De mogelijkheden en beperkingen van een computermodel dat het optimale ontwerp bepaalt van een uitkragende, vlakke vakwerkligger.

T.J.M. Wubs

TU Delft Delft University of Technology

Challenge the future

Bachelor Eindproject Civiele Techniek, TU Delft.

Eindrapport
November 2016

Onderzoek naar de mogelijkheden en beperkingen van een computermodel dat zelfstandig het optimale ontwerp van een uitkragende, vlakke vakwerkligger bepaalt.

Door T.J.M. Wubs

Studienummer: 4088549
Begeleiders: Dr. ir. R. Abspoel
Ir. P.A. de Vries
Dr. ir. P.C.J. Hoogenboom

(Bron foto voorpagina: Architizer 2011)

Voorwoord

Dit eindrapport beschrijft het onderzoek naar de mogelijkheden en beperkingen van een computermodel dat zelfstandig het optimale ontwerp bepaalt voor een uitkragende, vlakke vakwerkligger als onderdeel van de overkapping van een tribune. Het onderzoek is uitgevoerd ter afsluiting van de bachelor opleiding Civiele Techniek aan de Technische Universiteit Delft. Het betreft een individueel onderzoek met een tijdsduur van acht weken.

Gedurende deze periode hebben dr. ir. R. Abspoel en ir. P.A. de Vries de rol van eerste begeleider vervuld en dr. ir. P.C.J. Hoogenboom heeft zijn hulp als tweede begeleider aangeboden. Ik wil mijn drie begeleiders hartelijk bedanken voor de energie en tijd die zij geïnvesteerd hebben. Het commentaar en de adviezen die ik van hen heb ontvangen, hebben het mogelijk gemaakt dit resultaat te behalen. Ik heb met veel plezier gewerkt aan mijn onderzoek en het bijbehorende eindrapport.

Indien u vragen of opmerkingen heeft, kunt u contact met mij opnemen via onderstaand emailadres.

Tirsa Wubs
twubs@student.tudelft.nl

Delft, november 2016.

Samenvatting

Er is voor bachelor eindprojecten veel onderzoek gedaan naar vakwerkconstructies. Hierbij is regelmatig op een handmatige manier bepaald wat het optimale ontwerp is voor een specifieke constructievorm. Er zijn echter geen bachelor eindprojecten te vinden waarbij, voor de optimalisatie van de vakwerkconstructie, gebruik gemaakt wordt van optimalisatie software om op een gestructureerde en automatische manier het optimale ontwerp te bepalen. Het is dan ook de vraag hoe optimaal het *optimale* ontwerp is als deze handmatig is bepaald.

Het ontbreken van onderzoek naar de toepassing van optimalisatie software vormt de aanleiding van dit onderzoek. Dit onderzoek heeft als doel het bedenken, bouwen en testen van een computermodel dat, met behulp van bestaande optimalisatie software, zelfstandig het optimale ontwerp van een vakwerkligger bepaalt. De mogelijkheden en beperkingen van het computermodel zijn onderzocht aan de hand van de ontwerpvraag van een uitkragende, vlakke vakwerkligger, die onderdeel uitmaakt van de overkapping van een tribune.

Het computermodel dat voor dit onderzoek ontwikkeld is, bestaat uit een viertal modules en een optimalisatie algoritme. Drie modules zijn voor dit onderzoek ontwikkeld. Dit zijn de ontwerpmodule, de module ter bepaling van de belastingen en de FCs en de toetsingsmodule. Voor de vierde module, de rekenmodule, is gebruik gemaakt van het programma 3D Truss.

In het computermodel is een optimalisatie algoritme uit de bekende Python optimalisatie bibliotheek *SciPy*, geïmplementeerd. Dit algoritme werkt volgens de *continuous gradient-based optimization method*. Met oog op de beperkte tijdsduur van het onderzoek is dit algoritme gekozen. Uit de vergelijking van een viertal optimalisatie aanpakken is namelijk naar voren gekomen dat continue optimalisatie algoritmen in de regel sneller zijn in het bepalen van het optimum dan discrete. Ook is aangetoond dat bij het toegepaste algoritme, het optimalisatieproces zeer goed te sturen is.

Het gekozen optimalisatie algoritme kan alleen continue ontwerpvariabelen verwerken. Het optimalisatievraagstuk van dit onderzoek is echter een probleem met zowel continue als discrete ontwerpvariabelen. Het optimalisatievraagstuk is daarom opgesplitst in een continu hoofdprobleem, waarvoor het optimalisatie algoritme wordt toegepast, en een discreet subprobleem. Voor het discrete subprobleem wordt de brute force approach toegepast.

Het computermodel doet er nog geen zes uur over om het optimale ontwerp te bepalen. Hierbij zijn voor alle 102 combinaties van de discrete ontwerpvariabelen, de continue variabelen door het optimalisatie algoritme geoptimaliseerd. Het optimale ontwerp voor de beschouwde vakwerkligger, dat door het computermodel is bepaald, is een Warren truss met systeemverbindingen opgebouwd uit 10 velden. De belastingcombinatie die de opwaartse windbelasting behandelt, is de maatgevende fundamentele combinatie voor het ontwerp van de vakwerkligger. Daarbij is de eis met betrekking tot de knikstabiliteit bepalend voor de dimensionering van de randstaven. De dimensionering van de wandstaven wordt bepaald door de minimale wanddikteverhouding en door de maximale d/t-verhouding.

De resultaten van de optimalisatie tonen aan dat er voor het aantal velden van de beschouwde vakwerkligger het beste gekozen kan worden uit de range 7 tot en met 11. Binnen deze range zijn de kosten van de verschillende ontwerp gemiddeld 22 % hoger dan de kosten van het optimale ontwerp. Buiten de range is dit verschil minimaal 25 % en gemiddeld 70 %.

Er is sprake van drie belangrijke beperkingen van het ontwikkelde computermodel. De eerste beperking komt voort uit het feit dat er geen universele wijze bestaat voor de kostenbepaling van de staven en de verbindingen. Het gedeelte van het computermodel dat deze kosten berekent, moet dan ook bij ieder gebruik gecontroleerd en eventueel aangepast worden.

De tweede beperking heeft betrekking op de beperkte ontwerpvrijheid van het computermodel. Het model kan voor het optimale ontwerp alleen kiezen uit de opties, bijvoorbeeld voor de configuratie, die zijn geprogrammeerd.

De derde beperking is de mogelijke remming van creativiteit en innovatie, die het computermodel kan veroorzaken. Het ontwikkelen van een nieuw ontwerp, vraagt namelijk om extra handelingen, zoals uitbreidingen van het model, met het risico op een langer ontwerpproces en hogere kosten.

Tegenover deze derde beperking staat een mogelijkheid van het computermodel. Het computermodel kan in relatief korte tijd zeer veel ontwerpen doorrekenen, waardoor het optimalisatieproces wordt versneld. De tijd die hiermee bespaard wordt, kan besteed worden aan het onderzoeken van nieuwe mogelijkheden. In dit geval worden creativiteit en innovatie dus juist gestimuleerd. Er kan niet zonder meer vastgesteld worden of de beperking of de mogelijkheid overheerst. Dit zal per situatie verschillen.

Andere mogelijkheden die uit dit onderzoek naar voren zijn gekomen, leiden allemaal tot kostenbesparing. Door het gebruik van het computermodel kan human error voorkomen worden. Minder fouten betekent lagere kosten. De informatie die het computermodel levert, maakt het mogelijk om betere ontwerpen te maken. Deze kwaliteitsverhoging kan ook opgevat worden als een besparing van kosten.

