
9:     e(u) ← e(u) − δ_{u→v}
10:    e(v) ← e(v) + δ_{u→v}

11:    return δ_{u→v}
12: end function
```

Beschouw de situatie vóór een onverzadigde duwoperatie over een boog

(u, v). We weten dan dat de boog (u, v) een toegelaten boog is zodanig dat $h(u) = h(v) + 1$. Verder is het zo dat knoop u geselecteerd werd en dus $e(u) \geq \frac{\Delta}{2}$ en $h(u)$ minimaal is over alle knopen met reststroom groter of gelijk aan $\frac{\Delta}{2}$. Hieruit volgt onmiddellijk dat $e(v) < \frac{\Delta}{2}$.

De duwoperatie die we beschouwen is onverzadigd en de hoeveelheid stroom die geduwd wordt over boog (u, v) is dus sowieso kleiner dan $c_f(u, v)$. De duwoperatie zendt met andere woorden $\min\{e(u), \Delta - e(v)\} \geq \frac{\Delta}{2}$, waarbij $\Delta - e(v) > \frac{\Delta}{2}$.

Enkel de reststroom van knoop v zal verhogen door de onverzadigde duwoperatie. Zij $e'(v)$ de reststroom van v na de onverzadigde duwoperatie. De waarde voor $e'(v)$ is gelijk aan $e(v) + \min\{e(u), \Delta - e(v)\}$. Er kunnen zich nu twee gevallen voordoen:

1. $\min\{e(u), \Delta - e(v)\} = e(u) \Rightarrow e(u) \leq \Delta - e(v) \Rightarrow e'(v) = e(v) + e(u) \leq e(v) + \Delta - e(v) = \Delta$,
2. $\min\{e(u), \Delta - e(v)\} = \Delta - e(v) \Rightarrow e'(v) = e(v) + \Delta - e(v) = \Delta$.

Analyse

We beschouwen de potentiaalfunctie $\Phi = \sum \frac{e(v)h(v)}{\Delta}$. Tijdens een Δ-fase kan Φ toenemen of afnemen:

Een verhoogoperatie die $h(v)$ verhoogt met $\lambda \geq 1$, verhoogt ook Φ met hoogstens λ want $e(v) \leq \Delta$. We weten dat $h(v) = O(n)$ en dat knopen tijdens de uitvoering van het prestroom-duw-algoritme nooit dalen. Bijgevolg is het zo dat Φ in totaal hoogstens $O(n^2)$ eenheden toeneemt door verhoogoperaties over alle Δ-fasen heen. Een verandering van Δ-fase kan Φ doen toenemen met een factor 2 of ten hoogste $O(n^2)$, aangezien er dan een halvering is van Δ en $\frac{e(v)}{\Delta} \leq 1$ en dus $\Phi \leq \sum h(v) = O(n^2)$. Bijgevolg is de totale som van toenames van Φ gelijk aan $O(n^2) + O(n^2)(\lceil \log_2 c_{max} \rceil + 1) = O(n^2 \lceil \log_2 c_{max} \rceil)$.

De twee soorten duwoperaties (verzadigd/onverzadigd) doen Φ beide dalen. Indien er een onverzadigde duwoperatie wordt uitgevoerd van knoop u naar knoop v, weten we dat $\delta_{u \to v} \geq \frac{\Delta}{2}$. Dit geeft ons voor Φ:

$$
\begin{aligned}
\Phi &= \frac{(e(u)-\delta_{u\to v})h(u)}{\Delta} + \frac{(e(v)+\delta_{u\to v})h(v)}{\Delta} + \sum_{w\in V\setminus\{u,v\}} \frac{e(w)h(w)}{\Delta} \\
&= \frac{e(u)h(u)}{\Delta} - \frac{\delta_{u\to v}(h(v)+1)}{\Delta} + \frac{e(v)h(v)}{\Delta} + \frac{\delta_{u\to v}h(v)}{\Delta} \\
&+ \sum_{w\in V\setminus\{u,v\}} \frac{e(w)h(w)}{\Delta} \\
&= \frac{e(u)h(u)}{\Delta} - \frac{\delta_{u\to v}h(v)}{\Delta} - \frac{\delta_{u\to v}}{\Delta} + \frac{e(v)h(v)}{\Delta} + \frac{\delta_{u\to v}h(v)}{\Delta} \\
&+ \sum_{w\in V\setminus\{u,v\}} \frac{e(w)h(w)}{\Delta} \\
&= -\frac{\delta_{u\to v}}{\Delta} + \sum_{w\in V} \frac{e(w)h(w)}{\Delta}
\end{aligned}
$$

De waarde van Φ neemt af met $\frac{1}{2} \leq \frac{\delta_{u\to v}}{\Delta} \leq 1$. Ze daalt met andere woorden met minstens $\frac{1}{2}$ eenheid. De initiële en uiteindelijke waarde van Φ zijn gelijk aan nul. Het totaal aantal onverzadigde duwoperaties is dus ten hoogste twee keer de som van alle toenames van Φ gedurende de uitvoering van het algoritme, met andere woorden $O(n^2 \lceil \log_2 c_{max} \rceil)$.

Tot op heden hebben we de manier om een knoop met de kleinste hoogte onder alle knopen met een grote reststroom te identificeren, niet vermeld. We maken hiervoor gebruik van een gelijkaardige array van lijsten B_i als bij het hoogste-hoogte-eerst prestroom-duw-algoritme. Een lijst B_i bevat in dit geval alle actieve knopen v met $(h(v) = i) \wedge (e(v) \geq \frac{\Delta}{2})$. Index b geeft de laagste hoogte van een actieve knoop met grote reststroom aan en komt met andere woorden overeen met de i van de laagste niet-lege B_i.

Het onderhouden van de array van lijsten is opnieuw geen dominante operatie. Om alle onverzadigde duwoperaties uit te voeren, hebben we $O(n^2 \lceil \log_2 c_{max} \rceil)$ tijd nodig. Om alle verhoogoperaties en verzadigde duwoperaties uit te voeren hebben we twee maal $O(nm)$ tijd nodig. m kan in het slechtste geval $O(n^2)$ zijn, waardoor het noodzakelijk is om $O(nm)$ mee op te nemen in de tijdscomplexiteit van het excess scaling prestroom-duw-algoritme.

Stelling 6.20. *Het excess scaling prestroom-duw-algoritme heeft een uitvoeringstijd van $O(nm + n^2 \log_2 c_{max})$ en lost het maximumstroomprobleem op.*

Zo lang c_{max} polynomiaal begrensd is in n, met andere woorden $c_{max} = O(n^k)$, voor een constante k, is de theoretische tijdscomplexiteit van het excess scaling prestroom-duw-algoritme beter dan die van het FIFO- en hoogste-hoogte-eerst algoritme.

6.11 Minimale pathetische constructies

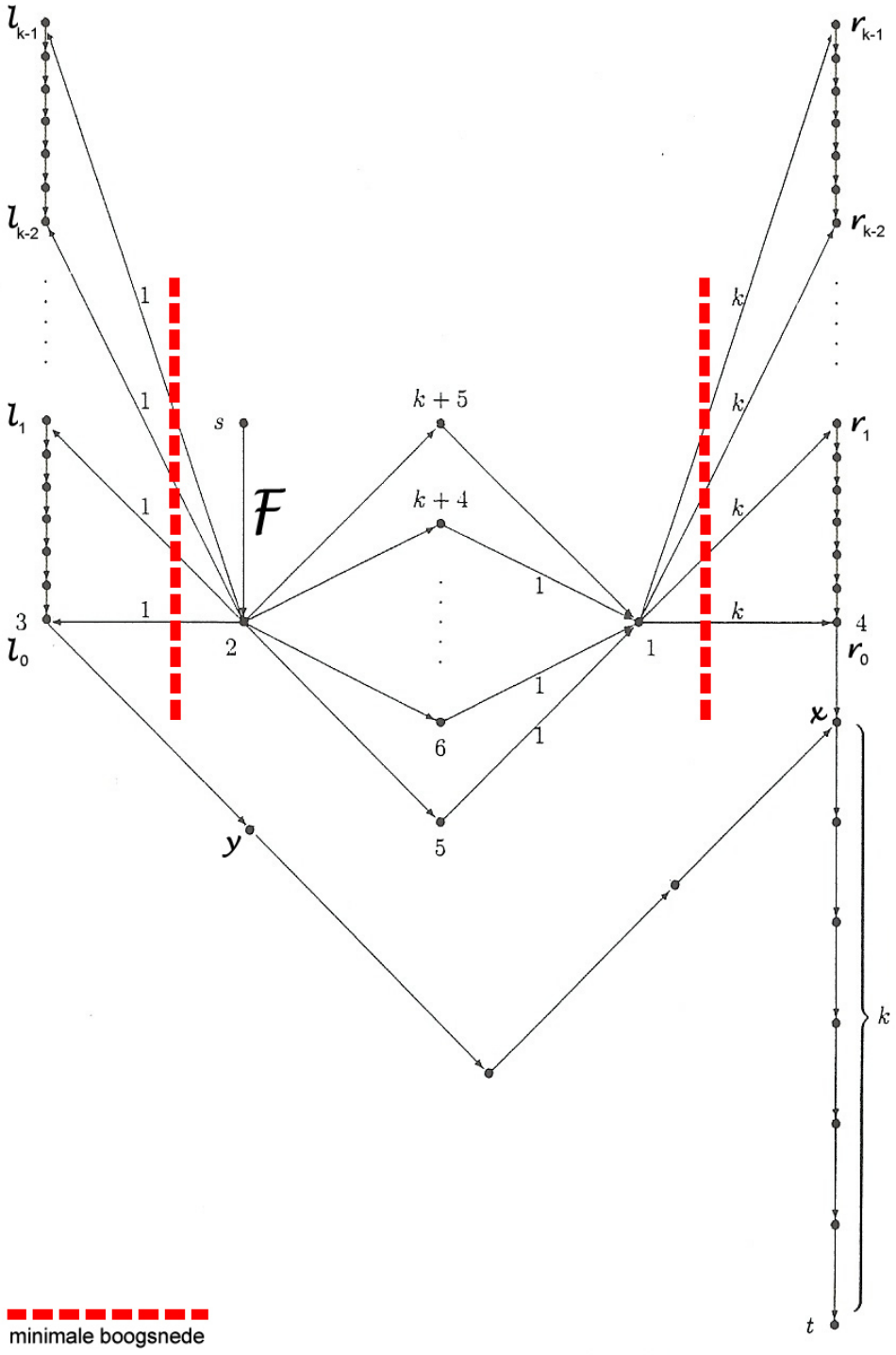
Het FIFO- en hoogste-hoogte-eerst prestroom-duw-algoritme hebben, in tegenstelling tot het excess scaling prestroom-duw-algoritme, strikte tijdsgrenzen. Een strikte tijdsgrens wil zeggen dat er klassen van stroomnetwerken bestaan die ervoor zorgen dat deze effectief zoveel berekeningen doen als aangegeven door hun worst-case complexiteit. De kleinste stroomnetwerken uit zo'n klasse van stroomnetwerken noemen we minimale pathetische constructies. Dergelijke stroomnetwerken tonen aan dat er geen verbetering mogelijk is door een slimmere analyse en zorgen ervoor dat worst-case uitvoeringen van het FIFO- en hoogste-hoogte-eerst prestroom-duw-algoritme gevisualiseerd kunnen worden.

We bekijken ter illustratie een voorbeeld van een dergelijke constructie die gebaseerd is op [5]. De constructie die we hier beschouwen in Figuur 6.7[2], is een stroomnetwerk dat substantieel eenvoudiger is dan de originele constructie van Cheriyan en Maheshwari. We zullen zodadelijk zien dat een uitvoering van het FIFO prestroom-duw-algoritme op dit stroomnetwerk een worst-case gedrag, meer bepaald een uitvoeringstijd van $\Theta(n^3)$, vertoont. Dit laatste bevestigt het feit dat het FIFO prestroom-duw-algoritme een strikte tijdsgrens heeft.

Het stroomnetwerk wordt geparametriseerd door k. We gaan ervan uit dat de knopen van het stroomnetwerk geordend behandeld worden en dit volgens stijgend knoopnummer. Het worst-case gedrag, dat we willen illustreren, is immers afhankelijk van deze orde. Figuur 6.7 verduidelijkt hoe de knopen genummerd zijn en toont tevens de bron s, de afvoer t, knoop x, knoop y, knopen $l_0, \ldots, l_{k-1}$ en knopen $r_0, \ldots, r_{k-1}$ en de capaciteiten van de bogen. De capaciteit van boog $(s, 2)$ bedraagt $F >> k^3$. Deze laatste keuze is van belang voor het uiteindelijk bekomen van een maximale stroom aangezien de minimale boogsnede, aangeduid in stippellijn, een capaciteit heeft van $k^2 + k$ en we werken met $k \geq 2$ om het gewenste effect te bekomen. Alle bogen waarvoor er geen capaciteit getoond wordt, hebben, net als boog $(s, 2)$, een capaciteit die gelijk is aan F.

Een uitvoering van het FIFO prestroom-duw-algoritme, op dit stroomnetwerk, start met een globale verhoging en de initialisatie van de prestroom. Een globale verhoging stelt de hoogtes van de intermediaire knopen gelijk aan hun kortste afstand tot de afvoer in het residueel stroomnetwerk. Op die manier zal de reststroom, aanwezig in deze knopen, via de kortste weg de afvoer bereiken. In Sectie 7.4 komen we hier uitgebreid op terug. Ten gevolge van de globale verhoging wordt $h(s) = n$, $h(t) = 0$, $h(x) = k$, $h(y) = k + 3$, $h(l_0) = k + 4$ en $h(r_0) = k + 1$, en ten gevolge van de initialisatie van de prestroom worden F eenheden stroom van s naar knoop

[2]Merk op dat deze figuur werd aangepast ten opzichte van de originele figuur. Mijn promotor en ikzelf zijn immers van mening dat de originele figuur onmogelijk het gewenste effect, volgens de originele beschrijving van de constructie, kan bereiken.



Figuur 6.7: Constructie voor worst-case gedrag van het FIFO prestroom-duw-algoritme.

2 geduwd. We verwijzen naar deze laatste hoeveelheid stroom als *de grote hoeveelheid stroom*, zelfs ondanks het feit dat deze zijn waarde kan variëren doorheen de uitvoering van het algoritme. Deze grote hoeveelheid stroom volgt vervolgens steeds het kortste simpel pad, in het residueel stroomnetwerk, naar de afvoer en beweegt hierdoor continu tussen knoop 2 en knoop 1 via de knopen $5, 6, \ldots, k+4, k+5$.

Wanneer we de uitvoering van het algoritme nu meer gedetailleerd gaan bekijken, zien we dat, bij een eerste ontlading van knoop 2, F **volledig** naar knoop 5 verhuist. Vervolgens vinden de volgende acties plaats:

1. Knoop 5 wordt ontladen, hierbij wordt één eenheid stroom naar knoop 1 geduwd.

2. Knoop 1 wordt ontladen, hierbij wordt één eenheid stroom naar knoop 4 geduwd.

3. Knoop 5 wordt ontladen, hierbij wordt de grote hoeveelheid stroom terug naar knoop 2 geduwd.

4. Knoop 4 wordt ontladen, hierbij wordt één eenheid stroom in de richting van de afvoer gestuurd via $4 \rightarrow t$ in het residueel stroomnetwerk.

Deze ontladingen herhalen zich voor de knopen $6, \ldots, k+4$ waarbij er telkens één eenheid stroom op $4 \rightarrow t$ in het residueel stroomnetwerk geplaatst wordt. Elke knoop op $x \rightarrow t$ zal uiteindelijk exact één eenheid reststroom hebben. Merk op dat boog $(1, 4)$ nu verzadigd is.

Vervolgens komt de grote hoeveelheid stroom, bij een volgende ontlading van knoop 2, via knoop $k+5$ in knoop 1 terecht. De residuele capaciteit van boog $(1, 4)$ is echter volledig benut. De kortste weg, in het residueel stroomnetwerk, om de grote hoeveelheid stroom bij de afvoer te krijgen, gaat nu plots via knoop 3. Knoop 1 wordt vervolgens ontladen en hierdoor belandt de grote hoeveelheid stroom weer terug in knoop $k+5$. De grote hoeveelheid stroom verhuist vervolgens terug naar knoop 2, nadat deze eerst nog eens van knoop $k+5$ naar knoop 1 en via knopen $5, \ldots, k+5$, terug gestuurd werd. Knoop 1 gaat knoop 2 immers vooraf in de orde wanneer knoop $k+5$ op het punt staat een eerste ontlading uit te voeren.

Vanuit knoop 2 wordt er vervolgens één eenheid stroom naar knoop 3 geduwd. De boog $(2, 3)$ is nu verzadigd en de kortste weg om de grote hoeveelheid stroom bij de afvoer te krijgen, gaat nu weer via knoop 1.

Het hele, hierboven beschreven, proces herhaalt zich nu opnieuw, maar dan met knoop r_1 in plaats van knoop 4 (r_0) en knoop l_1 in plaats van knoop 3 (l_0). Voor de aanvang van een nieuwe herhaling moet er echter steeds een nieuwe globale verhoging gebeurd zijn.

Merk op dat de beweging van de grote hoeveelheid stroom tussen knoop 2 en knoop 1 wordt veroorzaakt doordat steeds $h(r_i) < h(l_i) < h(r_{i+1}) <$

$h(l_{i+1})$, met $i \in \{0, \ldots, k-2\}$ en waarbij $h(r_i) = k + 1 + 6i$ en $h(l_i) = k + 4 + 6i$, en waardoor de grote hoeveelheid stroom steeds afwisselend via r_i, l_i, r_{i+1}, l_{i+1}, ... richting t gestuurd wordt. De hoogtes van knopen r_i en l_i veranderen immers nooit omdat de bogen, die overeenstemmen met de kortste simpele paden $r_i \to t$ en $l_i \to t$ in het residueel stroomnetwerk, nooit verzadigd worden. De structuur is zo gekozen zodat op elk moment, elke knoop op het simpel pad $x \to t$ en $y \to t$ ofwel géén, ofwel exact één eenheid reststroom heeft.

Men kan dus onmiddellijk inzien dat de maximale stroom van $k^2 + k$ eenheid per eenheid over simpele paden, van lengte $\geq k$, in het residueel stroomnetwerk gestuurd wordt. Dit vereist uiteraard $\Theta(k^3)$ onverzadigde duwoperaties.

Merk op dat het hoogste-hoogte-eerst prestroom-duw-algoritme slechts $\Theta(k^2)$ onverzadigde duwoperaties zal gebruiken op dit stroomnetwerk aangezien de hoogte van knoop 1 en knoop 2 telkens hoger ligt dan een actieve knoop r_i of l_i. Hierdoor zullen de knopen r_i en l_i eerst alle stroom verzamelen tot de bogen op de minimale boogsnede verzadigd zijn en vervolgens zal deze stroom met $\Theta(k)$ onverzadigde duwoperaties naar t gestuurd worden. Voor dit algoritme zullen de onverzadigde duwoperaties, die plaatsvinden tijdens de fasen dat er stroom wordt uitgewisseld tussen knoop 2 en knoop 1, echter dominant zijn. Er zijn k dergelijke fasen en elke fase vereist $\Theta(k)$ onverzadigde duwoperaties.

We behandelen verder geen dergelijke constructies meer vanwege hun complexiteit, maar verwijzen hiervoor naar [10].

6.12 Toegelaten-boog-selectie

Tot op heden hebben we de volgorde waarin buren van een knoop geselecteerd worden bij het uitvoeren van duwoperaties buiten beschouwing gelaten. Men kan echter de toegelaten bogen van een knoop volgens een welbepaalde strategie selecteren. We onderscheiden twee strategieën voor het selecteren van een toegelaten boog:

1. toegelaten boog met de grootste residuele capaciteit,
2. toegelaten boog met de kleinste residuele capaciteit.

Vanuit praktisch standpunt lijkt de keuze van een toegelaten boog met de grootste residuele capaciteit de beste te zijn, aangezien de buur hoogstwaarschijnlijk ook weer over genoeg uitgaande boogcapaciteit zal beschikken om van de stroom af te geraken. Er zijn echter geen theoretische resultaten bekend die formeel aantonen dat toegelaten bogen strategisch kiezen tot minder duwoperaties leidt.

Hoofdstuk 7

Heuristieken voor prestroom-duw-algoritmen

Prestroom-duw-algoritmen hebben ook een aantal *nadelen*. In dit hoofdstuk bespreken we deze nadelen en *heuristieken* die trachten deze nadelen te compenseren. Deze heuristieken hebben geen invloed op de theoretische worst-case tijdscomplexiteit van een prestroom-duw-algoritme, maar leiden vaak tot een significante verbetering van de praktische uitvoeringstijd.

7.1 2-fasig/Sequentieel prestroom-duw-algoritme

De aandachtige lezer heeft ondertussen al ondervonden dat er bij het uitvoeren van een prestroom-duw-algoritme mogelijk een aantal overtollige operaties worden uitgevoerd wanneer we enkel geïnteresseerd zijn in de waarde van een maximale stroom. We hebben het hier over de duw- en verhoogoperaties die samen reststroom terug naar de bron sturen indien deze de afvoer niet kan bereiken. Een oplossing hiervoor bestaat erin het algoritme te stoppen op het moment dat de prestroom reeds een maximale stroom is.

Om de uitvoering van een prestroom-duw-algoritme dan te kunnen stoppen, splitsen we deze op in de twee fasen die we tot op heden in parallel beschouwd hebben: de fase tot en met het vinden van een minimale boogsnede en de eventuele fase erna, waarbij de reststroom die niet tot in de afvoer is geraakt, weer naar de bron afvloeit. We creëren dan een soort van *sequentiële* versie van een prestroom-duw-algoritme en kunnen dan na de eerste fase gewoon stoppen indien we enkel de waarde van de maximale stroom willen bepalen. Des te dichter de minimale boogsnede bij de afvoer gelegen is, des te meer voordeel we hieruit halen: al de reststroom moet dan immers over een grotere afstand naar de bron gestuurd worden. Voor we een manier bekijken om een dergelijke opsplitsing te maken, bekijken we eerst de volgende stelling en zijn bewijs:

Stelling 7.1. *De reststroom van een actieve knoop v met $h(v) > n-2$ kan nooit de afvoer t bereiken.*

Bewijs. Stel dat $h(v) = n-1$, dat v actief is en dat v een stroom kan sturen naar de afvoer t. Opdat de reststroom in knoop v de afvoer zou kunnen bereiken moet er een simpel pad $p = (v = v_0, v_1, \ldots, v_k = t)$ bestaan waarover de stroom kan geduwd worden. Aangezien de hoogte van een knoop enkel kan stijgen, die van de afvoer steeds nul is en een duwoperatie van v_i naar v_{i+1} vereist dat $h(v_i) = h(v_{i+1}) + 1$, moet $k \geq n-1$. Dit is echter onmogelijk omdat $h(s)$ steeds gelijk is aan n en dus niet op p kan liggen. □

Fase 1: stuur zoveel mogelijk stroom naar de afvoer

1. Initialiseer de prestroom.
2. Definieer een deelverzameling $W \subset V$ die intermediaire knopen bevat met een hoogte groter dan $n-2$.
3. Ontlaad actieve knopen met een hoogte van 1 tot $n-2$. Knopen met een hoogte groter dan $n-2$ worden vervolgens toegevoegd aan de verzameling W. De knopen in W worden niet verder ontladen. Wanneer alle knopen in $V \setminus W$ inactief geworden zijn, stopt de eerste fase en hebben we een minimale boogsnede gevonden.

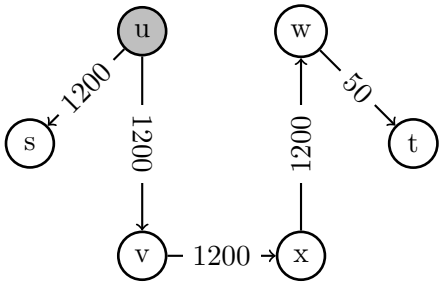
We zouden na de uitvoering van fase 1 reeds kunnen stoppen en weten dan de waarde van een maximale stroom, maar in geval van teveel stroom niet zijn volledige verdeling. Indien we deze verdeling dan ook nog wensen te weten, kunnen we fase 2 uitvoeren.

Fase 2: stuur het eventuele teveel aan stroom terug naar de bron

1. Verhoog alle knopen uit W, waarvan de hoogte $< n+1$, tot een hoogte van $n+1$.
2. Ontlaad de knopen uit W tot ze allemaal inactief geworden zijn.

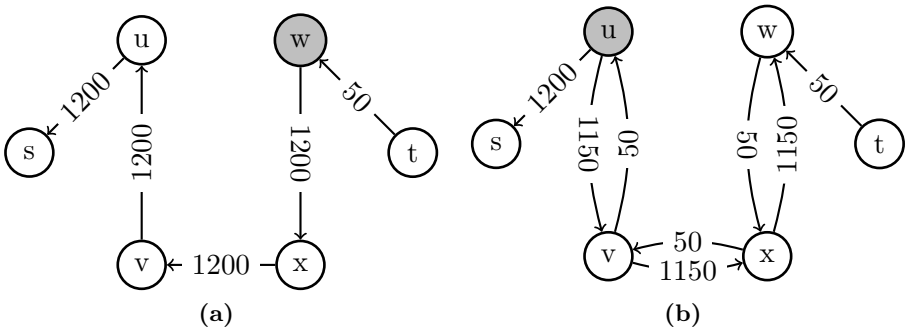
7.2 Het ping-pong effect

Laten we een stroomnetwerk beschouwen waarin er reeds een initiële prestroom gecreëerd werd. Het bijhorend residueel stroomnetwerk wordt afgebeeld in Figuur 7.1. We bekijken nu een verdere uitvoering van een prestroom-duw-algoritme op dit stroomnetwerk, ongeacht de manier waarop actieve knopen geselecteerd worden, aangezien er steeds maar één knoop actief zal zijn. De hoogtes en reststromen van de knopen, tijdens de verschillende stadia van de uitvoering, vinden we telkens terug in Tabel 7.1.



Figuur 7.1: Voorbeeld van het opduiken van het ping-pong effect.