Inhoudsopgave

1 Inleiding

Uit het bestuderen van onderzoeken die uitgevoerd zijn voor bachelor eindprojecten blijkt dat optimalisatie van onder andere vakwerkconstructies een veelvoorkomend onderwerp is. Het is echter opvallend dat bij alle onderzoeken die bestudeerd zijn, de optimalisatie uitgevoerd wordt als parameterstudie. Een aantal ontwerpen ofwel combinaties van variabelen worden met elkaar vergeleken om uiteindelijk het beste ontwerp te bepalen. In geen van de bestudeerde onderzoeken wordt gebruik gemaakt van bestaande optimalisatie software om op een gestructureerde en automatische manier het optimum te bepalen. Dit roept de volgende vraag op: *Is het gewenst om een computermodel te ontwikkelen dat in een relatief korte tijd het optimale ontwerp van een vakwerkconstructie kan bepalen?*

Een ingenieur doorloopt bij het ontwerpen, al dan niet impliciet, een optimalisatieproces aan de hand van rekenregels, ervaring, intuïtie etc.. Dit wordt *experience-based optimization* genoemd (Parkinson et al., 2013 book p. 1). Voor problemen met een beperkt aantal variabelen en eisen, is dit een geschikte aanpak. Zodra het aantal mogelijke oplossingen echter oploopt, is *experience-based optimization* niet meer efficiënt. Zeker in het geval dat er allerlei, wellicht tegenstrijdige, eisen gesteld worden, is een alternatieve aanpak noodzakelijk. De afhankelijkheden van dit soort problemen zijn vaak zeer complex en het doorrekenen van mogelijke oplossingen kost veel tijd (Parkinson et al., 2013 book p. 1). Indien er sprake is van zo'n complex probleem is het dan ook gewenst om de optimalisatie uit te laten voeren door een computermodel.

Verschillende onderzoeken naar de optimalisatie van vakwerkconstructies benadrukken de complexiteit van problemen, waarbij naast de optimale staafafmetingen ook gezocht wordt naar de optimale configuratie, zo schrijft Stolpe in zijn review van *Truss optimization* (Stolpe, 2015, p.350). Hier staat tegenover dat de reductie van de kosten- danwel het gewicht bij optimalisatie van de configuratie doorgaans 40 tot 100 % groter is dan de reductie die bereikt wordt indien enkel de staafafmetingen geoptimaliseerd worden (Chamoret et al., 2008, p.322).

Op grond hiervan lijkt het inderdaad gewenst om een computermodel te ontwikkelen voor de optimalisatie van vakwerkconstructies. Dat de toepassingsmogelijkheden van een dergelijk computermodel niet eerder onderzocht zijn voor een bachelor eindproject, vormt de aanleiding van dit onderzoek.

Bij dit onderzoek worden de mogelijkheden en beperkingen van een computermodel dat voor een vakwerkligger het optimale ontwerp bepaalt, onderzocht. Dit onderzoek beperkt zich tot de optimalisatie van het ontwerp van een uitkragende, vlakke vakwerkligger. Hierbij wordt zowel gezocht naar de optimale afmetingen van de staven als naar de optimale configuratie van de vakwerkligger. Het ontwerp wordt geoptimaliseerd naar kosten.

1.1 Doelstelling

Dit onderzoek heeft als doel het bedenken, bouwen en testen van een computermodel dat zelfstandig de optimale configuratie en dimensionering van een vakwerkligger bepaalt met behulp van bestaande optimalisatie software. Hierbij wordt het ontwerp geoptimaliseerd naar kosten. Het is niet de bedoeling dat er een volledig werkend product wordt afgeleverd. De nadruk van het onderzoek ligt op het verkrijgen van inzicht in de mogelijkheden en beperkingen van een denkbeeldig groter computermodel dat al het technische werk van een ingenieur zou kunnen uitvoeren. Voor dit onderzoek wordt een model ontwikkeld met een modulaire opbouw, zodat het model gemakkelijk uitgebreid kan worden.

<u>De Onderzoeksvraag & Ontwerpvraag</u>

De onderzoeksvraag die in dit rapport beantwoord wordt, is:
Wat zijn de mogelijkheden en beperkingen van een computermodel dat zelfstandig bepaalt wat, economisch gezien, het optimale ontwerp is van een uitkragende, vlakke vakwerkligger, die onderdeel uitmaakt van de overkapping van een tribune?

Om deze vraag te kunnen beantwoorden wordt er een computermodel ontwikkeld voor de optimalisatie van een specifieke ontwerpvraag. De ontwerpvraag die bij dit onderzoek beschouwd wordt, is:
Wat is het optimale ontwerp voor een uitkragende, vlakke vakwerkligger, die onderdeel uitmaakt van de overkapping van een tribune?
In dit rapport wordt deze ontwerpvraag ook wel aangeduid met het optimalisatievraagstuk van dit onderzoek.

<u>Deelvragen</u>

Naast bovenstaande onderzoeksvraag is er een tweetal deelvragen geformuleerd. De informatie die verkregen wordt door deze deelvragen te beantwoorden, is nodig om antwoord te kunnen geven op de onderzoeksvraag. De deelvragen zijn als volgt geformuleerd:

1. Welke modules moeten er voor het computermodel ontwikkeld worden om de stappen die een ingenieur doorloopt bij het ontwerpen van een vakwerkligger te automatiseren?
2. Welke optimalisatie aanpak is het meest geschikt voor het optimalisatievraagstuk van de vakwerkligger, zoals deze voor dit onderzoek beschouwd wordt?

1.2 Afbakening en programmeeraanpak

Dit onderzoek beschouwt de optimalisatie van een uitkragende, vlakke vakwerkligger, die onderdeel uitmaakt van de overkapping van een tribune. De focus van dit onderzoek ligt op het proces dat het computermodel doorloopt. Om dit proces en de mogelijkheden en beperkingen ervan in kaart te brengen, is het overbodig om het hele ontwerpgebied te beschouwen. Het ontwerpgebied wordt op drie manieren afgebakend. Ten eerste wordt het ontwerpgebied afgebakend door de keuze alleen vlakke vakwerkliggers te beschouwen. Deze keuze is gerechtvaardigd, omdat de wijze waarop het proces verloopt voor vlakke en ruimtelijke vakwerkliggers hetzelfde is. Voor dit onderzoek is het dan ook niet noodzakelijk om ruimtelijke vakwerkliggers te beschouwen. Het computermodel wordt wel zo ontwikkeld dat de stap van vlakke naar ruimtelijke vakwerkliggers voor een eventueel vervolgproject gemakkelijk gemaakt kan worden.

Ten tweede worden er één soort profiel en één staalsoort beschouwd. Voor andere profielen en materiaalsoorten verloopt het optimalisatieproces op dezelfde manier. Bij dit onderzoek worden warmgewalst buisprofielen van staalsoort S355 toegepast. Deze staalsoort is gekozen, omdat verwacht wordt dat de vervormingen voor een vakwerk niet maatgevend zullen zijn voor de dimensionering van de staven. In dat geval wordt de dimensionering bepaald op basis van sterkte. Een hogere staalsoort is dan een logische keuze.

Tot slot wordt het ontwerpgebied verder afgebakend door de variabelen die een rol spelen bij het ontwerp van de vakwerkligger te begrenzen. De variabelen en de bijbehorende begrenzingen worden vastgesteld naarmate het onderzoek vordert.

Behalve het ontwerpgebied wordt ook het onderzoeksgebied beperkt. Dit wordt gedaan door een viertal aannames op te stellen. Ten eerste wordt aangenomen dat de stabiliteit van de gehele dakconstructie verzekerd is.

Ten tweede wordt aangenomen dat de krachten die via de vakwerkligger aan de steunpunten worden afgedragen, opgenomen kunnen worden door de overige constructie-onderdelen.

De derde aanname, die bij het ontwikkelen van het computermodel voor de optimalisatie van de vakwerkligger gehanteerd wordt, is dat het vlakke vakwerk in alle knooppunten zijwaarts ondersteund wordt. Deze ondersteuning is nodig om instabiliteit van de vakwerkligger te voorkomen. Er wordt aangenomen dat de zijwaartse ondersteuning van de bovenrandstaaf wordt geregeld door de gordingen van de dakbedekking die aan de knooppunten bevestigd zijn. Hoe de zijwaartse ondersteuning van de onderrandstaaf precies geregeld wordt, wordt gezien de beperkte tijd van dit onderzoek buiten beschouwing gelaten.

De vierde aanname waaronder het computermodel is ontwikkeld, is dat de vakwerkligger zich gedraagt als zuiver vakwerk. Er wordt aangenomen dat de belastingen aangrijpen in de knopen van het vakwerk en dat de staven zodoende alleen door normaalkrachten belast worden. De eventuele excentriciteitsmomenten, die in de knopen kunnen optreden, en de secundaire momenten worden buiten beschouwing gelaten.

Er zijn diverse manieren waarop het optimalisatievraagstuk van de vakwerkligger aangepakt kan worden. Binnen de bestaande optimalisatie-software wordt onderscheid gemaakt tussen continue en discrete optimalisatie methoden. Welke methode het meest geschikt is, is afhankelijk van het type probleem. Optimalisatievraagstukken worden veelal ingedeeld naar het type ontwerpvariabelen van het probleem. Er wordt onderscheid gemaakt tussen drie typen problemen: *continuous problems, discrete problems* en *mixed continuous-discrete problem* (Straathof et al., 2008 p. 12). Deze onderverdeling is gebaseerd op het type ontwerpvariabelen van het probleem. Een variabele die continu is, kan binnen bepaalde grenzen alle tussenliggende waarden aannemen. De diameter van een staaf is bijvoorbeeld een continue variabele. Discrete variabelen kunnen daarentegen slechts een bepaald aantal waarden of vormen aannemen. Het aantal velden waaruit een vakwerkligger bestaat is een voorbeeld van een discrete variabele.

Gezien de beperkte tijd waarin het onderzoek uitgevoerd moet worden, wordt slechts één optimalisatie aanpak verwerkt in het computermodel. Om te kunnen bepalen welke optimalisatie aanpak hiervoor het meest geschikt is, worden eerst de kenmerken van enkele bekende optimalisatie aanpakken beschreven. Hierna worden ook de kenmerken van het optimalisatievraagstuk dat hoort bij het ontwerp van de vakwerkligger, toegelicht. Op basis van deze kenmerken wordt vervolgens de meest geschikte optimalisatie aanpak gekozen.

<u>Programmeeraanpak</u>

Het optimalisatievraagstuk waar het computermodel voor ontwikkeld wordt, is een complex vraagstuk met veel verschillende variabelen. Bij het ontwikkelen van een computermodel voor zo'n vraagstuk is het van belang dat er op een gestructureerde manier te werk wordt gegaan. Bij dit onderzoek wordt er gewerkt van *klein* naar *groot.* De uitkragende, vlakke vakwerkligger, waar het computermodel uiteindelijk zelfstandig het optimale ontwerp voor dient te bepalen, wordt eerst versimpeld door het aantal variabelen te verkleinen. Zo wordt er eerst een model gemaakt voor een eenvoudig vakwerk. Er wordt niet gekeken naar een hele ligger, maar slechts naar enkele velden. Ook heeft het model in eerste instantie nog veel input nodig van de gebruiker. De ingenieur dient in dit stadium nog veel beslissingen zelf te maken. Het model wordt echter steeds verder uitgebreid en

steeds meer van deze beslissingen worden overgenomen van de ingenieur. Een voorbeeld is het aantal velden waar de vakwerkligger uit is opgebouwd. In het beginstadium wordt een willekeurige hoeveelheid velden gekozen door de ingenieur. Het is echter de bedoeling dat het model uiteindelijk zelf bepaalt hoeveel velden er het beste toegepast kunnen worden.

De input die het model altijd nodig heeft, zijn de specifieke gegevens die horen bij de ontwerpvraag. Deze gegevens worden de ontwerpgegevens genoemd.

1.3 Structuur van het rapport

Na de inleiding wordt in hoofdstuk 2 de informatie besproken die verkregen is door een uitgebreide literatuurstudie. Hier wordt toegelicht welke informatie uit de bestudeerde bachelor eindrapport bruikbaar is voor dit onderzoek. Daarnaast wordt aan de hand van een ingenieursaanpak voor het ontwerpen van een vakwerkligger beschreven uit welke modules het computermodel wordt opgebouwd. Hierna worden achtereenvolgens de onderwerpen vakwerken, belastingen, toetsingsregels en optimalisatie aanpakken behandeld.

In hoofdstuk 3 worden de modules die ontwikkeld moeten worden, één voor één behandeld. Hierbij wordt beschreven hoe de informatie uit hoofdstuk 2 bij dit onderzoek wordt toegepast.

Hoofdstuk 4 gaat over de wijze waarop het ontwerp van de vakwerkligger geoptimaliseerd wordt. In de eerste paragraaf wordt beschreven hoe de kosten waarnaar het ontwerp geoptimaliseerd wordt, berekend worden. Vervolgens wordt in de tweede paragraaf de meest geschikte optimalisatie aanpak voor dit onderzoek gekozen. In de derde en laatste paragraaf wordt beschreven hoe de gekozen optimalisatie aanpak wordt toegepast. Hierbij wordt toegelicht welke bounds en startwaarden bij de optimalisatie gehanteerd worden en hoe deze zijn opgesteld.

De opbouw van het computermodel wordt beschreven in hoofdstuk 5. Aan de hand van een drietal flow diagrams wordt toegelicht welke stappen het computermodel doorloopt en welke keuzes hierbij gemaakt moeten te worden. Ook wordt de validatie van het model besproken.

In hoofdstuk 6 worden de resultaten van het optimalisatievraagstuk van de vakwerkligger gepresenteerd. Er wordt onderzocht hoe de verhouding van de staafkosten en de verbindingskosten het verloop van de totale kosten beïnvloedt. Ook wordt gecontroleerd of het optimale ontwerp dat door het computermodel bepaald is, daadwerkelijk optimaal is en of het voldoet aan alle constraints. Hierbij wordt tevens aangetoond welke belastingcombinatie en welke constraints maatgevend zijn voor de dimensionering van de vakwerkligger.

In hoofdstuk 7 worden de conclusies die op basis van dit onderzoek getrokken kunnen worden, besproken. In paragraaf 1 wordt de ontwerpvraag beantwoord. Vervolgens worden in paragraaf 2 de deelvragen en de onderzoeksvraag beantwoord. Hierbij wordt het antwoord op de onderzoeksvraag opgesplitst in een beschrijving van de mogelijkheden en een beschrijving van de beperkingen.

In hoofdstuk 8 worden aanbevelingen gedaan ter verbetering van het computermodel. Het rapport eindigt met de bronvermelding in hoofdstuk 9 en de bijlagen in hoofdstuk 10.

2 Literatuurstudie

In dit hoofdstuk wordt de informatie beschreven, die verkregen is door een uitgebreide literatuurstudie. In paragraaf 1 wordt beschreven welke informatie uit de bestudeerde bachelor eindrapporten bruikbaar is voor dit onderzoek.

Vervolgens wordt in paragraaf 2 toegelicht hoe de aanpak van een ingenieur voor het ontwerpen van een vakwerkligger vertaald kan worden naar een computermodel.

Paragraaf 3 gaat over vakwerkconstructies. De informatie uit deze paragraaf is nodig om te bepalen welke configuraties en verbindingstypen worden opgenomen in het computermodel.

In paragrafen 4 en 5 wordt theorie uit NEN-EN-1991 uiteengezet. Er is gekozen om niet alleen te verwijzen naar deze norm, maar om de theorie ook toe te lichten. Hier is voor gekozen, omdat het van groot belang is dat de verschillende parameters en variabelen op de juiste wijze verwerkt worden in het computermodel. Inzicht in de (onderlinge) afhankelijkheden van deze parameters en variabelen is hiervoor noodzakelijk.

Tot slot worden in paragraaf 6 enkele bekende optimalisatie aanpakken met elkaar vergeleken. Op basis van deze informatie wordt besloten welke aanpak het meest geschikt is voor dit onderzoek.

2.1 Bruikbare informatie uit bachelor eindrapporten

Door de jaren heen zijn de onderwerpen *optimalisatie* en *vakwerkconstructies* regelmatig aan bod gekomen bij bachelor eindprojecten. Zo zijn in 2014 twee onderzoeken uitgevoerd naar de krachtswerking van ruimtelijke vakwerken. Het eindrapport van A.J.W. Bloemers (2014) beschrijft de ontwikkeling van een computermodel waarmee de staafkrachten van een vakwerkboog bepaald kunnen worden. Dit model is ontwikkeld met het doel optimalisatie van vakwerkconstructies mogelijk te maken.

Het onderzoek van E. van der Stap (2014) had als doel het opstellen van ontwerpformules voor ruimtelijke vakwerkconfiguraties. Hierbij is gebruik gemaakt van het programma MatrixFrame om de krachtswerking van verschillende ontwerpen te bepalen. Vervolgens zijn verbanden gelegd tussen de afmetingen en/of configuraties en de gevonden krachten en vervormingen.

Onderzoeken waarbij gezocht wordt naar een optimale uitkomst zijn er in veel verschillende vormen. Door B. Elferink (2010) en K. Riemens (2011) is onderzoek uitgevoerd naar *de optimale koepel*. Bij beide onderzoeken zijn verschillende koepelvormen en -eigenschappen met elkaar vergeleken. Het optimale ontwerp voor een koepel is uiteindelijk door de studenten aan de hand van de vergelijkingen bepaalt.