Knoop u, v en x worden vervolgens en in deze volgorde volledig ontladen. Wanneer we dan knoop w eerst enkel verhogen en vanuit deze knoop 50 eenheden stroom naar de afvoer sturen, bekomen we de situatie zoals weergegeven in Figuur 7.2(a). Op dit moment bestaat er vanuit geen enkele knoop meer een simpel pad naar de afvoer, hetgeen er op wijst dat al de reststroom die zich nog in knoop w bevindt, terug naar de bron moet.



Figuur 7.2: Voorbeeld van het opduiken van het ping-pong effect.

Na het verder ontladen van knoop w worden x en v volledig ontladen en wanneer we knoop u eerst enkel verhogen, bekomen we een situatie zoals in Figuur 7.2(b). Dit principe wordt hierna opnieuw toegepast, maar in de andere richting, naar knoop x toe. Het resultaat hiervan is zichtbaar in Figuur 7.3(a). Men ziet duidelijk dat er tussen knopen u en w steeds dezelfde reststroom "heen" en "weer" gestuurd wordt en dit van u naar w en van w naar u. We kunnen dit vergelijken met het spelen van ping-pong of tafeltennis, waarbij het balletje van de ene speler naar de andere wordt geslagen en weer terug. Vandaar dat we hier spreken van een *ping-pong effect* waarbij we naar de "heen" verwijzen met de term *ping* en naar de "weer" met de term *pong*.

Dit gaat zo door tot de hoogte van knoop u gelijk is aan zeven, zie Fi-

Figuur 7.1	s	u	v	x	w	t
h(.)	6	0	0	0	0	0
e(.)	-1200	1200	0	0	0	0
Figuur 7.2(a)	s	u	v	x	w	t
h(.)	6	1	1	1	1	0
e(.)	-1200	0	0	0	1150	50
Figuur 7.2(b)	s	u	v	x	w	t
h(.)	6	3	2	2	2	0
e(.)	-1200	1150	0	0	0	50
Figuur 7.3(a)	s	u	v	x	w	t
h(.)	6	3	3	3	4	0
e(.)	-1200	0	0	0	1150	50
Figuur 7.3(b)	s	u	v	x	w	t
h(.)	6	7	6	6	6	0
e(.)	-1200	1150	0	0	0	50

Tabel 7.1: De hoogtes en reststromen van de knopen uit het voorbeeld van het opduiken van het ping-pong effect.

guur 7.3(b). Het prestroom-duw-algoritme kan nu bij een verdere ontlading van deze knoop zijn reststroom volledig afstaan aan de bron, waarop het prestroom-duw-algoritme eindigt.

Indien er in het originele stroomnetwerk ook een boog (w, u) zou bestaan met $c(w, u) = 1200$, breidt het ping-pong effect zich uit tot een zogenaamde *excess cycle*.

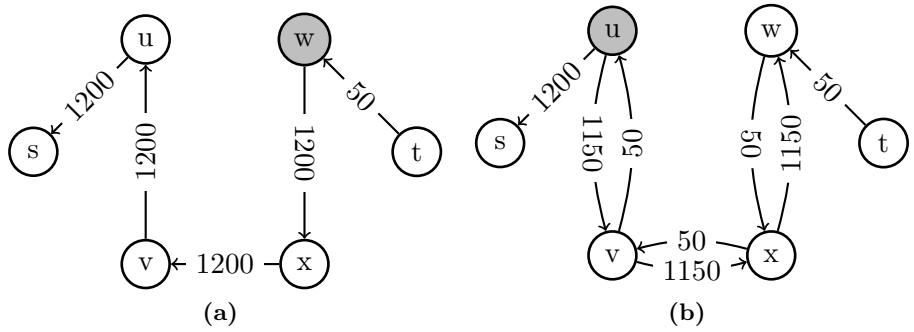
We besluiten deze sectie met de observatie dat het ping-pong effect tal van overbodige verhoog- en duwoperaties van actieve knopen introduceert tijdens de uitvoering van een prestroom-duw-algoritme. We gaan nu op zoek naar methoden om dit effect te verzwakken of hopelijk helemaal weg te werken.

7.3 Kloofheuristiek

De *kloofheuristiek* bestaat uit het controleren op het bestaan van een *afstandskloof* op het einde van elke verhoogoperatie. Er bestaat een afstandskloof in het stroomnetwerk op het moment dat:

$$\exists l \in \{0, \ldots n\}, \forall v \in V : h(v) \neq l$$

Knopen $v \in V$ waarvoor geldt dat $l < h(v)$ kunnen de afvoer vanwege de afstandskloof niet meer bereiken. De kloofheuristiek werkt deze afstands-



Figuur 7.3: Voorbeeld van het opduiken van het ping-pong effect.

kloof weg door dergelijke knopen onmiddellijk naar een hoogte van $n+1$ te brengen.

Merk op dat deze situatie niet kan voorkomen voor knopen met een hoogte vanaf n. Dergelijke knopen onderzoeken, is dus nutteloos.

We kunnen nu een 2-fasig prestroom-duw-algoritme eventueel uitbreiden met het gebruik van deze heuristiek. Knopen die een hoogte hebben $\leq n-2$ en waarvoor er een afstandskloof bestaat, kunnen we ook aan W toevoegen aangezien er voor dergelijke knopen ook geen simpel pad naar de afvoer bestaat in het residueel stroomnetwerk.

We passen deze heuristiek ter illustratie toe op het voorbeeld uit Sectie 7.2. Vlak na de verhoging van knoop u tot $h(u) = 3$, zie Figuur 7.2(b), zou de kloofheuristiek een waarde $l = 1$ vinden. Voor de knopen u, v, x en w geldt er: $l < h(v) = h(x) = h(w) < h(u) < n$. Deze knopen kunnen de afvoer niet meer bereiken en worden vervolgens naar een hoogte van $n+1$ gebracht. De heuristiek schakelt het ping-pong effect dus uit.

In het algemeen zorgt de heuristiek er dus voor dat er een hele hoop ontladingen, om knopen die de afvoer niet meer kunnen bereiken tot de hoogte van de bron te brengen, vermeden worden. Dit resulteert natuurlijk in een betere performantie, in het bijzonder voor stroomnetwerken waarin er een ping-pong effect optreedt.

7.4 Globale-verhoog-heuristiek

Definitie 7.2. *De afstand van een knoop v tot een knoop w is de lengte (het aantal residuele bogen) van het kortste simpel pad van v naar w in het residueel stroomnetwerk en definiëren we met $d(v, w)$.*

De initialisatie van de hoogtes van de knopen gebeurt momenteel op een naïeve manier. We weten echter dat de stroom van bron naar afvoer dient te vloeien en dat stroom enkel van hoog naar laag vloeit tussen twee

knopen met een hoogteverschil van één eenheid. Het is daarom interessant om reeds van in het begin deze "trap" te bouwen en, met andere woorden, al van bij de initialisatie van de prestroom een *exacte hoogte* toe te kennen aan de knopen van het stroomnetwerk. Dit resulteert sowieso in minder ontladingen, aangezien de "trap" naar de afvoer reeds gevormd is.

De exacte hoogte van een knoop is de hoogte die gelijk is aan de afstand van deze knoop tot de afvoer: $h(v) = d_t(v), \forall v \in V \setminus \{s, t\}$, met $d_t(v)$ als een verwijzing naar $d(v, t)$, de afstand van v tot t. Indien een knoop $v \in V \setminus \{s, t\}$ later, tijdens de uitvoering van het algoritme, de afvoer niet meer kan bereiken, definiëren we zijn exacte hoogte als volgt: $h(v) = d_s(v) + n$, met $d_s(v)$ als een verwijzing naar $d(s, v)$, de afstand van s tot v.

Alle kortste simpele paden berekenen we met behulp van een achterwaartse BFS[1] vanuit afvoer t, resp. bron s.

Tijdens de uitvoering van een prestroom-duw-algoritme gaan de hoogtes van de knopen mogelijk afwijken van hun exacte hoogtes, hetgeen natuurlijk een hoger aantal duw- en verhoogoperaties tot gevolg heeft ten opzichte van een ideale situatie waarbij exacte hoogtes ten aller tijde onderhouden worden. Dit laatste brengt uiteraard een hoge onderhoudskost met zich mee en dus is het geen slecht idee om hiervoor een kost-doeltreffende strategie te zoeken die zich ergens tussen de twee in bevindt.

Deze strategie, die we de globale-verhoog-heuristiek noemen, update periodiek de hoogtes van **alle** knopen in het stroomnetwerk tot hun exacte hoogtes. We verwijzen bijgevolg naar deze update als een globale verhoging. Met periodiek bedoelen we om de kx ontladingen, waarbij k een constante is en x een variabele die in zekere zin afhankelijk is van de structuur van het stroomnetwerk. Laten we nu eens kijken hoe we de frequentie kx zo optimaal mogelijk kunnen invullen.

Volgens [5] stellen we kx best gelijk aan:

- n indien $m \leq 10n$,
- $2n$ indien $m > 10n$.

Deze regeltjes zijn op een empirische manier tot stand gekomen, maar begeven zich volgens [5] toch dichtbij een optimale frequentie die afhangt van de implementatie van het prestroom-duw-algoritme, de structuur en de grootte van het stroomnetwerk in kwestie.

We zouden de frequentie waarmee we globale verhogingen uitvoeren ook kunnen uitdrukken in termen van m. Laten we eens bekijken of een dergelijke keuze enig verschil kan maken ten opzichte van n als frequentie. Merk op dat het aantal ontladingen gedomineerd wordt door het aantal onverzadigde duwoperaties, $O(n^2m)$.

[1] Breadth-First Search

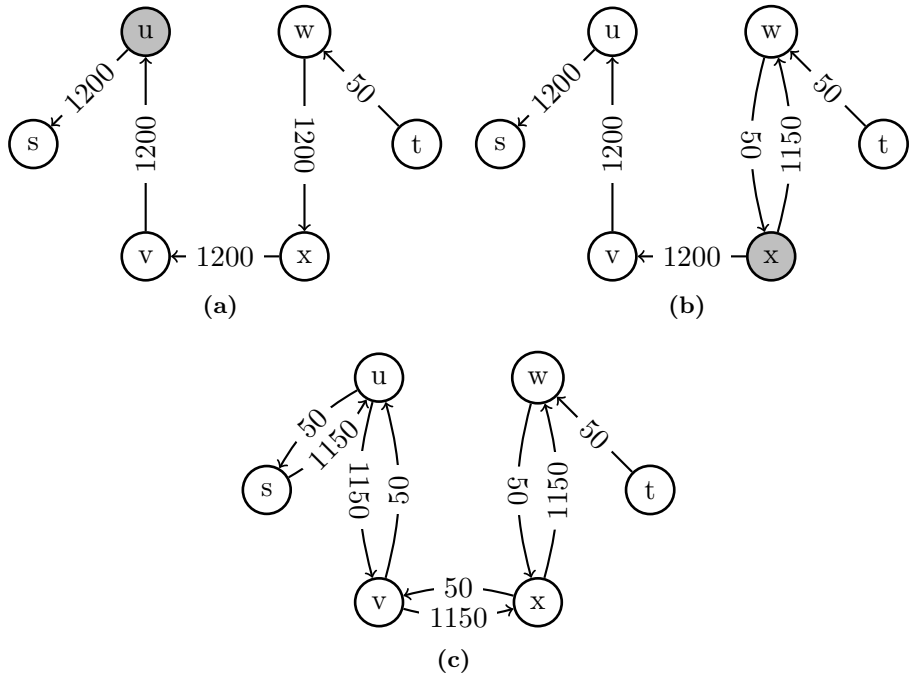
Indien we n als frequentie nemen, zijn het aantal globale verhogingen $O(\frac{n^2m}{n})$. Indien we m als frequentie nemen, zijn het aantal globale verhogingen $O(\frac{n^2m}{m})$ en zien we dat deze frequentie, in tegenstelling tot n, niet gevoelig is aan de dichtheid van het stroomnetwerk. Een dergelijke keuze van frequentie houdt dus rekening met het feit dat hoe dichter het stroomnetwerk is, des te groter het spectrum is van de hoogtes onder alle knopen en er bijgevolg in het algemeen minder globale verhogingen nodig zijn. Hoe dichter het stroomnetwerk is, des te meer tijd de globale-verhoog-heuristiek vergt, die immers afhankelijk is van het aantal bogen. We zouden er dus van kunnen uitgaan dat we de frequentie waarmee we globale verhogingen uitvoeren, misschien wel het beste kunnen kiezen in termen van m. Merk tevens op dat de uitvoeringstijd van het generieke prestroom-duw-algoritme met standaard ontlaadstrategie lijdt onder de keuze van n als frequentie bij zéér dichte stroomnetwerk ($m \approx n^2$), de totale kost van de globale verhogingen is dan immers dominant.

Stel nu, ter illustratie van de globale-verhoog-heuristiek, dat we initieel exacte hoogtes toekennen aan de knopen van het stroomnetwerk uit Sectie 7.2, Figuur 7.4(a), en de globale-verhoog-heuristiek toepassen, dan zou er na de zesde ontlading een globale verhoging gebeuren. We bevinden ons dan in de situatie zoals weergegeven in Figuur 7.4(b). De hoogtes en reststromen van de knopen zijn terug te vinden in Tabel 7.2. Door het plaatsvinden van een globale verhoging, wordt er voor knopen u, v, x en w onmiddellijk een "trap" naar de bron toe gevormd waarlangs alle reststroom kan wegvloeien, zie Figuur 7.4(c). Het ping-pong effect wordt op deze manier volledig voorkomen.

De kloofheuristiek is dus een zwakke versie van de globale-verhoog-heuristiek. Voor de knopen die de afvoer niet meer kunnen bereiken, wordt er door de globale-verhoog-heuristiek immers ook de "trap" naar de bron gevormd. De globale-verhoog-heuristiek gaat daarenboven ook de knopen

Figuur 7.4(a)	s	u	v	x	w	t
h(.)	6	4	3	2	1	0
e(.)	-1200	1200	0	0	0	0
Figuur 7.4(b)	s	u	v	x	w	t
h(.)	6	4	3	2	3	0
e(.)	-1200	0	0	1150	0	50
Figuur 7.4(c)	s	u	v	x	w	t
h(.)	6	7	8	9	10	0
e(.)	-50	0	0	0	0	50

Tabel 7.2: De hoogtes en reststromen van de knopen uit het voorbeeld van een globale verhoging.



Figuur 7.4: Voorbeeld van een globale verhoging.

die wel de afvoer kunnen bereiken een handje toesteken door het periodiek onderhouden van een "trap", maar dan naar de afvoer toe.

Door het feit dat de kloofheuristiek geen hulp voorziet om stroom ook terug richting de bron te sturen, is een implementatie van een prestroom-duw-algoritme met enkel de kloofheuristiek gevoeliger aan de locatie van de minimale boogsnede dan een implementatie met enkel de globale-verhoog-heuristiek.

De implementatie van een prestroom-duw-algoritme die enkel gebruik maakt van de globale-verhoog-heuristiek is, met betrekking tot performantie, dus bijgevolg superieur aan één die enkel gebruik maakt van de kloof-heuristiek. Het is echter zo dat de kloofheuristiek niet afhankelijk is van het aantal bogen in het stroomnetwerk, maar enkel informatie in verband met de hoogtes van de knopen vereist. Dit maakt de kloofheuristiek op zich minder zwaar dan de globale-verhoog-heuristiek en voornamelijk met betrekking tot dichte stroomnetwerken.

Al deze waarnemingen suggereren om misschien wel één of andere combinatie van beide heuristieken te gebruiken voor de meest robuuste implementatie van een prestroom-duw-algoritme, zeker indien we te maken hebben met zéér dichte stroomnetwerken [5].

7.5 Schatting van de maximale stroomwaarde

De maximale stroomwaarde van een stroomnetwerk komt, zoals reeds beschreven in Sectie 5.3, overeen met de capaciteit van een minimale boogsnede van dit stroomnetwerk. De capaciteit van een dergelijke minimale boogsnede berekenen, is dus equivalent met het oplossen van het maximumstroomprobleem. We kunnen voor deze capaciteit echter wel een bovengrens bepalen [8] alvorens te starten met de uitvoering van een prestroom-duw-algoritme op het stroomnetwerk en op die manier tijdswinst te boeken op deze zijn uitvoering.

Bij de initialisatie van het prestroom-duw-algoritme selecteren we een willekeurige boogsnede $[S, T]$ in het stroomnetwerk en beschouwen zijn capaciteit. Deze capaciteit is aldus een bovengrens op de maximale stroomwaarde van een maximale stroom die we doorheen het desbetreffende stroomnetwerk kunnen sturen en definiëren we als $|f^u|$[2].

We kunnen twee manieren bedenken om een dergelijke willekeurige boogsnede $[S, T]$ te selecteren, in de hoop deze zijn capaciteit zo laag mogelijk te houden en de maximale stroomwaarde zo dicht mogelijk te benaderen:

1. Op een naïeve manier via een kop-of-munt-verdeling van de knopen van het stroomnetwerk over S en T.

2. Op een meer intelligente manier via een gerandomiseerde BFS-zoektocht vanuit de bron waarbij enkel knopen, waarnaar vanuit de momenteel bezochte knoop een boog gaat die een capaciteit heeft die groter is dan een bepaalde drempelwaarde, in aanmerking komen om bezocht te worden. Alle bezochte knopen worden in S opgenomen, alle andere in T.

We creëren vervolgens een nieuwe, artificiële bron s_a en een nieuwe boog (s_a, s) met $c(s_a, s) = |f^u|$ en die s_a met de originele bron s verbindt. Merk op dat het nieuwe stroomnetwerk equivalent is met het oude en de extra kost om dit nieuwe stroomnetwerk te bekomen, verwaarloosbaar is. Het bepalen van $|f^u|$ met behulp van één van de bovenstaande selectiemethoden kost immers $O(m)$ tijd.

Voor de creatie van S en T op basis van een kop-of-munt-verdeling bepalen we voor elke knoop $v \in V \setminus \{s, t\}$ op een willekeurige manier een waarde 0 (kop) of 1 (munt). De kost van deze operatie bedraagt $O(n)$ tijd aangezien we hiervoor door alle knopen lopen. De creatie van S en T op basis van een BFS-zoektocht heeft als kost de tijdscomplexiteit van een BFS, nl. $O(m)$ tijd.

Tot slot stellen we $|f^u|$ gelijk aan de capaciteit van de, hetzij met een kop-of-munt-verdeling, hetzij met een BFS-zoektocht, gecreëerde boogsnede. We maken hiervoor gebruik van de, in Sectie 6.9.6 beschreven, lijst $L(v)$ voor

[2] u van upper bound

iedere knoop van het stroomnetwerk. We identificeren hiermee de voorwaarste bogen van $[S, T]$ en tellen tijdens dit proces hun capaciteiten bij elkaar op. De kost van deze operatie bedraagt dus $O(m)$.

Het voordeel van deze heuristiek is dat er mogelijk minder stroom in het stroomnetwerk gestuurd wordt die we later in de tweede fase van het prestroom-duw-algoritme weer moeten laten afvloeien naar de bron. Bijgevolg zal de uitvoeringstijd van een prestroom-duw-algoritme dat gebruik maakt van deze heuristiek, op stroomnetwerken met een normaliter lange tweede fase, dalen.

We kunnen uiteindelijk nog een stapje verder gaan en, in beperkte mate, meerdere boogsneden beschouwen en $|f^u|$ gelijkstellen aan het minimum van deze hun capaciteiten. Met beperkte mate bedoelen we dat de theoretische tijdscomplexiteit van het prestroom-duw-algoritme dominant dient te blijven. Deze manier van werken zal de benadering van de uiteindelijke maximale stroomwaarde van het stroomnetwerk natuurlijk alleen maar ten goede komen, aangezien we met een grotere kans een minimale boogsnede op voorhand trachten te selecteren.

7.6 Dynamische bomen

We bekijken in deze sectie, gebaseerd op [6], [7], [18] en [8], een heuristiek die, door het invoeren van een duurdere onverzadigde duwoperatie, het aantal onverzadigde duwoperaties probeert te reduceren om op die manier tijdswinst te boeken. We voeren hiervoor een opeenvolging van duwoperaties uit in één operatie over een simpel pad, in het residueel stroomnetwerk, in de richting van de afvoer. Om dit idee, deze heuristiek te realiseren, maken we gebruik van een *dynamische bomen datastructuur*. Alvorens we deze kunnen bestuderen, met hierop volgend de werking van de heuristiek, introduceren we twee nieuwe concepten.

Definitie 7.3. *Een* boom *is een gerichte graaf waarin elke twee knopen verbonden zijn door exact één pad.*

Definitie 7.4. *Een* woud *is een verzameling van bomen waarbij de verzamelingen, gevormd door de knopen van de betrokken bomen, disjunct zijn.*

7.6.1 Dynamische bomen datastructuur

De dynamische bomen datastructuur is een datastructuur die **volledig gescheiden** is **van het stroomnetwerk** en representeert een woud van bomen, meer bepaald *dynamische bomen*. De *wortel* van een boom komt overeen met een knoop uit het stroomnetwerk. De bogen of de *takken* van de bomen komen elk steeds overeen met een toegelaten boog uit het residueel stroomnetwerk. Ze vormen samen een deelverzameling van de toegelaten

bogen. Elke boog draagt een *waarde* met zich mee, nl. de huidige residuele capaciteit volgens de toegelaten boog waarmee deze overeenkomt. Een *kind*, in een boom, beschouwen we als de staart van de boog die het kind met zijn *ouder*, de kop van de boog, verbindt. De gerichtheid van een boog in een boom gaat dus naar de wortel toe.

Het dynamisch aspect van de dynamische bomen datastructuur komt tot uiting in de ondersteuning van minimaal de volgende operaties:

plantBoom(v)

Maak een boom aan, in het woud, met v als wortel en enige knoop.

wortel(v)

Geef de wortel terug van de boom die knoop v bevat.

waarde(v)

Geef de waarde terug van de boog in de boom die v verbindt met zijn ouder.

zoekMin(v)

Geef de knoop w terug, waarbij w de staart is van de boog met minimale waarde en het dichtst gelegen bij de wortel op het simpel pad van v naar de wortel in de boom. Waarom het dichtst gelegen? - Wel, omdat we steeds reststroom over een zo groot mogelijke afstand willen sturen om bijgevolg zoveel mogelijk duwoperaties ineens uit te voeren.

bewerk(v, x)

Trek de waarde x af van de waarde van elke boog die deel uitmaakt van het simpel pad van knoop v tot de wortel in de boom. We voeren hierbij ook steeds de operatie in Algoritme 11 uit, waarbij p een simpel pad is uit het residueel stroomnetwerk dat overeenkomt met het simpel pad van knoop v tot de wortel in de boom.

verbind(v, w, x)

Verbind de wortel v van een boom met een knoop w uit een andere boom met een boog van waarde x. Merk op dat hierdoor de twee betrokken bomen uit het woud samengevoegd worden.

Algoritme 11 Aanpassen van de stroomwaarden en reststromen op het simpel pad.