Volgens een soortgelijke aanpak zijn in 2015 door M.P. Hoogendoorn en L. Koning onderzoeken uitgevoerd naar de optimalisatie van een vakwerkboog respectievelijk een vakwerkkoepel. Hoogendoorn heeft het ontwerp van de vakwerkboog geoptimaliseerd naar gewicht. Hierbij zijn twee typen dakconstructies met elkaar vergeleken. Hoogendoorn heeft gebruik gemaakt van de programmeertaal Python in combinatie met het rekenprogramma 3D Truss. Dit programma berekent de staafkrachten, knoopverplaatsingen en reactiekrachten. Aan de hand van een parameterstudie heeft Hoogendoorn het optimale ontwerp bepaald. (Hoogendoorn, 2015).

Koning heeft een optimalisatie uitgevoerd naar kosten en *Embodied Carbon Footprint* (ECF). De vakwerkkoepel wordt hierbij op bepaalde ontwerpcriteria beoordeeld. Het doel van het onderzoek is vaststellen hoe een ontwerper ondersteund kan worden bij het ontwerpen van een koepelconstructie. Hierbij wordt de vraag gesteld of de optimalisatie omschreven kan worden door een set ontwerpformules of dat het probleem te complex is en er een rekentool nodig is om de optimalisatie aan te pakken. Koning maakt, net als Hoogendoorn, gebruik van Python in combinatie met 3D Truss. Ook voert zij, net als Hoogendoorn, een een parameterstudie uit waarbij ze de mogelijke ontwerpen met elkaar vergelijkt. De conclusie van haar onderzoek luidt: *“dit verband valt niet eenvoudigweg te beschrijven door een set van ontwerpformules. De ontwerper kan het beste ondersteund worden met een rekentool om de kosten en ECF van verschillende ontwerpen te kunnen*

vergelijken." (Koning, 2015 p. 43). Daarbij benadrukt Koning dat '*zo'n rekentool*' zeer waardevol kan zijn. Zij schrijft: *"het effect van kleine ontwerpbeslissingen kan zeer groot zijn. Door verschillende eigenschappen van de koepel tactisch te kiezen, kan wel meer dan 100% bespaard worden op kosten en/of ECF."* (Koning, 2015 p. 42).

Hoewel '*meer dan 100%'* kostenbesparing erg rooskleurig klinkt en het hoogstwaarschijnlijk niet het geval is dat er geld gecreëerd wordt bij de optimalisatie van een vakwerk, geeft de conclusie van Koning wel duidelijk aan dat een optimalisatietool erg waardevol kan zijn. Koning wijst in haar rapport ook op de beschikbare software voor het bepalen van het optimum van een probleem met veel vrijheidsgraden en/of restricties (Koning, 2015 p. 46).

Het computermodel dat voor dit onderzoek ontwikkeld wordt, wordt geschreven in Python. In de eindrapporten van Hoogendoorn (2015) en Koning (2015), wordt beschreven hoe Python gecombineerd kan worden met het rekenprogramma 3D Truss. Dit programma berekent de staafkrachten, de knoopverplaatsingen en de reactiekrachten die optreden in de vakwerkconstructies. Voor het computermodel dat ontwikkeld wordt is een rekenkern nodig. Het programma 3D Truss is hier zeer geschikt voor. Aan de hand van de informatie uit de eindrapporten van Hoogendoorn (2015) en Koning (2015) worden Python en 3D Truss aan elkaar gekoppeld.

Voor het programma 3D Truss moet een .TRS-inputfile geschreven worden. Het format waarin en de wijze waarop de informatie uit Python weggeschreven wordt, wordt toegelicht in het eindrapport van Hoogendoorn (2015, p.18).

Het programma 3D Truss wordt aangestuurd vanuit Python. Dit wordt *batch mode running* genoemd. In het eindrapport van Koning (2015, p. 11) wordt een manier beschreven waarop 3D Truss in batch mode gerund kan worden. Hierbij wordt het programma geopend als subprocess. Er wordt gebruikt gemaakt van een timer om te voorkomen dat Python 3D Truss afsluit voordat het klaar is met de berekeningen en met het wegschrijven van de output. De timer is ingesteld op een wachttijd van circa 20 seconden. Deze wachttijd is door Koning bepaald aan de hand van een schatting van de runtijd van het programma. Iedere keer dat 3D Truss aangeroepen wordt, kan er gedurende de wachttijd niets anders gedaan worden. Koning geeft in haar aanbevelingen aan dat deze manier van batch mode running om twee redenen niet ideaal is. Ten eerste is de rekentijd erg groot door de toepassing van de timer. Ten tweede is het risico aanwezig dat het model eruit klapt door een te korte wachttijd (Koning, 2015 p.44). Koning raadt aan zelf een rekenkern te ontwikkelen in Python ter vervanging van 3D Truss.

Bij dit onderzoek wordt, ondanks de aanbeveling van Koning, toch gebruik gemaakt van 3D Truss als rekenkern. Hier is voor gekozen, omdat het naar verwachting teveel tijd zou kosten om zelf een rekenkern te ontwikkelen. Er zal een andere oplossing gezocht worden voor het probleem dat Koning aandraagt.

Bij dit onderzoek wordt gebruik gemaakt van de manier waarop Koning de resultaten van 3D Truss inleest in Python. Voor een duidelijke beschrijving van deze aanpak wordt verwezen naar p. 11 van haar eindrapport (Koning, 2015).

Het ontwerp van de vakwerkligger wordt voor dit onderzoek geoptimaliseerd naar kosten. Om deze optimalisatie uit te kunnen voeren, dienen de verschillende kostenposten beschreven te worden. Bij het bepalen van de kosten van de onderdelen van de vakwerkligger wordt gebruik gemaakt van de beschrijving van de kosten van de koepelconstructie uit het eindrapport van Koning (2015, p. 16).

2.2 Van Ingenieursaanpak naar Computermodel

In deze paragraaf wordt een ingenieursaanpak voor het ontwerpen van een vakwerkligger beschreven. De stappen die de ingenieur doorloopt, worden weergegeven in een flow diagram. Vervolgens wordt, op basis van dit flow diagram, beredeneerd welke modules er nodig zijn om een computermodel te ontwikkelen dat deze taken van de ingenieur overneemt. De informatie die nodig is om de modules te ontwikkelen wordt in de hieropvolgende paragrafen besproken.

2.2.1 Ingenieursaanpak voor het ontwerpen van een vakwerkligger

Op basis van de kennis en ervaring, die is opgedaan tijdens de bacheloropleiding Civiele Techniek, wordt een globale ingenieursaanpak opgesteld voor het ontwerpen van een vakwerkligger. Het doel hiervan is een overzicht creëren van de stappen van het ontwerpproces en de kennis die hierbij nodig is.

Een ingenieur die de opdracht krijgt om een vakwerkligger te ontwerpen zal in de regel beginnen met een analyse van de ontwerpvraag. Hierbij wordt gekeken naar de toepassing van de vakwerkligger, het bijbehorende mechanicaschema, de afmetingen en de belastingen die een rol spelen. Ook worden vaak eisen en wensen van de opdrachtgever meegenomen in deze analyse. De opdrachtgever kan bijvoorbeeld aangegeven dat de constructie een bepaalde uitstraling moet hebben of dat vanwege het architectonisch ontwerp bepaalde variabelen vastliggen. Dit zijn eisen en wensen waarmee rekening gehouden dient te worden bij het maken van het draagconstructief ontwerp van de ligger.

Na de analyse van de ontwerpvraag wordt er een variantenstudie uitgevoerd. Hierbij worden verschillende schetsontwerpen met elkaar vergeleken. Op basis van vuistregels en/of ervaring worden de afmetingen van de vakwerkligger, de configuratie en de dimensionering van de staven bepaald. Een of enkele ontwerpen worden vervolgens verder uitgewerkt, zodat er een berekening en toetsing uitgevoerd kan worden.

Alvorens berekeningen gemaakt kunnen worden, zal de ingenieur moeten bepalen welke belastingen werken op de vakwerkligger. Een overzicht van deze belastingen geeft de grootte, de richting en het type belasting weer. Fundamentele belastingcombinaties worden opgesteld en er wordt onderzocht welke belastingcombinaties maatgevend kunnen zijn en dus meegenomen moeten worden in de berekening en de toetsing.

De volgende stap is het doorrekenen van het ontwerp van de vakwerkligger. Hierbij worden de staafkrachten en de knoopverplaatsingen bepaald. Om het ontwerp te kunnen toetsen, wordt ook de capaciteit van de staven berekend. Het gaat hierbij om de normaalkracht- en de knikcapaciteit. Vervolgens wordt de toetsing uitgevoerd. De sterkte, stijfheid en stabiliteit van de vakwerkligger worden gecontroleerd. Deze toetsen worden uitgevoerd conform NEN-EN-1991.

Op basis van de uitkomsten van de toetsen past de ingenieur de gekozen afmetingen aan. Vervolgens wordt het ontwerp opnieuw doorgerekend en worden de toetsen nogmaals uitgevoerd. Dit iteratieve proces gaat door totdat het ontwerp voldoet aan alle eisen en/of er geen sprake (meer) is van overdimensionering. Hierbij wordt in de regel ook gekeken naar de kosten van het ontwerp. Deze grootheid wordt in termen van optimalisatie het *object* genoemd. Er wordt gestreefd om een zo laag mogelijke waarde te behalen voor het object.

2.2.2 Vertaling van de ingenieursaanpak naar modules van het computermodel

Voor het computermodel wordt een modulaire opbouw gehanteerd, zodat het model gemakkelijk uitgebreid kan worden. Dit houdt in dat het model wordt opgebouwd uit losstaande modules. De ingenieursaanpak, zoals hiervoor beschreven, is weergegeven in het *flow diagram* in Figuur 1. De vertaalslag van de ingenieursaanpak naar de modules van het computermodel wordt aan de hand van dit flow diagram toegelicht.

In het flow diagram zijn twee kaders aangegeven. Het groene kader omvat de stappen van de ingenieur die vertaald worden naar modules voor het computermodel. Er zijn vier stappen omkaderd, dus er dienen ook vier modules ontwikkeld te worden. Deze modules zijn als volgt: een ontwerpmodule, een module ter bepaling van de belastingen en de fundamentele combinaties, een rekenmodule en een toetsingsmodule.

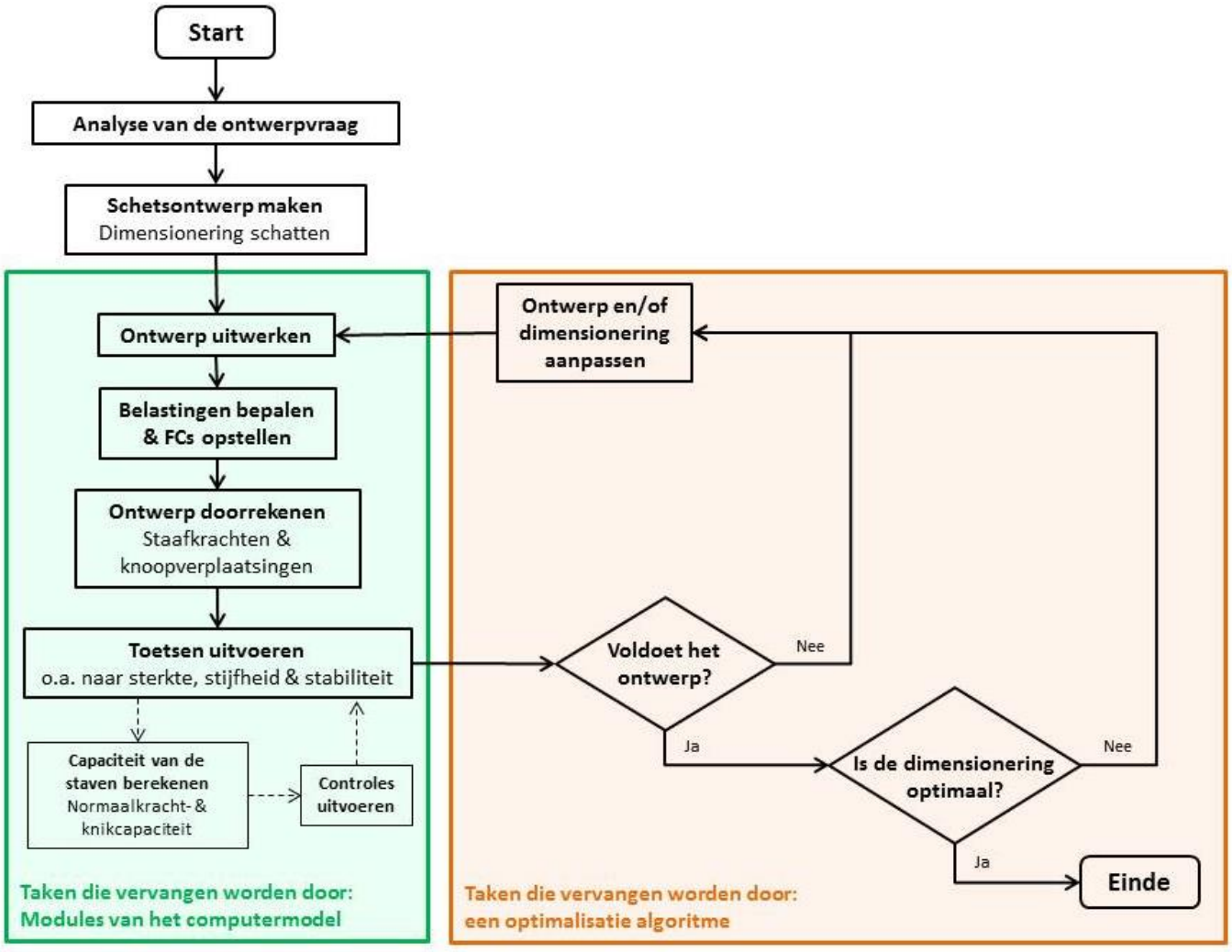
Het oranje kader geeft aan welke taken van de ingenieur overgenomen worden door een optimalisatie algoritme. Het gaat hierbij om het bepalen en aanpassen van het ontwerp en de dimensionering van de vakwerkligger. Het optimalisatie algoritme maakt bij deze aanpassingen

gebruik van de informatie die door de modules wordt aangeleverd, zoals de output van de toetsingsmodule.

Een ingenieur maakt bij het ontwerpen en dimensioneren van een vakwerkligger gebruik van diverse informatiebronnen. Denk bijvoorbeeld aan tabellenboekjes met productieafmetingen van verschillende profielen en de bijbehorende geometrische eigenschappen. Deze informatie heeft het computermodel ook nodig. In de vorm van een database met staafgegevens wordt deze informatie verwerkt in het model. Iedere module kan informatie uit deze database opvragen en gebruiken.

Twee taken van de ingenieur vallen buiten de twee kaders, zoals duidelijk zichtbaar is in Figuur 1**.** De eerste is het analyseren van de ontwerpvraag. Hierbij worden de specifieke gegevens van het ontwerp opgesteld. Het gaat hierbij onder andere om de locatie van de constructie. Deze *ontwerpgegevens* verschillen per optimalisatievraagstuk en dienen zodoende altijd als input aan het model te worden meegegeven. Om te zorgen dat de ontwerpgegevens overal in het model gemakkelijk gebruikt kunnen worden, wordt deze informatie in een constantsfile verwerkt.

De tweede taak die niet door het model overgenomen wordt, is stap waarbij het schetsontwerp gemaakt wordt en een eerste schatting voor de dimensionering wordt gedaan. Deze stap bepaalt als het ware het startpunt van het optimalisatieproces. Dit startpunt dient, net als de ontwerpgegevens, als input aan het model te worden meegegeven.



Figuur 1. Flow diagram Ingenieursaanpak incl. vertaling naar Computermodel

2.2.3 Beschrijving van de modules van het computermodel

Hieronder volgt een korte beschrijving van de modules waaruit het computermodel wordt opgebouwd. Alle modules, met uitzondering van de rekenmodule, worden zelf ontwikkeld. Voor de rekenmodule wordt gebruik gemaakt van het computerprogramma 3D Truss.

Database met staafgegevens & Constantsfile met ontwerpgegevens

Voor dit onderzoek worden één profiel en één staalsoort beschouwd, zoals is toegelicht in paragraaf 1.2. Het gaat hierbij om een warmgewalst buisprofiel van staalsoort S355. In de database worden de geometrische eigenschappen van buizen met diverse afmetingen verwerkt. De diameter en de wanddikte van van de buizen kunnen variëren binnen bepaalde grenzen. Deze grenzen worden bepaald binnen het optimalisatie algoritme.