```
1: procedure DUW_OVER_SIMPEL_PAD(p,x)
2:     for each (w, z) op p do
3:         f(w, z) ← f(w, z) + x
4:         f(z, w) ← −f(w, z)
5:         e(w) ← e(w) − x
6:         e(z) ← e(z) + x
7:     end for
8: end procedure
```

snoei(v)

Verwijder de boog die knoop v met zijn ouder verbindt. Het resultaat van deze operatie is de splitsing van de boom die knoop v bevat, waarbij een nieuwe boom met wortel v ontstaat.

7.6.2 Werking heuristiek

Het *generieke prestroom-duw-algoritme met standaard ontlaadstrategie in combinatie met dynamische bomen*, zie Algoritme 12, gaat eerst voor elke knoop v een boom planten. Elke knoop van het stroomnetwerk stelt bijgevolg een wortel van een boom uit het woud voor. Vervolgens wordt zoals steeds de prestroom geïnitialiseerd, zodat we van start kunnen gaan met de effectieve uitvoering van het algoritme.

Algoritme 12 Pseudocode van het generieke prestroom-duw-algoritme met standaard ontlaadstrategie in combinatie met dynamische bomen

```
1: function PRESTROOM-DUW(G)
2:     for each v ∈ V do
3:         plantBoom(v)
4:     end for
5:     INITIALISEER_PRESTROOM(G)

6:     while ∃ een actieve knoop v do
7:         ONTLADEN(v)
8:     end while

9:     return f
10: end function
```

Zolang als er actieve knopen zijn, worden deze zoals steeds ontladen, zie Algoritme 13.

Algoritme 13 Pseudocode van de standaard ontlaadstrategie van een actieve knoop met behulp van een boom-duwoperatie.

```
1:  procedure ONTLADEN(v)
2:      w ← de huidige knoop van L(v)
3:      tijdOmTeVerhogen ← false

4:      while e(v) ≠ 0 and !tijdOmTeVerhogen do
5:          if (v, w) is een toegelaten boog then
6:              BOOM-DUW(v,w)
7:          else
8:              if w is niet de laatste knoop van L(v) then
9:                  de volgende knoop van L(v) wordt diens huidige knoop
10:                 w ← de huidige knoop van L(v)
11:             else
12:                 de eerste knoop van L(v) wordt de huidige knoop w
13:                 snoei(u) voor elk kind van v
14:                 tijdOmTeVerhogen ← true
15:             end if
16:         end if
17:     end while