De ontwerpgegevens, die door de gebruiker als input worden meegegeven aan het model worden opgeslagen in een constantsfile. Overal in het model kunnen de gegevens uit de constantsfile opgevraagd worden. Op die manier is deze informatie gemakkelijk te gebruiken en altijd beschikbaar zonder dat het daadwerkelijk als input aan de verschillende functies binnen het model meegegeven moet worden.

Ontwerpmodule

Binnen de ontwerpmodule wordt bepaald hoe de vakwerkligger eruit komt te zien. Het ontwerp is afhankelijk van de configuratie die toegepast wordt. In paragraaf 2.3.2 worden enkele configuraties van vakwerkliggers beschreven. Het ontwerp van de vakwerkligger wordt in de ontwerpmodule gedefinieerd door de locatie van de knooppunten te bepalen. Ook wordt aangegeven waar de staven zich bevinden en wat voor type staven het zijn. Ieder staaftype is gekoppeld aan specifieke informatie uit de database met staafgegevens.

Module ter bepaling van de belastingen en de fundamentele combinaties

Binnen dit onderdeel van het computersysteem wordt bepaald hoe groot de belastingen zijn die op de vakwerkligger werken. Ook worden de verschillende belastingen met elkaar gecombineerd in zogenaamde fundamentele combinaties (FCs). De belastingen die van invloed zijn op de overkapping van een tribune, worden besproken in paragraaf 2.4. Ook komt de wijze waarop de grootte van de belastingen en de FCs bepaald wordt, in deze paragraaf aan bod.

Rekenmodule

De rol van rekenkern binnen het computermodel wordt vervuld door het programma 3D Truss. 3D Truss is een programma dat de staafkrachten, de knoopverplaatsingen en de reactiekrachten in de steunpunten berekent. Hierbij maakt het programma gebruik van de verplaatsingenmethode. Deze methode gaat uit van het aantal vrijheidsgraden van de vakwerkconstructie en drukt vervolgens de lengteverandering van de staven uit in deze vrijheidsgraden. Er wordt verondersteld dat de staven lineair elastisch gedrag vertonen. Met dit verband kunnen ook de staafkrachten uitgedrukt worden in verplaatsingen. Het stelsel met onbekende verplaatsingen dat zo ontstaat, wordt opgelost met behulp van de vergelijkingen voor krachtenevenwicht in alle drie de richtingen (Welleman, 2007).

Toetsingsmodule

In de toetsingsmodule wordt gecontroleerd of de output van de rekenkern, de staafkrachten en de knoopverplaatsingen, voldoet aan de gestelde eisen. De sterkte, stijfheid en stabiliteit van de vakwerkligger worden getoetst. Ook worden er eisen gesteld van de verbindingen van het vakwerk. Hoe de toetsen uitgevoerd worden, wordt toegelicht in paragraaf 2.5.

Optimalisatie algoritme

Dit onderdeel van het computersysteem voert de optimalisatie van het ontwerp uit. Er wordt gebruik gemaakt van een bestaande optimalisatie aanpak om op een slimme en efficiënte manier op zoek te

gaan naar de optimale combinatie van de ontwerpvariabelen. In paragraaf 2.6 worden enkele bekende optimalisatie aanpakken besproken. In paragraaf 4.2 wordt toegelicht welke aanpak in het computermodel wordt verwerkt en waarom.

2.3 Vakwerken

Vakwerkconstructies komen voor in verschillende vormen en maten en kunnen op verschillende manier toegepast worden. In deze paragraaf worden de kenmerken van een zuivere vakwerkconstructie toegelicht. Vervolgens worden voor een uitkragende, vlakke vakwerkligger enkele mogelijke configuraties geïntroduceerd. Tot slot worden kort de toepassingsmogelijkheden van vakwerken beschreven.

2.3.1 Kenmerken van een zuiver vakwerk

Een vakwerk bestaat uit staven en knopen. Indien er wordt gesproken over een zuiver vakwerk, wordt er aangenomen dat de belastingen aangrijpen in de knooppunten en dat de staven zuiver scharnierend met elkaar verbonden zijn. Ook wordt er aangenomen dat in de staven alleen normaalkrachten optreden.

Deze aannames kloppen niet helemaal. In vakwerken treden namelijk ook dwarskrachten en momenten op. De verbindingen zijn nooit volledig scharnierend, doordat de randstaven vaak doorlopen over de gehele lengte van het vakwerk. Hierdoor kunnen ook dwarskrachten en momenten overgedragen worden. Daarnaast is er vaak sprake van enige excentriciteit tussen de aansluiting van de staven in de knooppunten, waardoor er excentriciteitsmomenten ontstaan. Tot slot ontstaan er nog momenten in de knooppunten door de vervorming van de staven onder invloed van de belastingen. Deze momenten worden secundaire momenten genoemd (Abspoel et al., 2013).

Verder is ook de aanname dat alle belastingen aangrijpen in de knooppunten niet zonder meer geldig. De belasting ten gevolge van het eigen gewicht van de staven werkt namelijk niet op de knopen, maar op de staven zelf. Hierdoor ontstaan momenten en dwarskrachten in de staven. Het eigen gewicht is echter zo klein dat deze momenten en dwarskrachten verwaarloosd kunnen worden. De toepassing van veiligheidsfactoren bij de toetsing van de constructie houdt rekening met dit soort vereenvoudigingen.

Dit onderzoek gaat ervan uit dat de te optimaliseren vakwerligger zich gedraagt als een zuiver vakwerk. De secundaire momenten zijn van beperkte grootte, omdat *“1) de rotatiestijfheid van de verbindingen tussen rand- en wandstaven meestal laag is en 2) de axiale stijfheden van de staven hoog zijn (en dus de vervormingen van een vakwerk klein zijn)”* (Abspoel et al., 2013). Op dit punt is deze aanname dus gerechtvaardigd. De veronderstellingen dat de staven zuiver scharnierend verbonden zijn en centrisch op elkaar aansluiten zijn echter niet zonder meer gerechtvaardigd. Dit onderzoek gaat hier niet verder op in, omdat de nadruk op het proces ligt en niet op het ontwikkelen van een geheel werkend model. Het wordt echter wel aanbevolen om bij uitbreiding van het model aandacht te schenken aan deze vereenvoudigingen.

2.3.2 Veelvoorkomende configuraties van vakwerkliggers

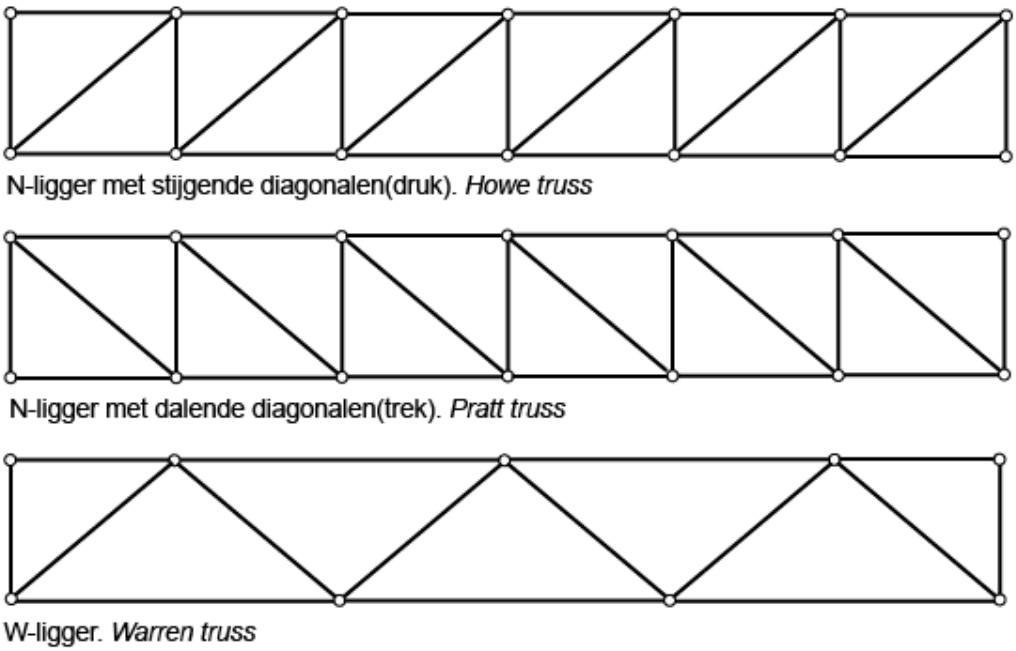
Een vakwerkligger, die onderdeel uitmaakt van de overkapping van een tribune kan verschillende vormen aannemen. Het doel van dit onderzoek is het ontwikkelen van een computermodel dat aangeeft hoe het optimale ontwerp van zo’n vakwerkligger eruitziet. Het bepalen van de optimale vorm, ofwel configuratie, is een belangrijk onderdeel hiervan.

De staven van een vakwerk zijn onder te verdelen in 2 groepen: randstaven en wandstaven. Bij de randstaven wordt verder onderscheid gemaakt tussen onder- en bovenrandstaven en bij de wandstaven tussen staanders en diagonalen. De diagonalen worden vervolgens onderverdeeld in stijgende en dalende diagonalen (Nijsse, 2012). Wanneer een 2D- of vlakke vakwerkligger wordt

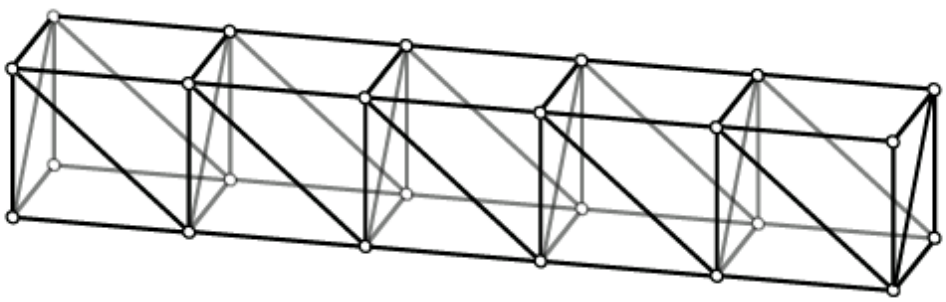
beschouwd, zijn er verschillende configuraties te benoemen. Drie bekende configuraties zijn de N-ligger met stijgende diagonalen *(Howe truss)*, de N-ligger met dalende diagonalen *(Pratt truss)* en de W-ligger *(Warren truss)* (Wardenier et al., 2010), zie Figuur 2. Kenmerkend voor de N-liggers is dat de diagonalen of door trek- of door drukkrachten belast worden. Dit in tegenstelling tot de W-ligger, waarbij de diagonalen op zowel trek- als drukkrachten gedimensioneerd moeten worden. Deze kenmerkende krachtsverdeling hoort bij een neerwaartse verticale belasting. Onder invloed van opwaartse windbelasting kan er echter een totaal andere krachtwerking ontstaan. Het is dan ook van belang dat hier voldoende aandacht aan wordt besteed (Welleman, 2007).

Vakwerkliggers kunnen als vlakke en als ruimtelijke vakwerken uitgevoerd worden. De benaming *ruimtelijk vakwerk* duidt in deze context niet op een vakwerk waarbij de belasting in meerdere richtingen wordt afgedragen. Net als bij een vlakke vakwerkligger draagt ook de *ruimtelijke* variant af in een richting. Het enige verschil is dat de doorsnede 3D is i.p.v. 2D. Een voorbeeld van een ruimtelijke vakwerkligger is weergeven in Figuur 3. Deze vakwerkligger heeft een vierkante doorsnede met een enkele diagonaal. De diagonaal is toegevoegd om de onderrandstaaf van de ligger zijdelings te ondersteunen.

Dit onderzoek richt zich op de optimalisatie van vlakke vakwerkliggers. De wijze waarop dit proces verloopt, zal voor ruimtelijke vakwerkliggers vergelijkbaar zijn. Voor dit onderzoek is deze uitbreiding dan ook niet noodzakelijk. De verschillende configuraties van ruimtelijke vakwerkliggers, worden zodoende niet verder toegelicht.



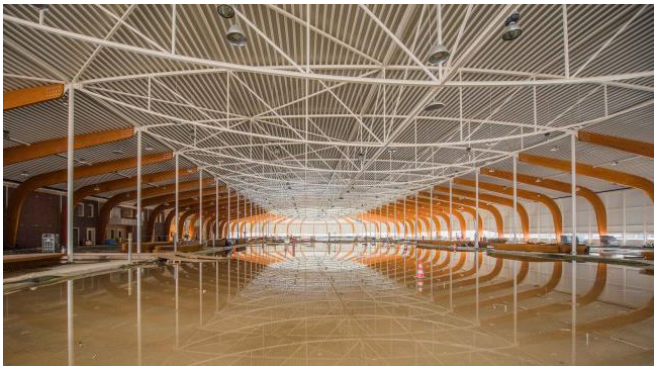
Figuur 2. Vlakke vakwerkconfiguraties



Figuur 3. Voorbeeld van een ruimtelijke vakwerkligger: Pratt configuratie en vierkante doorsnede met enkele diagonaal.

2.3.3 Toepassingsmogelijkheden

Vakwerkconstructies zijn uitermate geschikt voor het overspannen van grote afstanden in een of in twee richtingen. Denk bijvoorbeeld aan een brug, die bestaat uit vakwerkliggers of –bogen (Figuur 5) of aan de dakconstructie van een stationshal (Figuur 6 en Figuur 7), een sporthal (Figuur 4) of een stadion (Figuur 8 en Figuur 9). Ook in verticale richting worden vakwerken met grote lengte toegepast. Voorbeelden hiervan zijn een hijskraan of een hoogspanningsmast, maar ook de Eiffeltoren kan beschouwd worden als vakwerkconstructie. Dit rapport beschouwt de toepassing van een uitkragende, vlakke vakwerkligger als onderdeel van de overkapping van een tribune.



Figuur 4. Elfstedenhal, Leeuwarden. (Dolsma, 2015)

Figuur 5. Chaotianmen bridge, China. (Prandi, n.d.)

Figuur 6. Treinstation (Animaatjes, n.d.)

Figuur 7. Metro Station Blaak, Rotterdam. (Halkes, 2015)

Figuur 8. Al Shaqab Arena (OpenBuilding, n.d.)

Figuur 9. Ghelamco Arena (Lambert Engineering, 2015)

2.3.4 Verbindingen

Voor dit onderzoek wordt het ontwerp van een vakwerkligger geoptimaliseerd naar kosten. De kosten van een vakwerkligger zijn afhankelijk van het type verbindingen dat wordt toegepast. Hierbij wordt onderscheid gemaakt tussen las- en systeemverbindingen.

Lasverbindingen

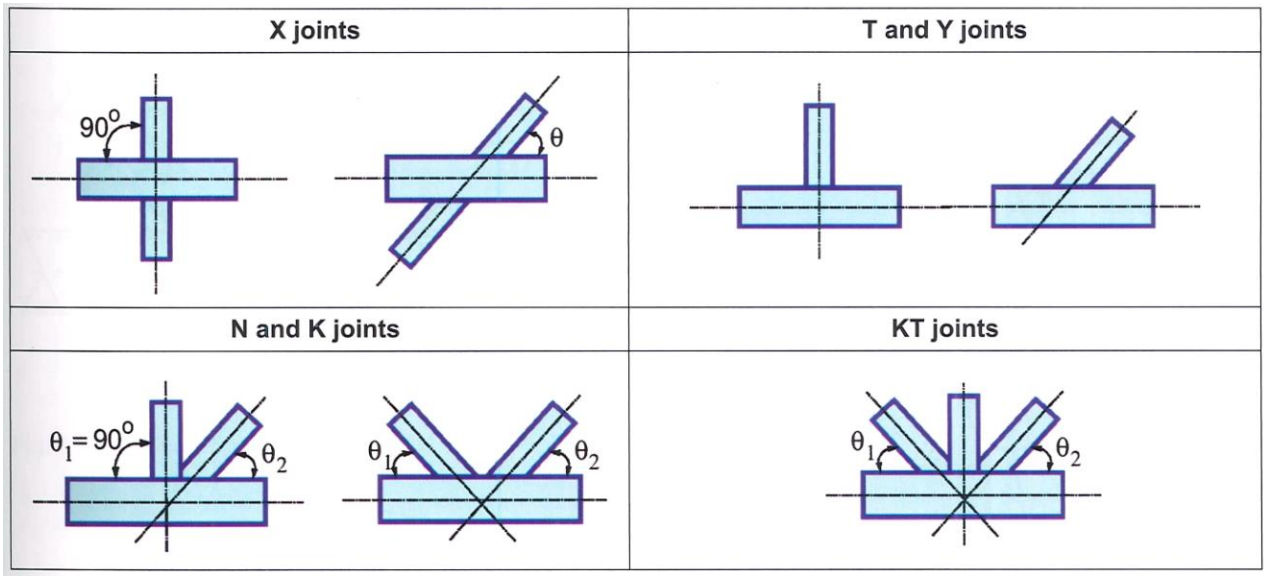
De lasverbindingen, die voor dit onderzoek beschouwd worden, worden in de fabriek onder goed te controleren omstandigheden vervaardigd. Hierbij worden de randstaven zoveel mogelijk als doorlopende staven toegepast. De lasverbindingen zijn dan ook verbindingen tussen de rand- en de wandstaven. Er wordt onderscheid gemaakt tussen zes knooptypen . Dit zijn de X-, T-, Y-, K-, N- en KT-knopen (Wardenier et al., 2010 p. 65). In Figuur 10 zijn deze knopen weergegeven.