18:     if tijdOmTeVerhogen then
19:         VERHOOG(v)
20:     end if
21: end procedure
```

Ditmaal wordt er echter gebruik gemaakt van een *boom-duwoperatie*, zie Algoritme 14, in plaats van een duwoperatie en indien de te ontladen knoop verhoogd wordt, snoeien we zijn kinderen. Deze laatste operatie is nodig aangezien bij een verhoging de bogen, komende vanuit zijn kinderen, niet langer toegelaten bogen zijn vanuit deze kinderen naar de knoop.

Algoritme 14 Pseudocode van de boom-duwoperatie over een boog (v, w).

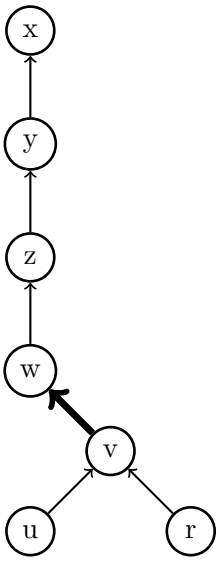
```
1:  procedure BOOM-DUW(v,w)
2:      verbind(v, w, c_f(v, w))
3:      while v ≠ wortel(v) ∧ e(v) ≠ 0 do
4:          δ ← min(e(v), waarde(zoekMin(v)))
5:          bewerk(v, δ)
6:          while waarde(zoekMin(v)) = 0 do
7:              u ← zoekMin(v)
8:              snoei(u)
9:          end while
10:     end while
11: end procedure
```

De boom-duwoperatie vormt het hart van de dynamische bomen heuristiek. De operatie voegt met tak (v, w) twee bomen van het woud samen en duwt vervolgens zoveel mogelijk reststroom vanuit v in de richting van de wortel van de nieuwe boom waar hij deel van uitmaakt. Alle takken die als gevolg van deze operatie verzadigd zijn, worden verwijderd uit de boom waaraan de oorspronkelijke boom van v werd toegevoegd. Hierdoor ontstaan mogelijk meerdere nieuwe bomen in het woud.

De correctheid van het algoritme in combinatie dynamische bomen volgt uit het feit dat, tussen twee boom-duwoperaties, actieve knopen steeds wortels zijn van dynamische bomen en elke boog, die deel uitmaakt van een tak van een dynamische boom, overeenkomt met een toegelaten boog in het residueel stroomnetwerk.

Opmerking 7.5. *Elke actieve knoop is een wortel van een dynamische boom, maar niet elke wortel van een dynamische boom is een actieve knoop. We houden hierbij de bron en de afvoer buiten beschouwing.*

We illustreren deze vaststelling met een voorbeeld. Wanneer we twee dynamische bomen verbinden door middel van een boog tussen de wortel van de ene boom, v, en een willekeurige knoop w uit de twee boom, krijgen we de situatie zoals afgebeeld in Figuur 7.5. Laten we veronderstellen dat $e(x) = 20$, $e(y) = 0$, $e(z) = 0$, $e(w) = 0$, $e(v) = 5$, $e(u) = 0$ en $e(r) = 0$. Alle bogen hebben een waarde van vijf, behalve de bogen van v naar w en van y naar x. Deze laatste hebben een waarde van 10.



Figuur 7.5: Voorbeeld van het verbinden van twee dynamische bomen.

Wanneer we nu knoop v ontladen met behulp van Algoritme 13, raakt de boog van w naar z verzadigd en wordt op deze plaats een nieuwe boom afgesplitst met als wortel w. Deze zijn reststroom is echter gelijk aan nul, hetgeen de bovenstaande opmerking duidelijk maakt.

7.6.3 Analyse

Om de worst-case uitvoeringstijd van het generieke prestroom-duw-algoritme met standaard ontlaadstrategie in combinatie met dynamische bomen te bepalen, bekijken we eerst hoeveel keer elke operatie in het worst-case geval wordt uitgevoerd.

De operatie $snoei(v)$ gebeurt voor elk kind van v als v verhoogd moet worden. Een verhoogoperatie kan $O(n)$ keer voorkomen voor v en v kan maximaal $n-1$ kinderen hebben. Dit levert ons reeds $O(n^2)$ snoeioperaties op. Verder wordt deze operatie uitgevoerd in een boom-duwoperatie ten gevolge van de verzadiging van een boog (verzadigde duwoperatie). Er zijn $O(nm)$ verzadigde duwoperaties, zie Stelling 6.13, en bijgevolg hebben we, mede vanwege het feit dat $m \geq n-1$, een totaal van $O(nm)$ snoeioperaties.

Elke operatie $verbind(v, w, x)$ voegt een boog toe aan de verzameling van bomen. Een dergelijke boog kan verdwijnen wanneer een snoeioperatie wordt uitgevoerd. Initieel bevat het bos enkel bomen zonder bogen. Er zijn $O(nm)$ snoeioperaties en dus kunnen er $O(nm+n-1)$ verbindoperaties plaatsvinden. De verzameling van bomen kan uiteindelijk $n-1$ bogen bevatten, hetgeen overeenkomt met de extra $n-1$ verbindoperaties. Merk op dat er net zoveel boom-duwoperaties zijn als verbindoperaties en het aantal

onverzadigde duwoperaties begrensd wordt door deze verbindoperaties.

Het aantal snoeioperaties legt dus een bovengrens op het aantal keren dat de binnenste while-lus van Algoritme 14 in totaal kan worden uitgevoerd. Het aantal keer dat de buitenste while-lus in totaal kan worden uitgevoerd is ook gelijk aan deze bovengrens. Per snoeioperatie mag de buitenste while-lus immers maar één keer extra uitgevoerd worden om v al zijn verdere reststroom kwijt te laten geraken zonder hiervoor een verzadiging van een boog te veroorzaken. Het aantal keer dat operaties $bewerk(v, x)$, $wortel(v)$, $waarde(v)$ en $zoekMin(v)$ afzonderlijk worden uitgevoerd is bijgevolg ook $O(nm)$ keer.

De operatie $plantBoom(v)$ wordt natuurlijk sowieso n keer uitgevoerd met een tijd $O(1)$. De operaties $waarde(v)$, $verbind(v, w, x)$ en $snoei(v)$ hebben normaliter een worst-case uitvoeringstijd van $O(1)$ en de operaties $wortel(v)$, $bewerk(v, x)$ en $zoekMin(v)$ normaliter een worst-case uitvoeringstijd van $O(n)$ (lengte van het langste simpel pad van v naar $wortel(v)$). Door het gebruik van een impliciete representatie van de structuur van het woud, kunnen we deze laatste operaties hun uitvoeringstijd reduceren naar $O(\log n)$, ten koste van de verhoging van de uitvoeringstijd van de andere operaties naar $O(\log n)$. In [18] werd een dergelijke impliciete representatie ontwikkeld voor een worst-case analyse. Deze valt echter buiten het bestek van deze masterproef.

Alle operaties, behalve $plantBoom(v)$, worden dus $O(nm)$ keer uitgevoerd met een uitvoeringstijd van $O(\log n)$. Dit levert ons, samen met de reeds aangetoonde correctheid van het generieke prestroom-duw-algoritme met standaard ontlaadstrategie in combinatie met dynamische bomen, het volgende resultaat:

Stelling 7.6. *Het generieke prestroom-duw-algoritme met standaard ontlaadstrategie in combinatie met dynamische bomen eindigt in $O(nm \log n)$ tijd en lost het maximumstroomprobleem op.*

Opmerking 7.7. *Als $m = O(n^{2-\alpha})$, met $0 < \alpha \leq 1$, dan heeft het generieke prestroom-duw-algoritme met standaard ontlaadstrategie in combinatie met dynamische bomen een betere bovengrens in tijd dan het FIFO- en hoogste-hoogte-eerst prestroom-duw-algoritme. Hoogste-hoogte-eerst eindigt dan immers in $O(n^{3-\frac{\alpha}{2}})$ tijd en het algoritme in combinatie met dynamische bomen in $O(n^{3-\alpha} \log n)$.*

Merk op dat deze heuristiek in de praktijk een beperkte bruikbaarheid heeft vanwege de substantiële overhead die gepaard gaat met het onderhouden van de bijhorende datastructuur [6].

Hoofdstuk 8

Onderzoek

We hebben vervolgens een aantal experimenten uitgevoerd om de, in deze masterproef, bestudeerde prestroom-duw-algoritmen en heuristieken, behalve dynamische bomen, te onderzoeken vanuit een praktisch standpunt. In hetgeen volgt, bespreken we wat er precies onderzocht werd, hoe we dit gedaan hebben en wat de bekomen resultaten zijn. De resultaten worden voor elk experiment geïnterpreteerd en, indien mogelijk, vergeleken met reeds eerder vermelde opmerkingen, suggesties en/of vaststellingen doorheen deze masterproef.

8.1 Inleiding

In deze sectie identificeren we eerst de gebruikte stroomnetwerken en prestroom-duw-algoritmen, in combinatie met eventuele heuristieken. Vervolgens gaan we dieper in op het doel van elk experiment en vermelden welke stroomnetwerken en algoritmen hierbij bewust gebruikt werden en waarom.

8.1.1 Netwerkklassen

De gebruikte stroomnetwerken zijn *instanties* van drie verschillende *netwerkklassen*: *Acyclic Dense* (AD), *Genrmf* (G) en *Washington-Line-Moderate* (WLM), en worden elk gedefinieerd door één of meerdere parameters.

De Acyclic Dense netwerkklasse bevat acyclische stroomnetwerken. Deze stroomnetwerken hebben een zo hoog mogelijke boogdensiteit (zonder de acyclische eigenschap te verbreken). De bogen hebben willekeurige capaciteiten. De enige parameter is n, het aantal knopen van het stroomnetwerk.

Een stroomnetwerk uit de Genrmf-netwerkklasse wordt gedefinieerd door vier parameters: a, b, c_1 en c_2. Het resulterende stroomnetwerk bestaat uit b lagen van $(a \times a)$ *rasters*. Elke knoop in het raster is, behalve over de diagonaal, verbonden met zijn buren in het raster door middel van een boog met capaciteit $c_2 a^2$. Er bevinden zich dus bogen in twee richtingen volgens

de lijnen die het raster zogezegd vormen. Twee opeenvolgende lagen zijn, naar de afvoer toe, vanuit elke knoop van de ene laag verbonden met een willekeurig gekozen knoop uit de daaropvolgende laag. Waarvoor parameter c_1 gebruikt wordt, zal later, in Sectie 8.2, duidelijk worden.

Stroomnetwerken waarbij $b \approx a^2$ noemen we *lang* en behoren tot de subklasse *Genrmf-Long*. Stroomnetwerken waarbij daarentegen $a \approx b^2$, noemen we *breed* en behoren tot de subklasse *Genrmf-Wide*.

Washington-Line-Moderate stroomnetwerken zijn tamelijk dichte stroomnetwerken en worden gespecificeerd door drie parameters: n, m en deg. Een dergelijk stroomnetwerk bestaat uit $n \times m$ knopen in een raster plus een bron en een afvoer. De bron is verbonden met de eerste m knopen van het raster, de laatste m knopen zijn verbonden met de afvoer, en dit in beide gevallen door middel van bogen met een zéér hoge capaciteit in vergelijking met de capaciteiten van de andere bogen. Voor elke knoop in het raster bestaan er deg bogen met willekeurige capaciteiten. Deze bogen verzorgen de verbinding van een knoop met deg willekeurig geselecteerde knopen uit de volgende deg kolommen.

Instanties van WLM hebben de opmerkelijke eigenschap dat de locatie van de minimale boogsnede ofwel dichtbij de bron, ofwel dichtbij de afvoer gelegen is. We hebben deze eigenschap gebruikt bij het opzetten van bepaalde experimenten.

We geven tot slot een overzicht van dichtheden:

- AD: $m = \frac{n(n-1)}{2}$,
- GL: $m \approx 5n$,
- GW: $m \approx 5n$,
- WLM: $m = O(n^{\frac{3}{2}})$.

8.1.2 Algoritmen

We identificeren nu, door middel van een overzicht, de verschillende gebruikte prestroom-duw-algoritmen, eventueel in combinatie met één of meerdere heuristieken.

- FIFO prestroom-duw-algoritme (FIFO) eventueel in combinatie met:
  - een schatting van de maximale stroom (FIFO$_s$),
  - de kloofheuristiek (FIFO$_k$),
  - de globale-verhoog-heuristiek (FIFO$_g$),
  - de globale-verhoog-heuristiek en de kloofheuristiek (FIFO$_{gk}$);
- Sequentiële versie van het FIFO prestroom-duw-algoritme (FIFO$_{\frac{1}{2}}$);

- Hoogste-hoogte-eerst prestroom-duw-algoritme (HHE) eventueel in combinatie met:
  - de kloofheuristiek (HHE_k),
  - de globale-verhoog-heuristiek (HHE_g),
  - de globale-verhoog-heuristiek en de kloofheuristiek (HHE_{gk});
- Excess scaling prestroom-duw-algoritme (ES).

Voor de implementatie van deze prestroom-duw-algoritmen en heuristieken verwijzen we naar Hoofdstuk 10. We merken echter wel al op dat ontladingen van actieve knopen gebeuren volgens de standaard ontlaadstrategie en dat bij de uitvoering van een algoritme knopen enkel met exacte hoogtes starten indien er gebruik gemaakt wordt van de globale-verhoog-heuristiek. Het gebruik van een dynamische bomen datastructuur werd niet geïmplementeerd.

8.1.3 Experimenten

De lezer kan zich nu de vraag stellen waarom we nu net stroomnetwerken uit deze netwerkklassen en net deze prestroom-duw-algoritmen en heuristieken gebruikt hebben bij het uitvoeren van experimenten. Deze vraag zal beantwoord worden na het definiëren van elk experiment en zijn doel.

Experiment 1 (EXP_1)

In experiment EXP_1 bestuderen we het gedrag van prestroom-duw-algoritmen FIFO, HHE en ES op instanties van netwerkklassen: GL, GW en WLM. Instanties van deze netwerkklassen zijn immers structureel bepaald door de belangrijkste aspecten van stroomnetwerken, nl. lengte, breedte en locatie van de minimale boogsnede, die de uitvoeringstijd van prestroom-duw-algoritmen enorm kunnen beïnvloeden. Hierdoor krijgen we mogelijk een goed zicht op de algemene onderlinge relatie van FIFO, HHE en ES met betrekking tot hun uitvoeringstijden.

Experiment 2 (EXP_2)

Door een vergelijking van de uitvoering van FIFO, $\text{FIFO}_{\frac{1}{2}}$ en FIFO_s op instanties van WLM, proberen we de waarde van de heuristiek die de maximale stroom op voorhand probeert te schatten, te achterhalen. We hebben hierbij gekozen voor WLM vanwege zijn eigenschap met betrekking tot de minimale boogsnede. Op die manier kunnen we, voornamelijk in geval van een minimale boogsnede kort bij de afvoer, makkelijk controleren of FIFO_s er effectief voor zorgt dat er gemiddeld minder reststroom naar de bron moet afvloeien (tweede fase van de uitvoering van het prestroom-duw-algoritme

wordt versneld), hetgeen de motivatie is waarom deze heuristiek het leven werd ingeroepen. Met behulp van de uitvoeringstijden van $\text{FIFO}_{\frac{1}{2}}$ proberen we vervolgens in te schatten wat de eventuele tijdswinst precies is voor de tweede fase wanneer we gebruik maken van FIFO_s.

Experiment 3 (EXP_3)

In experiment EXP_3 bestuderen we het gedrag van FIFO_g en HHE_g bij een uitvoering ervan op instanties van AD, GL, GW en WLM. We voerden beide algoritmen twee keer uit op al deze instanties, één keer met een globale verhoging na elke n ontladingen en één keer met een globale verhoging na elke m ontladingen. We kunnen op die manier nagaan in termen van welke parameter, n of m, we de frequentie voor het uitvoeren van globale verhogingen het beste kiezen. We herinneren de lezer hierbij aan de vergelijking die er tussen beide frequenties gemaakt werd in Sectie 7.4. Instanties van de netwerkklasse AD zullen ons meer bepaald een idee kunnen geven over hoe zeer de globale-verhoog-heuristiek afhankelijk is van de dichtheid van een stroomnetwerk. Deze instanties zijn immers zéér dichte stroomnetwerken. Merk op dat alle, hierna, besproken experimenten die betrekking hebben op de globale-verhoog-heuristiek, werden uitgevoerd op basis van de frequentie die de beste resultaten heeft opgeleverd in dit experiment.

Experiment 4 (EXP_4)

Op het einde van Sectie 7.4 werd een combinatie van de kloof- en globale-verhoog-heuristiek gesuggereerd. We bekijken in dit experiment wat de invloed is van een dergelijke combinatie op de uitvoeringstijden van het FIFO- en hoogste-hoogte-eerst prestroom-duw-algoritme. We hebben daartoe FIFO_{gk} en HHE_{gk} uitgevoerd op dezelfde instanties van netwerkklassen AD, GL, GW en WL als in EXP_3. We willen natuurlijk ook weten of dit eventueel betere resultaten oplevert dan het gebruik van enkel de kloofheuristiek of enkel de globale-verhoog-heuristiek. We hebben daarom FIFO_k en HHE_k ook uitgevoerd op al deze instanties. Voor resultaten van het enkel gebruik van de globale-verhoog-heuristiek doen we beroep op de resultaten van EXP_3.

Experiment 5 (EXP_5)

We hebben tot slot FIFO en de belangrijkste varianten van het FIFO prestroom-duw-algoritme, het sowieso meest robuuste prestroom-duw-algoritme, vergeleken over instanties van netwerkklassen GL, GW en WLM. De keuze van net deze netwerkklassen heeft nog steeds dezelfde reden als in EXP_1. Met belangrijkste varianten bedoelen we de in de theorie het meest bestudeerde, nl. FIFO_k, FIFO_g en FIFO_{gk}.

8.2 Werkwijze

De experimenten werden uitgevoerd op een systeem met Pentium IV 3.0 GHz processor en een intern geheugen van 1024MB.

Instanties van netwerkklassen werden gegenereerd met generatoren van de DIMACS[1] Challenge[2] die men kan downloaden van een FTP-server[3].

Bij het uitvoeren van een experiment werden er per type stroomnetwerk twintig verschillende instanties gegenereerd en volgens het invoerformaat (*.max*) van het DIMACS Maximum Flow Problem Format[4] bewaard. Bestanden die zijn opgebouwd volgens dit formaat zullen we doorheen het verdere verloop van deze masterproef benoemen als max-bestanden. Alle instanties van één type stroomnetwerk werden bekomen via eenzelfde instelling van de parameters van zijn netwerkklasse. De keuze van deze hun waarden hangt niet alleen af van geheugengebruik, maar vooral van tijdsbeperkingen. We geven later een overzicht van de instelling van de parameters per gebruikt type stroomnetwerk.

Een willekeurigheid onder de instanties wordt verzorgd door de generatoren. Deze willekeurigheid heeft betrekking op de capaciteiten van de bogen. De capaciteiten van een GL- of GW-instantie bijvoorbeeld worden, voor bogen tussen elk paar lagen, willekeurig gekozen uit het bereik $[c_1, c_2]$. Daarenboven worden bij dit soort stroomnetwerken de verbindingen tussen twee lagen ook willekeurig gekozen.

Elk te bestuderen algoritme werd vervolgens uitgevoerd op de gegenereerde instanties en voor elke combinatie van algoritme met type stroomnetwerk werden de volgende gegevens bewaard:

- onverzadigde duwoperaties,
- verzadigde duwoperaties,
- verhoogoperaties,
- ontdekte kloven,
- globale verhogingen,
- ontladingen,
- uitvoeringstijd.

Alle gegevens, behalve de uitvoeringstijd, drukken we uit in aantallen en worden bijgehouden door de algemene implementatie van een prestroomduw-algoritme, zie Hoofdstuk 10. De uitvoeringstijd drukken we uit in seconden (s) en omvat eigenlijk twee componenten:

[1] Center for Discrete Mathematics and Theoretical Computer Science
[2] http://dimacs.rutgers.edu/
[3] ftp://dimacs.rutgers.edu/pub/netflow/generators/network/
[4] http://www.avglab.com/andrew/CATS/maxflow_formats.html

- *Gebruikerstijd*: de tijd nodig om de applicatie zijn eigen code uit te voeren;
- *Systeemtijd*: de tijd nodig om op aanvraag van de applicatie, bijvoorbeeld voor I/O, code van het besturingssysteem uit te voeren.

Beide vormen samen de effectieve CPU-tijd die aan de uitvoering van een applicatie besteed werd. We meten de uitvoeringstijd dus volgens de CPU-tijd om effecten van andere systeemactiviteiten niet in rekening te brengen en zijn natuurlijk vooral geïnteresseerd in de gebruikerstijd. We willen immers enkel de effectieve uitvoeringstijd van een algoritme bestuderen zonder hierbij de allocatie van geheugen en het schrijven naar bestanden in rekening te brengen.

Om de gebruikerstijd te meten, maken we gebruik van het Management-package uit Java 1.5 om de JVM[5] te monitoren. Via de static methods van zijn ManagementFactory class kunnen verschillende zogenaamde 'MXBean'-objecten worden teruggegeven. Deze objecten kunnen informatie van de JVM rapporteren en meer bepaald de gebruikers- en CPU-tijd van een thread opvragen. Uit deze informatie kunnen we de systeemtijd berekenen.

We illustreren dit alles even met onderstaande voorbeeldcode. We roepen eerst ManagementFactory.getThreadMXBean() aan om een ThreadMXBean terug te krijgen die huidige JVM threads beschrijft. De functie getCurrentThreadCpuTime() geeft de CPU-tijd terug van de thread waarin we de code voor meting van de twee tijden plaatsen. De functie getCurrentThreadUserTime() geeft daarentegen de gebruikerstijd van deze thread weer. Beide tijden worden uitgedrukt in nanoseconden. Alvorens we één van beide functies mogen oproepen, moeten we eerst controleren of de JVM implementatie of het besturingssysteem dergelijke aanvragen van CPU- en/of gebruikerstijd ondersteunt.

```
import java.lang.management.*;

/**
 * Geeft de gebruikerstijd terug in nanoseconden.
 */
public long getUserTime() {
    ThreadMXBean bean = ManagementFactory.getThreadMXBean();
    return bean.isCurrentThreadCpuTimeSupported() ?
        bean.getCurrentThreadUserTime() : 0L;
}

/**
 * Geeft de systeemtijd terug in nanoseconden.
```

[5] Java Virtual Machine

```
 */
public long getSystemTime() {
    ThreadMXBean bean = ManagementFactory.getThreadMXBean();
    return bean.isCurrentThreadCpuTimeSupported() ?
        (bean.getCurrentThreadCpuTime() -
        bean.getCurrentThreadUserTime()) : 0L;
}
```

De bovenstaande functies geven de gebruikers- en systeemtijd sinds de desbetreffende thread gestart is, terug. Om nu vervolgens de gebruikers- en/of systeemtijd van een prestroom-duw-algoritme in combinatie met één of meerdere heuristieken te bepalen, roepen we deze functies voor en na de uitvoering van het algoritme op en nemen het verschil van de tijden die ze teruggegeven hebben.

```
long startUserTime   = getUserTime();
long startSystemTime = getSystemTime();

/* Uitvoering algoritme (task) */

long taskUserTime    = getUserTime() - startUserTime;
long taskSystemTime  = getSystemTime() - startSystemTime;
```

Alle gegevens werden vervolgens telkens ingevuld in een soort template met reeds ingevulde rekenformules en vervolgens naar een CSV[6]-bestand geschreven. Bij het openen van een dergelijk CSV-bestand met een rekenbladprogramma zoals Excel, zijn de gewenste statistische resultaten:

- minimum,
- gemiddelde,
- maximum,
- standaardafwijking,

per gegeven, vervolgens reeds automatisch berekend.

De oplossing van het algoritme, meer bepaald de maximale stroomwaarde en de verdeling ervan over het stroomnetwerk, schreven we telkens weg naar een bestand volgens het uitvoerformaat (*.sol*) van het DIMACS Maximum Flow Problem Format.

[6] http://en.wikipedia.org/wiki/Comma-separated_values

8.3 Resultaten

We interpreteren in deze sectie de resultaten die zijn voortgekomen uit de experimenten. In Appendix A kunnen telkens voor ieder experiment de gemiddelden van de belangrijkste en relevante gegevens worden teruggevonden. Alle experimenten werden uitgevoerd op instanties van de volgende type stroomnetwerken, bepaald door een specifieke instelling van de parameters van hun netwerkklasse:

Stroomnetwerken		n	m
AD	256 seed	256	32640
	321 seed	321	51360
	403 seed	403	81003
GL	$a = 6\ b = 31\ c_1 = 1\ c_2 = 10000$	1116	4800
	$a = 7\ b = 42\ c_1 = 1\ c_2 = 10000$	2058	9065
	$a = 8\ b = 64\ c_1 = 1\ c_2 = 10000$	4096	18368
GW	$a = 16\ b = 4\ c_1 = 1\ c_2 = 10000$	1024	4608
	$a = 21\ b = 5\ c_1 = 1\ c_2 = 10000$	2205	10164
	$a = 28\ b = 5\ c_1 = 1\ c_2 = 10000$	3920	18256
WLM	6 64 4 5	258	1236
	6 128 4 8	514	3975
	6 256 4 8	1026	8069
	6 512 4 11	2050	22279
	6 1024 4 16	4098	65019

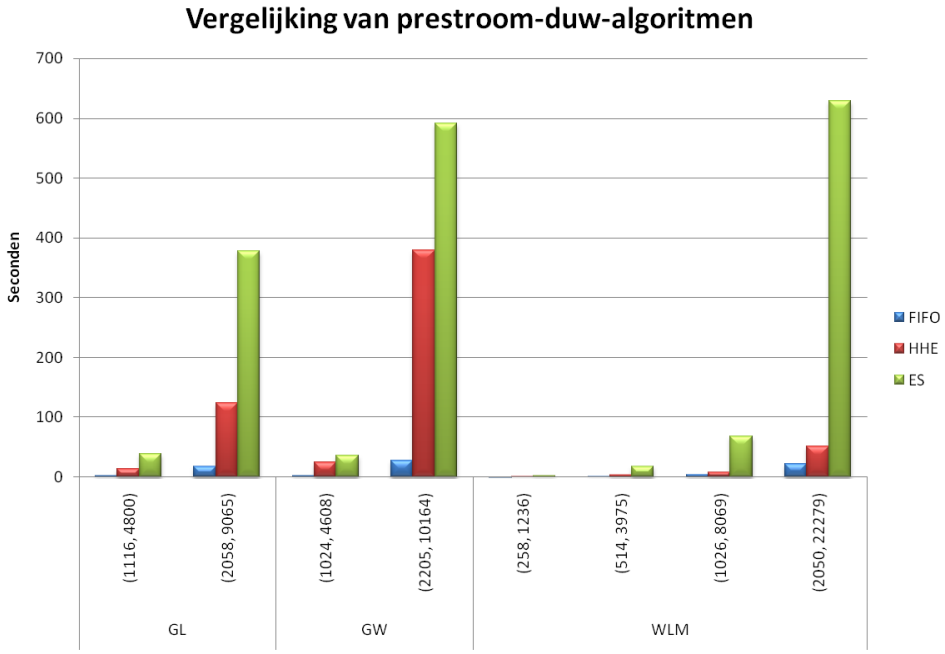
Tabel 8.1: Overzicht van de verschillende soorten gebruikte stroomnetwerken.

Alle resultaten, inclusief de gecompileerde versie van elke generator, kunnen teruggevonden worden in de mapjes resultaten_experimenten\ en stroomnetwerk_generatoren\ op de bijgeleverde DVD.

8.3.1 $\mathbf{EXP}_1$

Uit Figuur 8.1 zien we duidelijk dat FIFO en HHE dominant zijn ten opzichte van ES. Nochtans is het zo dat ES in het algemeen minder ontladingen uitvoert dan FIFO en HHE. Het vermoeden is dat ES voornamelijk tijd verliest vanwege duwoperaties die niets over de toegelaten bogen duwen, aangezien ES er steeds voor zorgt dat geen enkele reststroom ooit Δ overstijgt en de knoop naar waar er geduwd wordt als reststroom mogelijk Δ heeft.

Volgens [16] is een uitvoering van HHE, op de beschouwde instanties, steeds sneller dan FIFO. Bij dit experiment is dit echter niet het geval, ondanks het feit dat HHE minder verhoogoperaties introduceert en deze



Figuur 8.1: Vergelijking prestroom-duw-algoritmen.

het tijdsverlies op de duwoperaties makkelijk zouden kunnen overbruggen vanwege de hogere individuele kost van een verhoogoperatie. HHE zou waarschijnlijk ook voor dit experiment de snelste gebruikerstijden hebben, aangezien de array van lijsten B_i geïmplementeerd werd met behulp van de java.util.Vector. Meer bepaald de efficiëntie van het vooraan verwijderen werd bij deze keuze over het hoofd gezien, hoewel hier bij de implementatie van de datastructuur voor FIFO wel rekening mee gehouden werd. ES maakt ook gebruik van dezelfde datastructuur als HHE, maar dit neemt niet weg dat de andere toch dominant blijven. Vanwege tijdsredenen kon het experiment niet opnieuw uitgevoerd worden met een aangepaste datastructuur. De datastructuur zelf werd ondertussen natuurlijk aangepast.

De standaardafwijkingen van de gebruikerstijden leren ons echter wel dat FIFO stabieler is over de hele lijn, hetgeen overeenkomt met zijn robuustheid die we reeds hebben opgemerkt in 6.10.3, waar we FIFO vergeleken hebben met HHE.

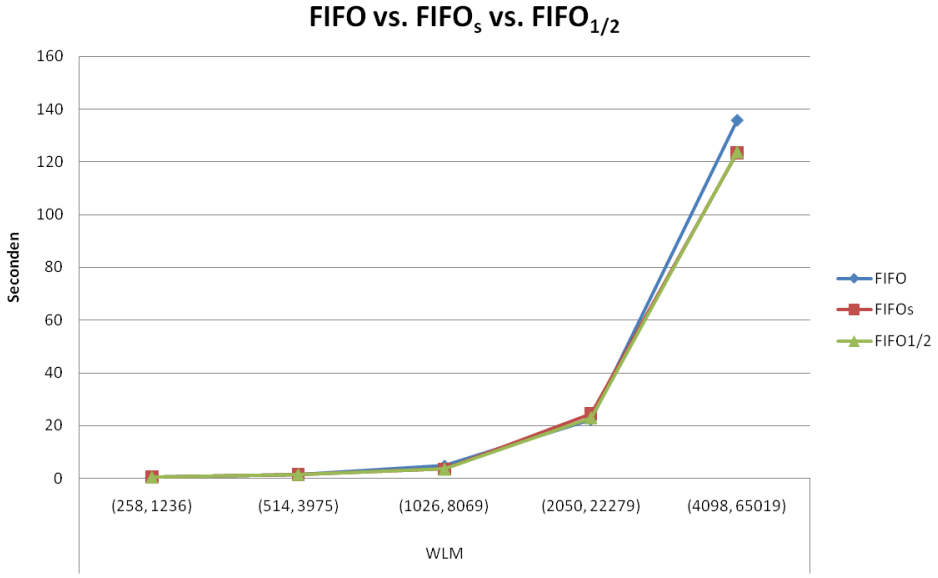
Vanwege de dominantie van FIFO en HHE ten opzichte van ES werd er niet verder geëxperimenteerd met dit laatste prestroom-duw-algoritme. We merken ook op dat elke verdere vergelijking tussen verschillende varianten van HHE geen probleem vormt voor de geldigheid van de vergelijking aangezien de verzamelde gegevens zich sowieso op dezelfde manier tot elkaar verhouden als wanneer de datastructuur reeds was aangepast. Het aantal

keer dat er vooraan een knoop verwijderd wordt, verandert hierdoor, voor elke variant, immers niet. Varianten van HHE en FIFO vergelijken we niet rechtstreeks met betrekking tot gebruikerstijd.

8.3.2 EXP_2

$FIFO_s$ introduceert extra operaties ten opzichte van FIFO. Dit komt vermoedelijk door de toevoeging van de artificiële bron s_a aan elk stroomnetwerk. $FIFO_s$ heeft echter soms een betere gemiddelde gebruikerstijd dan FIFO. Dit is het geval wanneer $FIFO_s$ stabielere uitvoeringen heeft gehad dan FIFO. De stijging in verzadigde duwoperaties is mogelijk de oorzaak van de betere gebruikerstijd.

De gemiddelde gebruikerstijd van $FIFO_s$ komt dan ongeveer overeen met die van $FIFO_{\frac{1}{2}}$, en dus met een tijdswinst van quasi 100% van de tijd die de tweede fase bij FIFO inneemt. $FIFO_s$ maakt FIFO op dergelijke momenten nog stabieler dan het al is en wanneer we naar grotere stroomnetwerken gaan, lijkt het gebruik van $FIFO_{\frac{1}{2}}$ vervangbaar door $FIFO_s$ om in dezelfde tijd, in plaats van enkel de maximale stroom, ook de verdeling ervan over het stroomnetwerk te berekenen. We maken deze laatste opmerking duidelijk aan de hand van Figuur 8.2.



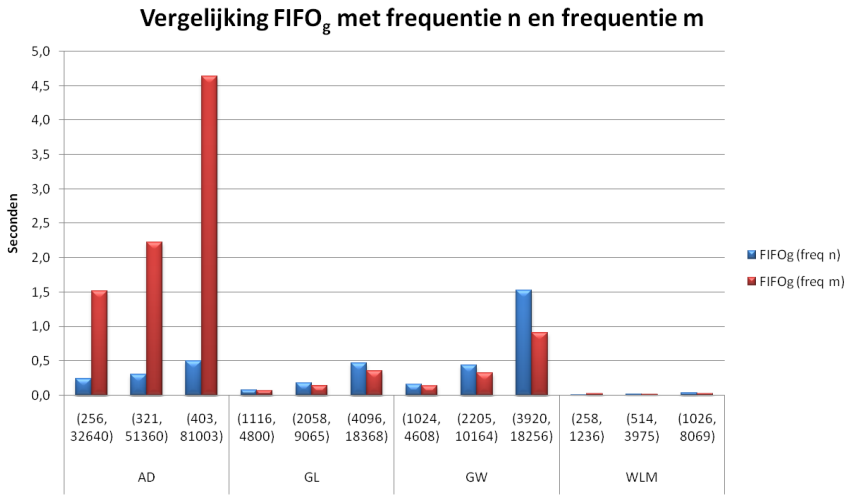
Figuur 8.2: FIFO vs. $FIFO_s$ vs. $FIFO_{\frac{1}{2}}$.

$FIFO_s$ heeft stabielere uitvoeringen gehad ten opzichte van FIFO indien FIFO die ook had ten opzichte van zijn uitvoeringen op andere stroomnetwerken. In geval van WLM spreken we van de onstabielste uitvoeringen

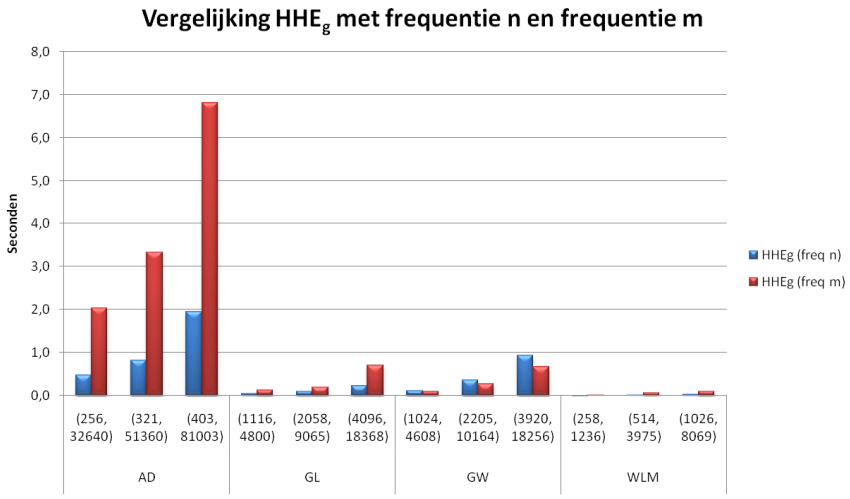
indien voor 50% van de instanties de minimale boogsnede kort bij de afvoer ligt en voor de overige 50% kort bij de bron. FIFO_s lijkt dus enkel tijd te winnen indien de minimale boogsnede ofwel steeds kort bij de bron ligt, ofwel steeds kort bij de afvoer. Indien de minimale boogsnede steeds kort bij de bron ligt, kan er quasi geen tijd gewonnen worden in de tweede fase. Dit bevestigt ons vermoeden dat FIFO_s gemiddeld gezien minder reststroom naar de bron doet afvloeien. De bekomen metingen geven hier echter niet genoeg uitsluitsel over. Een nieuw experiment waarbij FIFO_s meerdere willekeurige boogsneden in beschouwing neemt bij het bepalen van $c(s_a, s)$ zou dit misschien wel kunnen bieden.

8.3.3 EXP_3

Uit Figuur 8.3 blijkt dat m de beste frequentie is voor FIFO_g bij het uitvoeren van de globale-verhoog-heuristiek. Voor HHE_g is dit n, zie Figuur 8.4. De reden hiervoor is ongetwijfeld het feit dat actieve knopen in het geval van een FIFO_g uniformer behandeld worden waardoor er minder nood is aan globale verhogingen. Een globale verhoging om de n ontladingen zal, meer dan bij HHE_g, weinig tot geen verandering brengen aan de hoogtes van de knopen aangezien deze bij FIFO_g in het algemeen veel minder afwijken ten opzichte van hun exacte hoogte. Met m als frequentie wordt er voor FIFO_g dus bijgevolg op een meer doeltreffende en efficiënte manier omgesprongen met globale verhogingen.



Figuur 8.3: Vergelijking van FIFO_g met frequentie n en m.



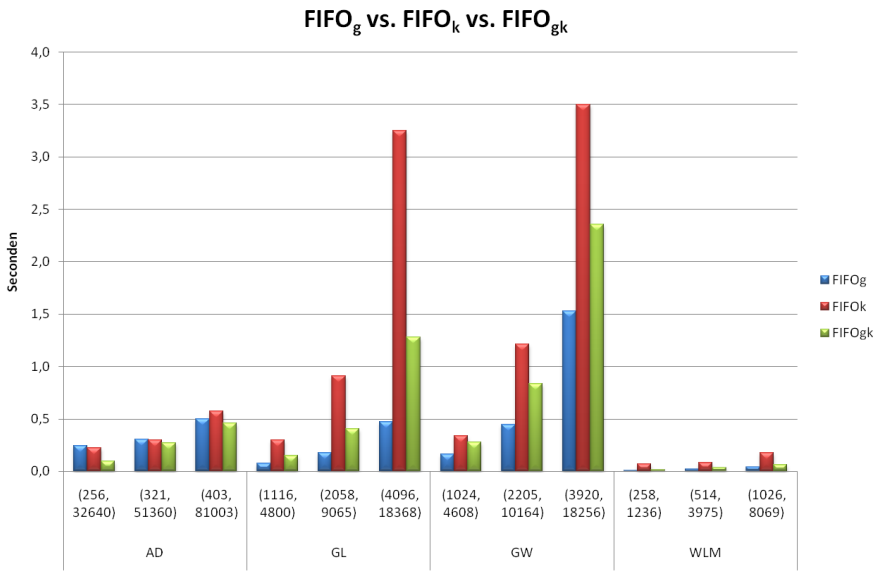
Figuur 8.4: Vergelijking van HHE$_g$ met frequentie n en m.

Opvallend is dat beide algoritmen het best n gebruiken als frequentie bij hun uitvoering op instanties van netwerkklasse AD. Instanties van deze klasse zijn immers zéér dichte stroomnetwerken en de keuze van m als frequentie levert blijkbaar te weinig globale verhogingen op om een betere uitvoering te garanderen dan met n als frequentie.

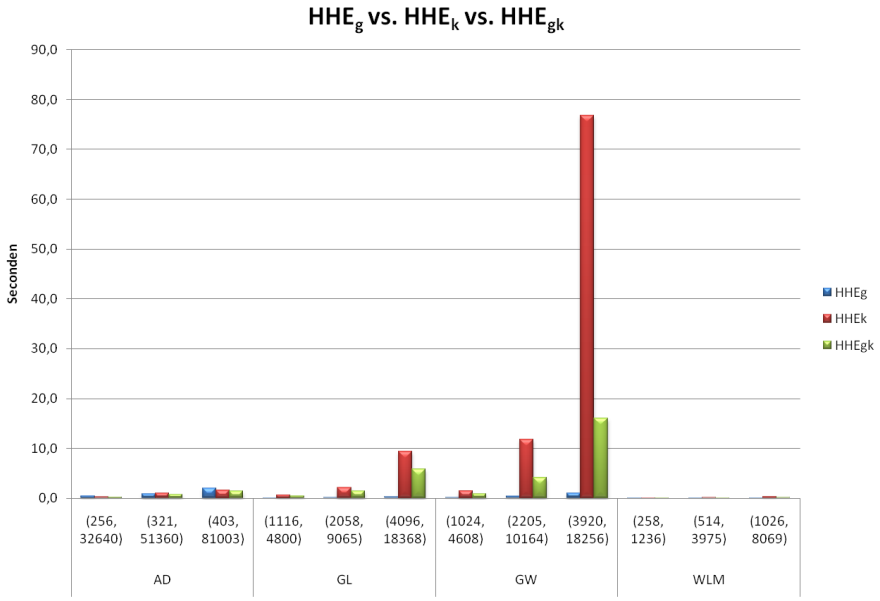
Geen van beide frequenties springt dus echt als winnaar naar voor. De meest robuuste lijkt, vanwege de vaststelling in verband met zéér dichte stroomnetwerken en het feit dat een overschakeling van n naar m voor HHE$_g$ drastischer lijkt dan een overschakeling van m naar n voor FIFO$_g$, toch n te zijn. We kiezen dus uiteindelijk toch voor n als frequentie voor het bepalen na hoeveel ontladingen we een globale verhoging uitvoeren.

8.3.4 EXP$_4$

In dit experiment hebben we ons toegespitst op de onderling relatie tussen de globale-verhoog- en kloofheuristiek. We interpreteren hiertoe Figuren 8.5 en 8.6.



Figuur 8.5: FIFO_g vs. FIFO_k vs. FIFO_{gk}.



Figuur 8.6: HHE_g vs. HHE_k vs. HHE_{gk}.

De globale-verhoog-heuristiek in combinatie met de kloofheuristiek bewijst zijn nut het meest voor de zéér dichte stroomnetwerken van netwerkklasse AD. De aanvulling met de kloofheuristiek is hierbij het meest interes-

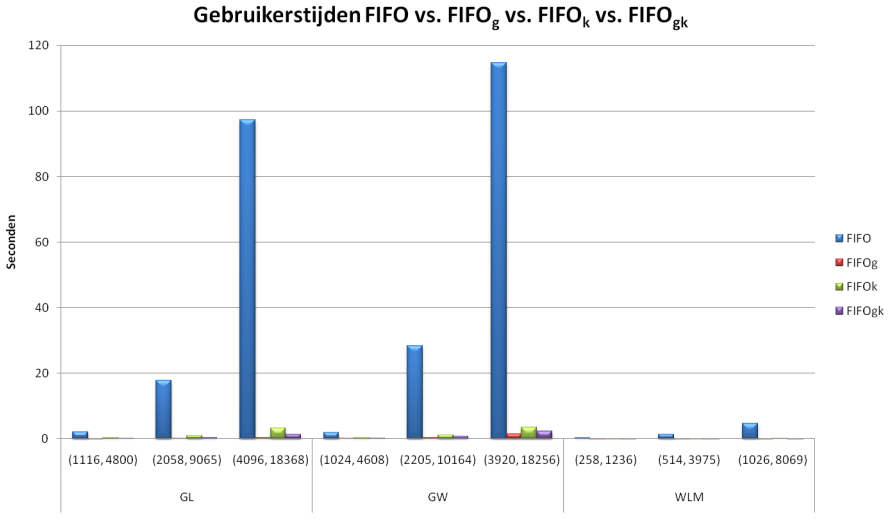
sant voor HHE_g vanwege de volgende redenen. Bij HHE_g gaan de actieve knopen aan de bronzijde van een kloof alle aandacht krijgen tot ze boven de bron gestegen zijn of er een globale verhoging is uitgevoerd. De andere actieve knopen, aan de andere zijde van de kloof, worden gedurende deze tijd verwaarloosd. Deze tijd van verwaarlozing is minder groot bij FIFO_g aangezien actieve knopen op een uniformere manier behandeld worden waardoor alle actieve knopen, zowel voor als achter een kloof, aandacht krijgen in de periode tussen twee globale verhogingen.

Daaropvolgend zijn eerst FIFO_k en HHE_k het meest geschikt voor een uitvoering op instanties van AD. De vele bogen van deze instanties moeten voor elke globale verhoging natuurlijk overlopen worden, iets waar de kloofheuristiek geen last van heeft aangezien deze enkel afhankelijk is van het aantal knopen.

Voor de rest komt het er eigenlijk op neer dat de globale-verhoog-heuristiek superieur is aan de globale-verhoog-heuristiek in combinatie met de kloofheuristiek en nog meer aan de toepassing van enkel de kloofheuristiek.

8.3.5 EXP_5

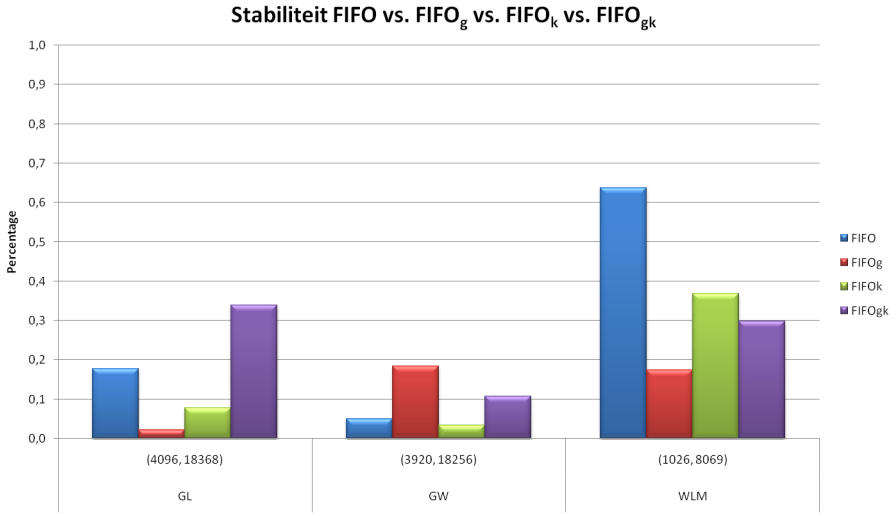
Uit de vergelijking van FIFO, FIFO_g, FIFO_k en FIFO_{gk} op instanties van netwerkklassen GL, GW en WLM leren we dat de verhoudingen tussen de operaties zich in het algemeen op dezelfde manier gedragen als de verhoudingen in gebruikerstijd. Merk opnieuw de superioriteit van de globale-verhoogheuristiek op wanneer we naar Figuur 8.7 kijken.



Figuur 8.7: Gemiddelde gebruikerstijden van FIFO, FIFO_g, FIFO_k en FIFO_{gk}.

Uitgaande van een normale verdeling, hebben we, op basis van de statistische resultaten, de stabiliteit berekend van de algoritmen op de instanties

van de beschouwde netwerkklassen. De resultaten hiervan worden getoond in Figuur 8.8. Hoe hoger het percentage, des te stabieler is het algoritme op de soort instantie. Merk op dat FIFO het stabielste is, gevolgd door de globale-verhoog-heuristiek in combinatie met de kloofheuristiek. Deze laatste waarneming doet ons steeds meer en meer het belang van deze combinatie als robuuste heuristiek inzien.



Figuur 8.8: Stabiliteit van FIFO, FIFO$_g$, FIFO$_k$ en FIFO$_{gk}$ uitgedrukt in percentages.

Hoofdstuk 9

Stroomnetwerktekenalgoritme

Tijdens de implementatie van de applicatie, waarvan we ook tevens zijn werking beschrijven in Hoofdstuk 10, zijn we tot de vaststelling gekomen dat het visualiseren van een stroomnetwerk in 3D zeker niet zo triviaal is. De grootste uitdaging bevindt zich in het tekenen ervan in het vlak (2D ruimte), van waaruit we implementatiegewijs vertrekken. Deze tekening in het vlak moet het stroomnetwerk zo overzichtelijk en leesbaar mogelijk visualiseren.

Aangezien een stroomnetwerk in principe een graaf is, zouden we hiervoor het krachtgestuurd graaftekenalgoritme uit bachelorproef [14] letterlijk kunnen gebruiken. Het probleem met dit algoritme is dat het een graaf visualiseert vanuit het standpunt van de discipline graaf tekenen die probeert de structuur van een graaf, bepaald door zijn knopen en bogen, in de verf te zetten. Het algoritme probeert met andere woorden een tekening te vinden zodanig dat de bogen van de graaf elkaar zo weinig mogelijk snijden en onderling ongeveer allemaal dezelfde lengte hebben.

We werken hier echter met een stroomnetwerk en wensen de uitvoering van een prestroom-duw-algoritme hierop visueel te illustreren binnen een 3D ruimte. Het stroomnetwerk dient nu zo overzichtelijk en leesbaar mogelijk gevisualiseerd te worden met het oog op het begrijpen van het prestroom-duw-algoritme in plaats van het begrijpen van de structuur van de graaf. Het is daarom belangrijk dat we, naast de structuur van de graaf, ook de bron, de afvoer en de zogenaamde 'trap' die geleidelijk aan gevormd wordt over de intermediaire knopen heen, in gedachte houden bij het tekenen van een stroomnetwerk in het vlak. We hebben hiertoe het krachtgestuurd graaftekenalgoritme aangepast.

Vooraleer we bespreken wat we precies hebben aangepast en waarom, leggen we kort de werking van het krachtgestuurd graaftekenalgoritme intuïtief uit.

9.1 Krachtgestuurd graaf tekenen

Wanneer we een graaf tekenen op een krachtgestuurde manier, bekijken we de graaf als een fysisch model. We beschouwen de graaf als een systeem van elektrisch geladen deeltjes, de knopen, waartussen er krachten heersen en die verbonden zijn met elkaar door veren, de bogen. De aanwezige krachten zijn meer bepaald de afstotings- en aantrekkingskrachten tussen de knopen. De knopen stoten elkaar af en de bogen trekken de deeltjes naar elkaar toe. Door de uitoefening van deze krachten bevat de graaf kinetische energie rondom elke knoop.

Het graaftekenalgoritme probeert nu de totale kinetische energie in de graaf naar nul te reduceren. Het berekent hiertoe, op basis van de posities van de knopen (in het xy-vlak, met $x \in [0, 1]$ en $y \in [0, 1]$) en de aanwezige krachten, de snelheden van de knopen op vaste tijdstippen. Merk op dat de posities van de knopen willekeurig gekozen worden bij aanvang van het algoritme. Op basis van de snelheden van de knopen wordt hun afgelegde afstand, volgens de x- en/of y-as, over een periode tussen twee tijdstippen berekend. De positie van elke knoop wordt hier vervolgens aan aangepast. Dit proces herhaalt zich tot op het moment dat de kinetische energie (opgesplitst in een x- en y-component), die er heerst rondom elke knoop, quasi nul is.

Voor meer details omtrent dit algoritme verwijzen we naar Hoofdstuk 3 van [14].

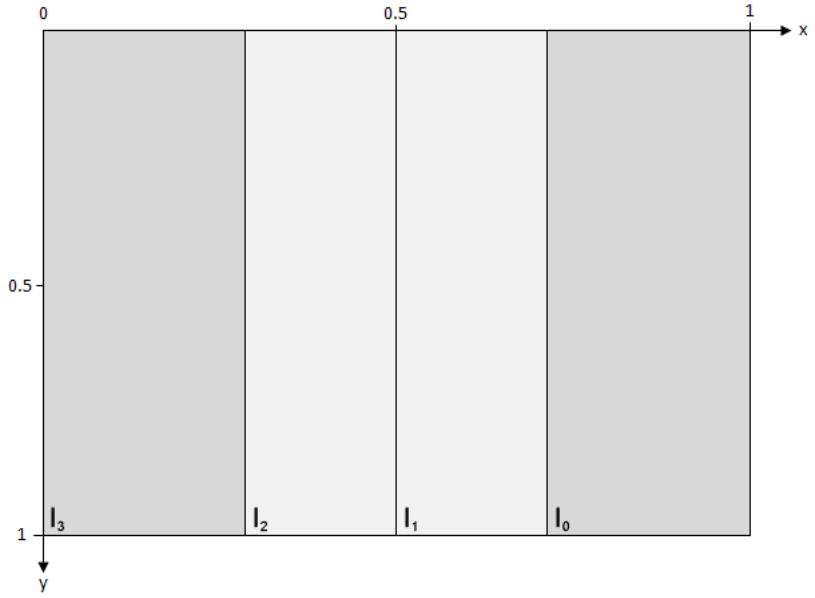
9.2 Krachtgestuurd stroomnetwerk tekenen

We bespreken nu hoe we het krachtgestuurd graaftekenalgoritme hebben aangepast om een stroomnetwerk zo te tekenen in het xy-vlak dat de reeds aangehaalde doelstellingen, voor de visualisatie ervan in een 3D ruimte, bereikt worden.

In plaats van, vóór de aanvang van het algoritme, de posities van de knopen volledig willekeurig te kiezen, verdelen we de knopen van het stroomnetwerk over verticaal geplaatste *stroken* I. Laten we ter illustratie het stroomnetwerk uit Figuur 5.2 beschouwen.

Voor de bron en de afvoer reserveren we sowieso twee stroken. Om te bepalen hoeveel stroken we nodig hebben om de intermediaire knopen in te plaatsen, berekenen we m.b.v. een BFS de afstand $d(v, t)$ van elke intermediaire knoop v tot de afvoer t. Merk op dat de bogen van het residueel stroomnetwerk vóór aanvang van een uitvoering van een prestroom-duw-algoritme steeds overeenkomen met de bogen van het bijhorende stroomnetwerk en we dus zo ook de kortste afstand in het stroomnetwerk berekenen. De grootste kortste afstand g bepaalt nu het aantal van deze laatste stroken. We splitsen vervolgens het vlak op in $g + 2$ stroken. De stroken $I_0, \ldots, I_{g+1}$ zijn nu van

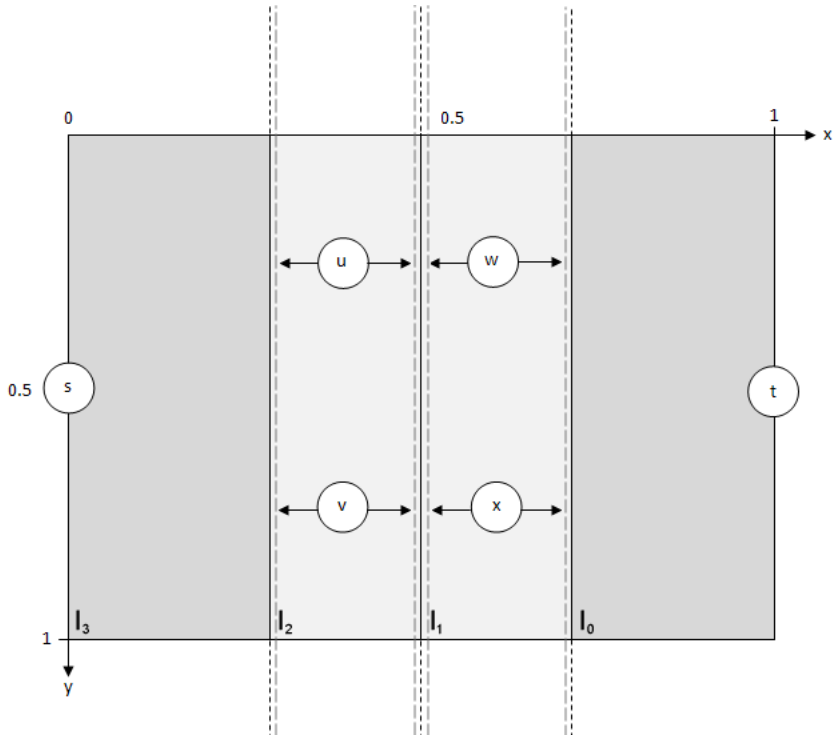
rechts naar links geplaatst in het bereik $x \in [0, 1]$, waarbij stroken I_0 en I_{g+1} breder zijn dan de andere stroken. Deze situatie wordt voor het beschouwde stroomnetwerk weergegeven in Figuur 9.1.



Figuur 9.1: Illustratie verdeling stroken voor het stroomnetwerk in Figuur 5.2.

We verdelen alle knopen van het stroomnetwerk in een volgende fase van rechts naar links over de verschillende stroken, zie Figuur 9.2. De afvoer plaatsen we helemaal rechts in het midden van strook I_0. De intermediaire knopen v plaatsen we in strook I_i waarbij $i = d(v, t)$. Alle intermediaire knopen in een strook worden daarbij netjes verdeeld over de lengte van de strook. De bron plaatsen we helemaal links in het midden van strook I_{g+1}. Merk op dat, door de keuze van bredere stroken voor de bron en de afvoer, beide knopen duidelijk te onderscheiden zijn van de rest van het stroomnetwerk.

De knopen zijn nu netjes gepositioneerd in het vlak. De mogelijkheid bestaat nu echter nog steeds dat de knopen zo gepositioneerd zijn dat de bogen, die deze met elkaar verbinden, de duidelijkheid van de visualisatie van het stroomnetwerk verstoren. Om dit op te lossen, laten we een aangepaste versie van het krachtgestuurd graaftekenalgoritme los op de posities van de knopen in het vlak. We laten hierbij de krachten volledig inwerken op de knopen, net als in het originele, maar zorgen ervoor dat de knopen enkel volgens de y-as buiten de originele ruimte van hun strook bewegen en de intermediaire knopen volgens de x-as tot een bepaalde marge van de grenzen van hun strook. Deze marge zorgt ervoor dat intermediaire kno-



Figuur 9.2: Illustratie verdeling knopen van het stroomnetwerk in Figuur 5.2 over zijn stroken.

pen van verschillende stroken niet vlak naast mekaar worden geplaatst. De bron en afvoer worden telkens terug op hun originele positie volgens de x-as geplaatst. Door het feit dat we de knopen enkel in de y-richting laten bewegen, raken de knopen hun kinetische energie, volgens de richting van de x-as, niet kwijt. We nemen de x-component van de kinetische energie dan ook niet in beschouwing als stopcriterium voor het algoritme aangezien dit anders oneindig zou doorgaan aangezien de horizontale bewegingsruimte van de knopen beperkt is en deze hierdoor hun bijhorende energie niet kwijt raken.

We doen dit hele proces slechts één keer, namelijk vóór de aanvang van de visualisatie van een prestroom-duw-algoritme. Indien we dit proces ook zouden uitvoeren tijdens een dergelijke visualisatie, zouden de knopen mogelijk veranderen van strook aangezien het residueel stroomnetwerk ondertussen werd gewijzigd. We willen, in de 3D visualisatie, knopen echter gefixeerd houden in het vlak en enkel laten stijgen in de 3D ruimte om er voor te zorgen dat de gebruiker van de applicatie het noorden niet verliest.

Hoofdstuk 10

Implementatie en werking applicatie

De applicatie die we hebben ontwikkeld, werd geschreven in Java en gebruikt om experimenten mee uit te voeren en de besproken technieken en effecten met betrekking tot prestroom-duw-algoritmen in te zien of nog beter te leren begrijpen. In dit laatste hoofdstuk bespreken we in het kort de belangrijkste aspecten met betrekking tot deze applicatie en zijn werking.

10.1 Architectuur en implementatie

De implementatie is opgebouwd volgens het welgekende MVC[1]-model. We bespreken enkel de belangrijkste delen van de applicatie, nl. degene die het belangrijkste onderdeel van het model, de view en de controller vormen.

Zoals uit Appendix B zal blijken, is de view opgebouwd uit 2 grote componenten, nl. de editor en de animator. Beiden maken gebruik van het graaf- en stroomnetwerktekenalgoritme. De animator werd geprogrammeerd met behulp van Java OpenGL of kortweg JOGL. Het is een wrapperbibliotheek die OpenGL-programmatie mogelijk maakt in de programmeertaal Java. Kennis over deze bibliotheek werd vergaard met behulp van [12].

De controller is tweeledig in de zin dat er een view- en modelcontroller zijn die op een hoger niveau nog eens door de hoofdcontroller worden aangestuurd. Deze hoofdcontroller geeft gewoon acties tussen beide controllers door en voorziet dus een extra scheiding tussen het view- en modelgedeelte van de controller.

Tot het model behoren de datastructuren van een graaf en hier bovenop een stroomnetwerk en de geïmplementeerde prestroom-duw-algoritmen FIFO, FIFO_s, HHE en ES. Deze algoritmen werden geïmplementeerd bovenop een generiek prestroom-duw-algoritme waarin de ontladingsprocedure, duw- en

[1]Model-View-Controller (http://en.wikipedia.org/wiki/Model-view-controller)

verhoogoperatie en mogelijkheid tot activering van heuristieken zijn voorzien. Dit zorgt ervoor dat deze algoritmen enkel hun strategie van het selecteren van actieve knopen omvat, hetgeen netjes overeenkomt met de theoretische beschrijving. Het bestaan van het generiek prestroom-duw-algoritme zorgt er voor dat nieuwe prestroom-duw-algoritmen in een mum van tijd geïmplementeerd kunnen worden. Daarenboven werd er tevens voor gezorgd dat de Java-klassen, die overeenkomen met deze algoritmen, tijdens de uitvoering van de applicatie inplugbaar zijn, hetgeen ervoor zorgt dat enkel een compilatie van de overeenstemmende Java-klasse dus vereist is. We hebben hier duidelijk te maken met een invulling van het strategy pattern[2].

Als laatste punt merken we nog op dat stroomnetwerken door de model-controller worden ingelezen in het model gebruikmakende van max-bestanden. Een dergelijk bestand ziet er bijvoorbeeld uit als volgt:

```
p max 6 6
n 1 s
n 6 t

a 1 2 1200
a 2 3 1200
a 3 4 1200
a 4 5 1200
a 5 2 1200
a 5 6 50
```

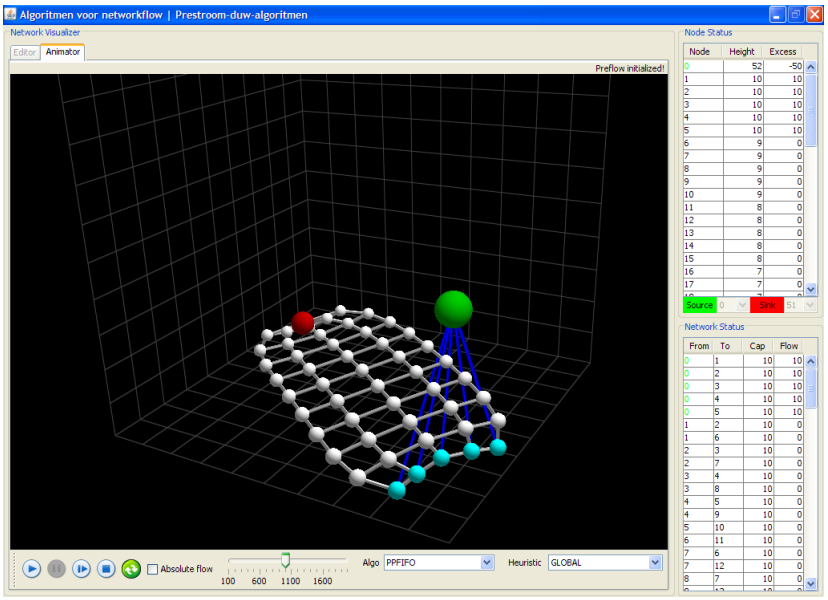
Het bestand begint met een probleemregel, voorafgegaan door de letter p. Vervolgens worden de bron en afvoer gedefinieerd en daarna de bogen met bijhorende capaciteit.

10.2 Illustratie werking

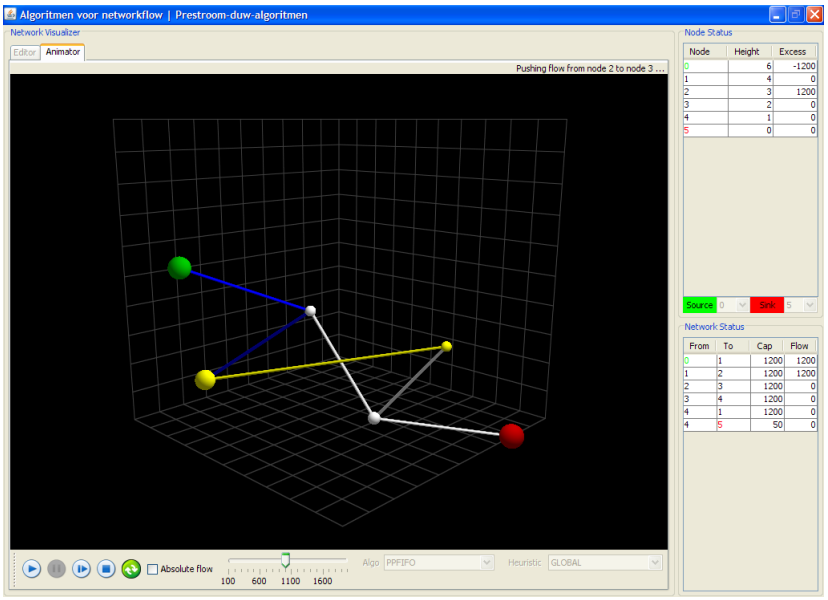
Ter illustratie van de werking en het nut van de applicatie, zowel met betrekking tot onderwijs als onderzoek, lichten we een aantal screenshots van de applicatie kort toe. De lezer wordt aangeraden eerst de handleiding van de applicatie even door te nemen in Appendix B alvorens zelf over te gaan tot het experimenteren met de, in deze sectie, behandelde stroomnetwerken[3] en visuele effecten. Een korte toelichting vindt u steeds als bijschrift bij elke afbeelding.

[2] http://en.wikipedia.org/wiki/Strategy_pattern

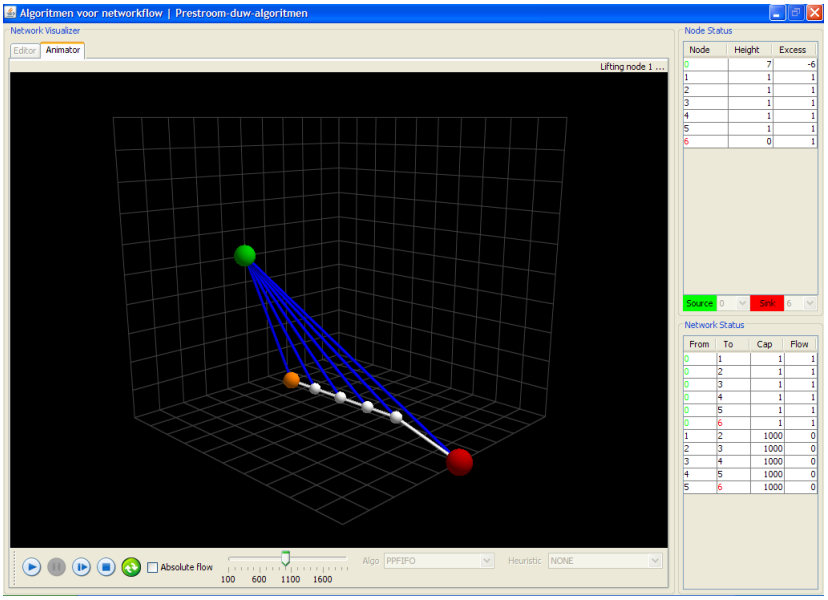
[3] Deze stroomnetwerken kan u, in max-formaat, terugvinden op de bijgeleverde DVD en dit in het mapje voorbeelden_stroomnetwerken_in_max_formaat.



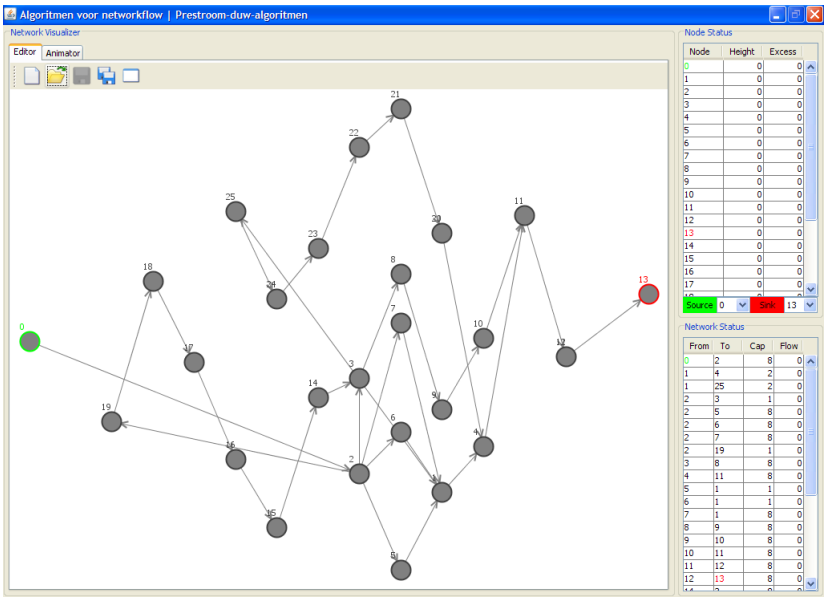
Figuur 10.1: Met de applicatie kunnen we stroomnetwerken van tot ongeveer 50 knopen op een propere manier visualiseren.



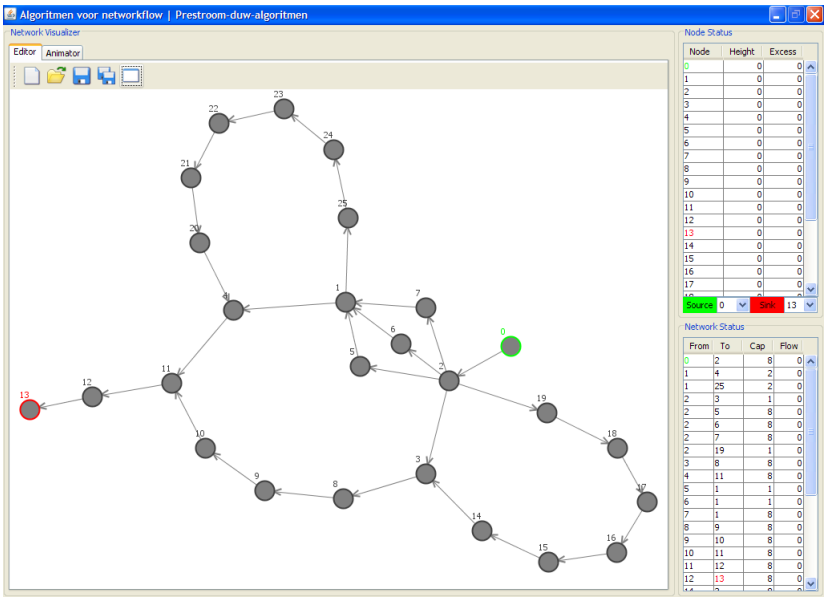
Figuur 10.2: Een aangepaste versie van het stroomnetwerk dat gebruikt werd voor de uitleg van het ping-pong effect. Er bestaat nu immers een boog (w, u) waardoor het effect van een excess cycle zichtbaar wordt met de applicatie.



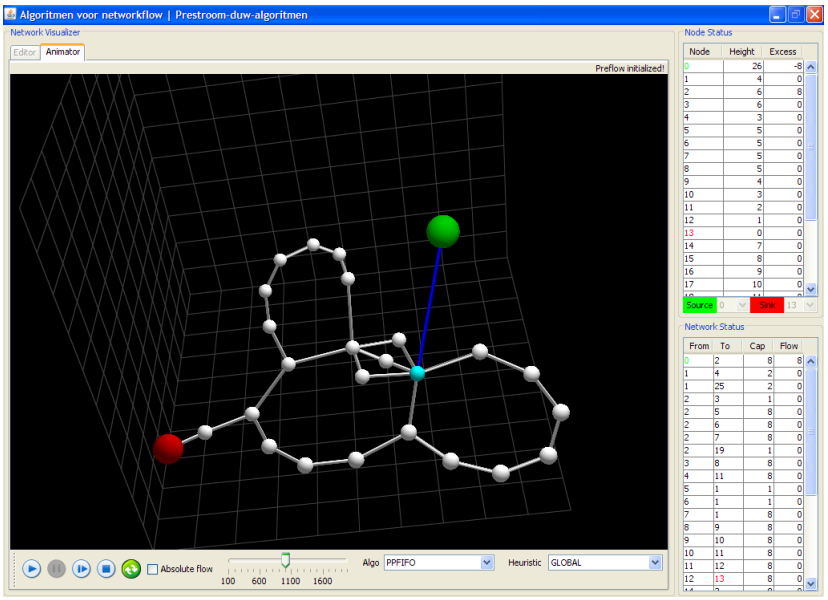
Figuur 10.3: Het stroomnetwerk dat overeenkomt met de constructie die gemaakt werd ter illustratie van de manier waarop FIFO en HHE verschillend omgaan met het oplossen van het maximumstroomprobleem. Het effect dat daarbij besproken werd, was effectief duidelijk zichtbaar.



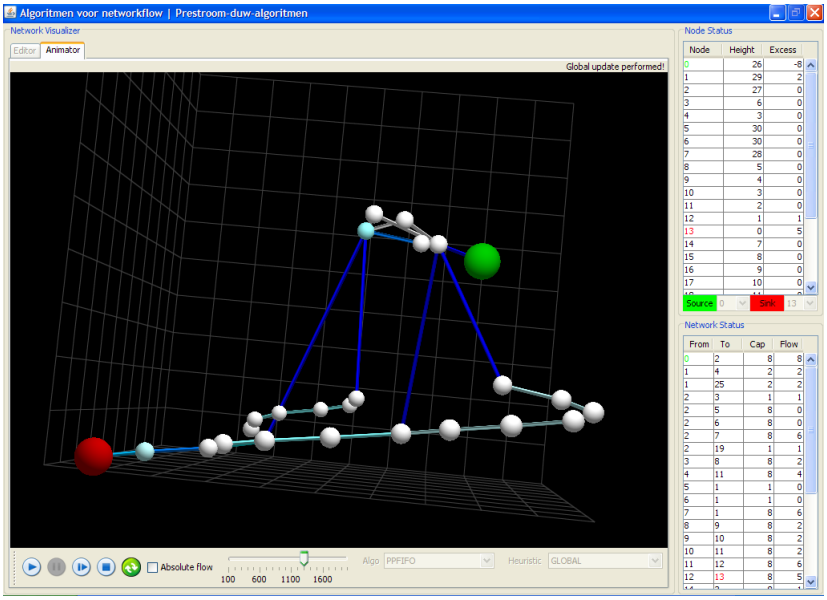
Figuur 10.4: We zien in deze screenshot dat het effect van de pathetische constructie van Cheriyan via een dergelijke tekening in het vlak (van het stroomnetwerktekenalgoritme) niet of nauwelijks duidelijk zal worden. We waren dus genoodzaakt een modus te voorzien die toelaat om toch ook het graaftekenalgoritme zijn tekening te gebruiken bij het opbouwen van de visualisatie van het stroomnetwerk in 3D, zie Figuur 10.5.



Figuur 10.5: Een tekening in het vlak van het stroomnetwerk van Cheriyan en dit volgens het graaftekenalgoritme.



Figuur 10.6: 3D visualisatie van de pathetische constructie van Cheriyan.



Figuur 10.7: Let op de trappenvorming richting bron en afvoer en het duidelijk naar voren komen van de minimale boogsnede voor het stroomnetwerk van Cheriyan.

Hoofdstuk 11

Conclusie

Met behulp van de vergaarde theoretische kennis omtrent prestroom-duw-algoritmen, bouwden we een applicatie die ons toelaat met de bestudeerde prestroom-duw-algoritmen in uitvoering, al dan niet in combinatie met bepaalde heuristieken, te experimenteren en/of hun uitvoering in 3D te visualiseren. Dit laatste bleek vooral handig bij het begrijpen van bepaalde effecten die tijdens de uitvoering van een prestroom-duw-algoritme kunnen plaatsgrijpen. We kunnen dus besluiten dat de applicatie zeker een meerwaarde kan bieden voor educationele- en onderzoeksdoeleinden.

Uit de experimenten hebben we kunnen afleiden dat FIFO het meest robuust is en de combinatie van de globale-verhoog-heuristiek met de kloof-heuristiek een goede keuze blijkt te zijn om deze robuustheid extra kracht bij te zetten. HHE is echter het snelste algoritme. De globale-verhoog-heuristiek is met betrekking tot gebruikerstijd superieur. Het aantal knopen lijkt voor deze heuristiek de beste frequentie te zijn voor het uitvoeren van een globale verhoging. De heuristiek die de maximale stroom schat, lijkt te doen waarvoor deze bedacht is. Verder onderzoek is hier echter nodig aangezien we niet echt uitsluitsel kunnen geven. Het voorzichtig omspringen met datastructuren is de voornaamste les die ik uit het voeren van deze experimenten heb geleerd.

De resultaten uit deze experimenten zouden daarenboven gebruikt kunnen worden om praktische problemen zoals bijvoorbeeld *contourdetectie bij beeldverwerking* te ondersteunen in de zoektocht naar betere oplossingsmethoden. Deze beeldverwerkingstoepassing maakt immers gebruik van maximale-stroom-algoritmen bij het zoeken naar een oplossing. De stroomnetwerken die in deze toepassing gebruikt worden, zijn meestal grote 2D of 3D grids, waarbij de knopen pixels representeren en de capaciteiten van de bogen lager zijn tussen naburige pixels naargelang het intensiteits- of kleurcontrast tussen beide pixels groter wordt [9]. Het komt er met andere woorden op neer dat contouren op die manier overeenkomen met minimale boogsnedes.

Appendices

Bijlage A

Resultaten experimenten

Gebruikerstijd (s)

	(n,m)	FIFO	HHE	ES
GL	(1116, 4800)	2,055	14,270	38,446
	(2058, 9065)	17,749	123,642	378,673
GW	(1024, 4608)	1,915	24,682	36,165
	(2205, 10164)	28,391	379,112	591,446
WLM	(258, 1236)	0,330	0,738	1,988
	(514, 3975)	1,271	3,751	17,501
	(1026, 8069)	4,624	7,875	68,177
	(2050, 22279)	22,171	52,048	628,665

Onverzadigde duwoperaties

	(n,m)	FIFO	HHE	ES
GL	(1116, 4800)	561.460	595.402	630.430
	(2058, 9065)	1.665.837	1.762.370	1.873.189
GW	(1024, 4608)	527.132	554.835	558.043
	(2205, 10164)	2.380.901	2.495.962	2.562.041
WLM	(258, 1236)	30.222	33.377	36.214
	(514, 3975)	149.377	161.797	180.905
	(1026, 8069)	362.598	379.675	428.866
	(2050, 22279)	1.871.796	1.932.970	2.194.434

Verzadigde duwoperaties

	(n,m)	FIFO	HHE	ES
GL	(1116, 4800)	23.410	29.129	24.615
	(2058, 9065)	67.292	74.854	71.661
GW	(1024, 4608)	12.820	25.986	9.904
	(2205, 10164)	52.795	85.366	47.583
WLM	(258, 1236)	6.121	3.597	6.720
	(514, 3975)	29.468	15.214	35.263
	(1026, 8069)	41.472	30.331	41.534
	(2050, 22279)	198.925	117.760	211.060

Verhoogoperaties

	(n,m)	FIFO	HHE	ES
GL	(1116, 4800)	500.438	466.504	526.262
	(2058, 9065)	1.493.969	1.394.572	1.565.031
GW	(1024, 4608)	456.970	368.159	462.075
	(2205, 10164)	2.094.941	1.657.869	2.047.688
WLM	(258, 1236)	29.675	29.202	31.578
	(514, 3975)	147.766	146.971	154.298
	(1026, 8069)	351.499	347.951	367.126
	(2050, 22279)	1.827.766	1.814.386	1.883.008

Figuur A.1: Gemiddelden van de belangrijkste gegevens uit EXP_1.

Gebruikerstijd (s)

	(n,m)	FIFO	$FIFO_S$	$FIFO_{1/2}$
WLM	(258, 1236)	0,330	0,338	0,341
	(514, 3975)	1,271	1,391	1,387
	(1026, 8069)	4,624	3,409	3,388
	(2050, 22279)	22,171	24,393	22,917
	(4098, 65019)	135,854	123,522	123,681

Onverzadigde duwoperaties

	(n,m)	FIFO	$FIFO_S$	$FIFO_{1/2}$
WLM	(258, 1236)	30.222	30.871	29.028
	(514, 3975)	149.377	150.875	145.789
	(1026, 8069)	362.598	364.994	356.967
	(2050, 22279)	1.871.796	1.877.592	1.851.858
	(4098, 65019)	7.524.085	7.535.368	7.465.755

Verzadigde duwoperaties

	(n,m)	FIFO	$FIFO_S$	$FIFO_{1/2}$
WLM	(258, 1236)	6.121	6.645	5.927
	(514, 3975)	29.468	30.580	28.888
	(1026, 8069)	41.472	43.583	40.968
	(2050, 22279)	198.925	203.139	197.227
	(4098, 65019)	816.470	824.885	811.693

Verhoogoperaties

	(n,m)	FIFO	$FIFO_S$	$FIFO_{1/2}$
WLM	(258, 1236)	29.675	30.594	28.687
	(514, 3975)	147.766	149.763	144.765
	(1026, 8069)	351.499	354.964	346.610
	(2050, 22279)	1.827.766	1.835.628	1.810.280
	(4098, 65019)	7.387.287	7.402.110	7.336.399

Figuur A.2: Gemiddelden van de belangrijkste gegevens uit EXP_2.

Gebruikerstijd (s) - frequentie n

	(n,m)	$FIFO_{g\ (freq\ n)}$	$HHE_{g\ (freq\ n)}$
AD	(256, 32640)	0,245	0,480
	(321, 51360)	0,308	0,819
	(403, 81003)	0,501	1,947
GL	(1116, 4800)	0,076	0,046
	(2058, 9065)	0,179	0,095
	(4096, 18368)	0,470	0,231
GW	(1024, 4608)	0,160	0,117
	(2205, 10164)	0,443	0,365
	(3920, 18256)	1,529	0,938
WLM	(258, 1236)	0,011	0,005
	(514, 3975)	0,020	0,017
	(1026, 8069)	0,039	0,027

Gebruikerstijd (s) - frequentie m

	(n,m)	$FIFO_{g\ (freq\ m)}$	$HHE_{g\ (freq\ m)}$
AD	(256, 32640)	1,514	2,041
	(321, 51360)	2,222	3,335
	(403, 81003)	4,633	6,817
GL	(1116, 4800)	0,072	0,127
	(2058, 9065)	0,145	0,196
	(4096, 18368)	0,352	0,705
GW	(1024, 4608)	0,140	0,091
	(2205, 10164)	0,327	0,272
	(3920, 18256)	0,915	0,678
WLM	(258, 1236)	0,024	0,009
	(514, 3975)	0,016	0,070
	(1026, 8069)	0,025	0,097

Figuur A.3: Gemiddelden van de gebruikerstijden voor beide frequenties (n en m) uit EXP_3.

Gebruikerstijd (s)

	(n,m)	$FIFO_g$	HHE_g	$FIFO_k$	HHE_k	$FIFO_{gk}$	HHE_{gk}
AD	(256, 32640)	0,245	0,480	0,221	0,268	0,098	0,211
	(321, 51360)	0,308	0,819	0,297	0,937	0,268	0,764
	(403, 81003)	0,501	1,947	0,574	1,584	0,463	1,359
GL	(1116, 4800)	0,076	0,046	0,298	0,643	0,147	0,484
	(2058, 9065)	0,179	0,095	0,909	2,066	0,405	1,418
	(4096, 18368)	0,470	0,231	3,254	9,348	1,277	5,840
GW	(1024, 4608)	0,160	0,117	0,336	1,462	0,275	0,854
	(2205, 10164)	0,443	0,365	1,215	11,758	0,837	4,141
	(3920, 18256)	1,529	0,938	3,502	76,752	2,353	16,034
WLM	(258, 1236)	0,011	0,005	0,071	0,036	0,016	0,015
	(514, 3975)	0,020	0,017	0,084	0,135	0,033	0,060
	(1026, 8069)	0,039	0,027	0,180	0,280	0,065	0,142

Figuur A.4: Gemiddelden van de gebruikerstijden uit EXP_4.

Gebruikerstijd (s)

	(n,m)	FIFO	$FIFO_g$	$FIFO_k$	$FIFO_{gk}$
GL	(1116, 4800)	2,055	0,076	0,298	0,147
	(2058, 9065)	17,749	0,179	0,909	0,405
	(4096, 18368)	97,313	0,470	3,254	1,277
GW	(1024, 4608)	1,915	0,160	0,336	0,275
	(2205, 10164)	28,391	0,443	1,215	0,837
	(3920, 18256)	114,809	1,529	3,502	2,353
WLM	(258, 1236)	0,330	0,011	0,071	0,016
	(514, 3975)	1,271	0,020	0,084	0,033
	(1026, 8069)	4,624	0,039	0,180	0,065

Figuur A.5: Gemiddelde gebruikerstijden van FIFO, $FIFO_g$, $FIFO_k$ en $FIFO_{gk}$ uit EXP_5.

Stabiliteitspercentages

	(n,m)	FIFO	$FIFO_g$	$FIFO_k$	$FIFO_{gk}$
GL	(4096, 18368)	0,178	0,022	0,078	0,339
GW	(3920, 18256)	0,049	0,184	0,033	0,108
WLM	(1026, 8069)	0,637	0,173	0,368	0,299
	Gemiddelde	0,288	0,126	0,160	0,249

Figuur A.6: Stabiliteiten van FIFO, $FIFO_g$, $FIFO_k$ en $FIFO_{gk}$ uitgedrukt in percentages uit EXP_5.

Bijlage B

Handleiding applicatie

B.1 Installatie en uitvoering

De installatie van de applicatie, die op de bijgeleverde DVD kan teruggevonden worden, houdt eigenlijk enkel de installatie van JOGL in, waarna er vervolgens kan overgegaan worden tot het uitvoeren van de applicatie met Java. De applicatie werd enkel getest onder Windows en met JRE (build 1.6.0_07-b06). Het stappenplan om JOGL, tevens op de DVD aanwezig onder het mapje jogl_win\, kenbaar te maken aan Java onder Windows, gaat als volgt:

1. Navigeer in Windows naar 'Omgevingsvariabelen', terug te vinden onder 'Systeem' in het 'Configuratiescherm'.
2. Plaats het volledige pad naar jogl_win\lib in de systeemvariabele PATH
3. Plaats het volledige pad naar jogl_win\lib\jogl.jar en jogl_win\lib\gluegen-rt.jar in de CLASSPATH gebruikersvariabele. Voeg tevens '.' toe aan CLASSPATH indien dit nog niet gebeurd is.

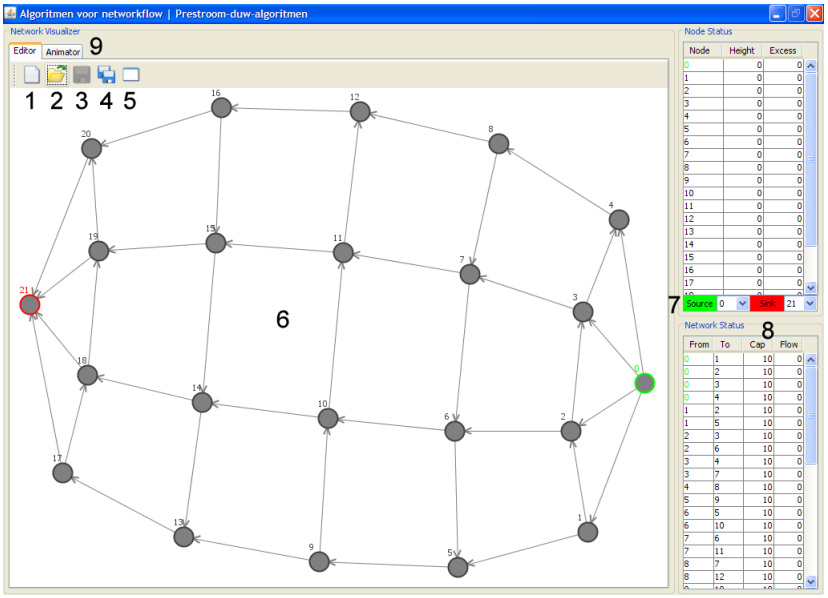
Om de applicatie nu zelf uit te voeren, dient u met de commandline naar MaxFlowProbFramework\build\classes te navigeren en vervolgens het commando 'java GUIMain' uit te voeren.

B.2 Functionaliteiten

De applicatie bestaat uit twee modules: een editor om stroomnetwerken te construeren en een animator om de uitvoering van de geïmplementeerde prestroom-duw-algoritmen en heuristieken, op de geconstrueerde stroomnetwerken, te visualiseren. Beide modules kunnen niet tegelijkertijd worden uitgevoerd en zijn daarom, in de user interface, ondergebracht in twee aparte tabbladen. We geven vervolgens, per module, een overzicht van zijn functionaliteiten en hoe deze kunnen gebruikt worden. Deze functionaliteiten

worden genummerd overlopen volgens de nummering op de bijgevoegde figuren.

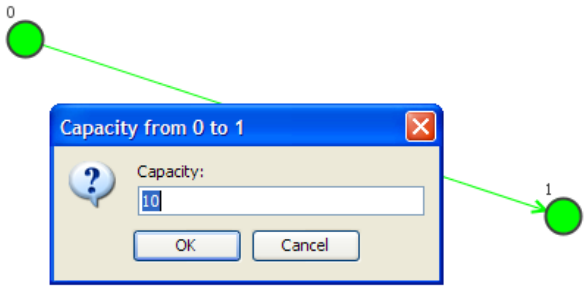
B.2.1 De editor



Figuur B.1: De editor.

1. Maak een nieuw bestand aan.
2. Openen van een nieuw bestand.
3. Bewaren van het huidig geopend bestand.
4. Bewaren van het huidig geopend bestand, mogelijk onder een andere naam.
5. Veranderen van het zicht op het stroomnetwerk. Hiermee bedoelen we het zicht bekomen door het graaftekenalgoritme of het stroomnetwerktekenalgoritme. Deze mogelijkheid werd voorzien omdat effecten bekomen door constructies soms makkelijker te zien zijn wanneer we het stroomnetwerk visualiseren vanuit het standpunt van de discipline graaf tekenen.
6. In de editor kunnen stroomnetwerken geconstrueerd worden met behulp van de volgende acties:

- Bij een muisklik in de canvas wordt een nieuwe knoop aangemaakt.
- Bij het rechtsklikken, op een knoop in de canvas, verschijnt er een menu dat de mogelijkheid biedt om deze knoop te verwijderen. De bogen waarvan deze knoop kop of staart is, worden natuurlijk ook verwijderd.
- Het verplaatsen van een knoop door op deze knoop te klikken, vervolgens de muisknop ingedrukt te houden, met de muis te slepen naar een nieuwe positie en de muisknop los te laten. De nieuwe positie van de knoop en zijn bogen zal, tijdens het slepen, ten aller tijde zichtbaar zijn.
- Indien we bij het verplaatsen van een knoop in een knoop terecht komen waarmee deze knoop nog niet verbonden is, wordt er een dialoogvenster getoond zoals weergegeven in Figuur B.2. Bij ingave van een capaciteit groter dan 0 wordt de boog effectief toegevoegd. Indien we terecht komen in een knoop waarmee deze knoop reeds wel verbonden is, zal de knoop rood oplichten. Merk op dat bogen in beide richtingen kunnen toegevoegd worden, hierbij aangegeven door pijlen in beide richtingen op de boog.



Figuur B.2: Toevoegen van een boog.

De bron en de afvoer worden in een groene en resp. rode kleur aangeduid en de knopen zijn voorzien van nummers zodanig dat ze makkelijk te onderscheiden zijn tijdens het tekenen.

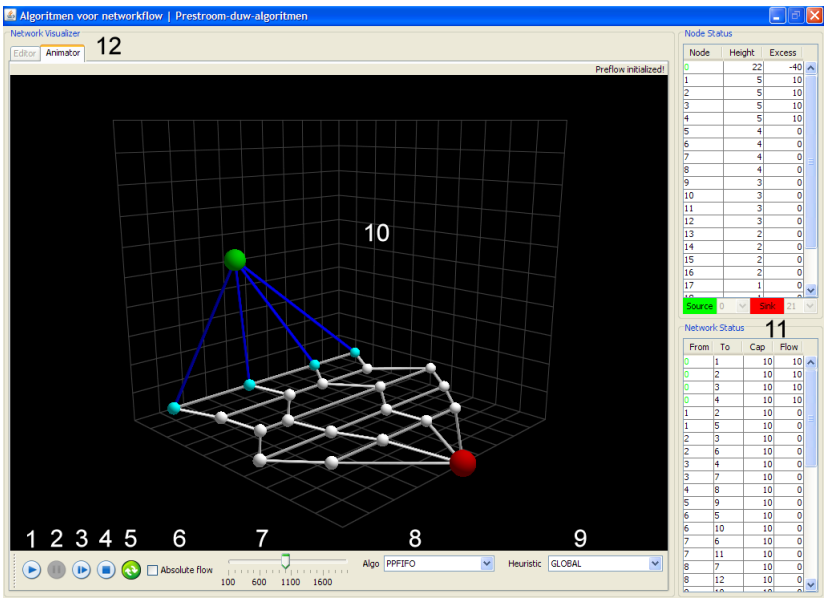
7. Instellen van de bron en afvoer. We merken op dat de constructie over een bron en afvoer moet beschikken alvorens het stroomnetwerk zijn visualisatie berekend wordt en de animator beschikbaar wordt.

8. Enkel in de editor hebben we de mogelijkheid om zelf de netwerkstatus te wijzigen. We beschikken hier meer bepaald over de mogelijkheid tot

het aanpassen van de capaciteiten van de bogen. Het verwijderen van een boog kan bekomen worden door het invoeren van een capaciteit gelijk aan nul.

9. Met behulp van deze tab kan er overgeschakeld worden naar de animator.

B.2.2 De animator



Figuur B.3: De animator.

1. Het automatisch laten afspelen van de visualisatie.
2. Het pauzeren van de visualisatie.
3. Het stapsgewijs doorlopen van de visualisatie.
4. Het voortijdig beëindigen van de visualisatie. De beginwaarden van de visualisatie worden terug aangenomen.
5. Het terugkeren naar de beginpositie van de camera. Het is immers mogelijk om tijdens het verloop van de visualisatie doorheen de 3D ruimte te bewegen op de volgende manieren:
    - Omhoog, omlaag, links en rechts met behulp van de pijltjestoetsen.

- Draaien volgens de verschillende assen met behulp van een sleepactie met de muis.
- In- of uitzoomen met behulp van een scrollbeweging met de muis.

6. Indien deze optie is aangevinkt, zal er in het stroomnetwerk een absolute stroom weergegeven worden. We hebben deze optie voorzien om de verhoudingen tussen boogbezettingen eventueel te verduidelijken.

7. Het regelen van de snelheid waarmee de animatie, bij het automatisch afspelen ervan, wordt uitgevoerd.

8. De mogelijkheid om een ander prestroom-duw-algoritme te kiezen, ter visualisatie.

9. De mogelijkheid om een andere heuristiek te kiezen, ter visualisatie.

10. Met betrekking tot de animator werden de volgende gedragingen en kleurcodes vastgelegd:

    - De bron en afvoer hebben, net als in de editor, een groene en rode kleur.
    - Een operatie wordt aangekondigd met het opzwellen en/of kleuren van de betrokken knoop/knopen. In het geval van een verhoogoperatie maken we gebruik van een oranje tint. Het spreekt natuurlijk voor zich dat de betrokken knopen hierdoor een niveau zullen stijgen in de 3D ruimte. In het geval van een duwoperatie maken we gebruik van een gele tint. Nadat de operatie werd uitgevoerd, zal de zwelling van de knoop verdwijnen. De hoeveelheid stroom over de bogen van het stroomnetwerk wordt aangegeven door een blauwe kleurtint die meer uitgesproken is wanneer de bezetting van een boog het grootst is. Bij het aangeven van de reststroom, op de bol die overeenkomt met de knoop, wordt deze vergeleken ten opzichte van de totale capaciteit van de inkomende bogen van de desbetreffende knoop.

11. Via deze tabellen kan de gebruiker van de applicatie ten aller tijde de status van het hele stroomnetwerk gedetailleerd volgen.

12. De tab om terug te keren naar de editor zal enkel actief zijn indien de animator niet bezig is met een uitvoering.

Bibliografie

[1] Kirchhoff's current law. http://www.physics.uoguelph.ca/tutorials/ohm/Q.ohm.KCL.html.

[2] Maximum flow. http://www.cs.uu.nl/docs/vakken/an/an-maxflow.ppt.

[3] Network flows and graphs. http://www.ieor.berkeley.edu/~ieor266/Lecture19-20.pdf.

[4] *Network flows*, volume 3 of *The Open University Technology/Mathematics, A Third Level Interfaculty Course, Graphs, Networks and Design.* 1981.

[5] *Goldberg's algorithm for maximum flow in perspective: a computational study*, volume 12 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science.* 1993.

[6] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: theory, algorithms, and applications*, chapter 1, 2, 6, 7. Prentice Hall, 1993.

[7] Eva Tardos Andrew V. Goldberg and Robert E. Tarjan. Network flow algorithms. http://www.cs.cornell.edu/People/eva/Network.Flow.Algorithms.pdf.

[8] T. Badics and E. Boros. Implementing a maximum flow algorithm: Experiments with dynamic trees.

[9] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. 2002.

[10] J. Cheriyan and S. N. Maheshwari. Analysis of preflow push algorithms for maximum network flow.

[11] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms, second edition*, chapter 26. The MIT Press, 2001.

[12] Gene Davis. *Learning Java Bindings for OpenGL (JOGL).* AuthorHouse, 2004.

[13] Roberto Tamassia Giuseppe Di Battista, Peter Eades and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualisation of Graphs*, chapter 1. Prentice Hall, 1999.

[14] Michiel Grauwels. Visualisatie krachtgestuurd graaf tekenen. Eindwerk voorgedragen tot het behalen van de graad van bachelor in de informatica/ict/kennistechnologie, 2007.

[15] Princeton University Kevin Wayne. Edmonds-karp heuristics. `http://www.math.wm.edu/~rrkinc/maxflow.pdf`.

[16] Ajay K. Mishra Ravindra K. Ahuja, Murali Kodialam and James B. Orlin. Computational investigations of maximum flow algorithms.

[17] Alexander Schrijver. Flows in railway optimization. 2008.

[18] Daniel D. Sleator and Robert E. Tarjan. A data structure for dynamic trees.

[19] Uri Zwick. Max-flow algorithms and applications. `http://compgeom.cs.uiuc.edu/~jeffe/teaching/algorithms/notes/16-maxflowalgs.pdf`.

[20] Uri Zwick. The smallest networks on which the ford-fulkerson maximum flow procedure may fail to terminate. `http://www.cs.toronto.edu/~bor/375s06/flow_example.ps`.