Bij T- en Y-knopen wordt zowel de horizontale als de verticale component van de staafkracht uit de wandstaaf opgenomen door de randstaaf. Indien de hoek tussen de rand- en wandstaaf 90° bedraagt, wordt de knoop een T-knoop genoemd. Dit is het geval bij de verbinding tussen een staander en een randstaaf. De verbinding tussen een diagonaal en een randstaaf wordt een Y-knoop genoemd. Hierbij is de hoek tussen rand- en wandstaaf kleiner dan 90° (minimaal 30 en maximaal 60°).

K- en N-knopen zijn verbindingen waarbij de horizontale component van de staafkracht uit de wandstaaf wordt opgenomen door een andere wandstaaf. De twee wandstaven bevinden zich aan dezelfde kant van de randstaaf. Het verschil tussen K- en N-knopen is de hoek tussen de rand- en wandstaven. Indien een van de hoeken een rechte hoek is, wordt de knoop een N-knoop genoemd. Een knoop waar een staander en een diagonaal samenkomen, is dus een N-knoop. Een knoop waar twee diagonalen samenkomen, is daarentegen een K-knoop.

Bij X- knopen wordt, net als bij K- en N-knopen, de horizontale component van de staafkracht uit de wandstaaf opgenomen door een andere wandstaaf. Bij een X-knoop bevindt de tweede wandstaaf zich echter aan de andere kant van de randstaaf. Dit betekent dat de kracht van de ene wandstaaf via de randstaaf naar de andere wandstaaf wordt afgedragen.

Het zesde knooptype dat besproken wordt, is de KT-knoop. Deze knoop is, zoals de naam al aangeeft, een combinatie van de K- en T-knoop. Er worden twee diagonalen en een staander verbonden met de randstaaf.



Figuur 10. Classificatie knooptypen (Wardenier et al., 2010 p. 71)

Systeemverbindingen

Systeemverbindingen worden ook wel geprefabriceerde verbindingen genoemd. Voor dit onderzoek wordt aangenomen dat, indien systeemverbindingen worden toegepast, alle verbindingen van de vakwerkligger volgens hetzelfde principe vervaardigd worden. Een voorbeeld van een systeemverbinding is weergegeven in Figuur 11. Bij dit type systeemverbinding worden de staven met elkaar verbonden door een balvormig verbindingsstuk. Dit type wordt ook wel het *ball joint sytem* genoemd (Hunting, 2003). De staven worden hierbij in de regel met een schroefdraadverbinding aan de bal bevestigd.

Een ander voorbeeld is een systeemverbinding van het bedrijf *Octatube* uit Delft, afgebeeld in Figuur 12**.** Deze systeemverbinding bestaat uit een verbindingsstuk met staafuiteinden die aan elkaar gelast zijn. De staven van het vakwerk worden hierbij met bouten aan de staafuiteinden van het verbindingsstuk bevestigd (Octatube, n.d.).

Figuur 11. Ball joint system (Hunting, 2003)

Figuur 12. Octatube systeemoplossing (Octatube, n.d.)

2.4 Belastingen

In deze paragraaf wordt besproken welke constructieve eisen er gesteld worden aan de uitkragende vakwerkligger en hoe deze eisen tot stand zijn gekomen. Ook worden de belastingen die een rol spelen bij het ontwerp van de ligger beschreven en wordt toegelicht hoe de grootte van deze belastingen berekend moet worden. Dit alles is conform NEN-EN-1991.

2.4.1 Constructieve betrouwbaarheid

De constructieve betrouwbaarheid die van een bouwwerk geëist wordt, hangt af van zijn functie. Er wordt rekening gehouden met de kans op bezwijken en de gevolgen die daarbij horen. Hieronder vallen zowel het verlies van mensenlevens als economische en sociale schade. De overkapping van een tribune valt onder gevolgklasse 3, ofwel Consequence Class 3 (CC3). Voor deze gevolgklasse zijn de eisen met betrekking tot constructieve veiligheid het strengst. Gevolgklasse 3 correspondeert met betrouwbaarheidsklasse 3, ofwel Reliability Class 3 (RC3). Aan deze betrouwbaarheidsklasse zijn een betrouwbaarheidsindex β en een K_{FI}-factor gekoppeld. Hoe hoger de RC hoe hoger β en de factor K_{FI}. De K_{FI}-factor dient toegepast te worden bij toetsing van de uiterste grenstoestanden, oftewel de Ultimate Limit State (ULS). Hierbij wordt de K_{FI}-factor vermenigvuldigd met de partiële factoren. De K_{FI}-factor die hoort bij CC3 en RC3 is 1,1 (De Vries et al, 2013).

De partiële factoren, de veiligheidsfactoren, die bij controle van de ULS toegepast dienen te worden, zijn afgeleid van de betrouwbaarheidsindex β en hangen af van de *limit state* die bekeken wordt. Bij de controle van de ULS van de overkappingsconstructie hoort groep B. De veiligheidsfactoren die horen bij groep B en RC3 zijn in Tabel 1 weergegeven. Deze zijn al vermenigvuldigd met de K_{FI}-factor. In Tabel 1 zijn ook de veiligheidsfactoren weergegeven die bij toetsing van de Serviceability Limit State (SLS) toegepast moeten worden. Deze bedragen 1,0 voor zowel de permanente als de variabele belastingen. De karakteristieke waarden van de belastingen in de SLS zijn dus gelijk aan de rekenwaarden.

Tabel 1. Partiele factoren aan de belastingkant

Grenstoestand	Ontwerpsituatie/ belastingcombinatie	Betrouwbaarheids-klasse	Υ_G		Υ_Q
			Ongunstig	Gunstig	
ULS Groep B	Blijvende of tijdelijke ontwerpsituatie; Fundamentele combinaties	RC3	$\Upsilon_{G,hoog}$ 1,5	0,9	1,65
			$\Upsilon_{G,laag}$ 1,3	0,9	1,65
SLS	-	-	1,0	1,0	1,0

Bron: De Vries et al, 2013.

2.4.2 Permanente belastingen

De permanente belastingen die werken op overkappingsconstructies zijn de rustende belasting van de dakbedekking en het eigen gewicht van het vakwerk. De gegevens die nodig zijn om de grootte van deze belastingen te bepalen zijn:

- Gewicht van de dakbedekking [kN/m^2]
- Soortelijk gewicht staal: ρ_{staal} = 7850 kg/m^3 (Raaij et al., 2014)

2.4.3 Variabele belastingen

De variabele belastingen die van belang zijn bij het ontwerpen van een overkappingsconstructie zijn de wind- en de sneeuwbelasting en de belasting door goederen & personen. Voor deze drie belastingen worden de benodigde formules hieronder beschreven. Ook wordt de betekenis van de parameters uit de formules uiteengezet.

- **Windbelasting:** $\mathbf{F_{wind} = c_s c_d * c_f * q_p(z_e) * A_{ref}}$

Waarbij:

$c_s c_d$ = bouwwerkfactor = 1,0 [-]
c_f = krachtcoëfficient [-]
$q_p(z_e)$ = extreme stuwdruk op referentie hoogte z_e [kN/m^2]
A_{ref} = referentie oppervlak [m^2]

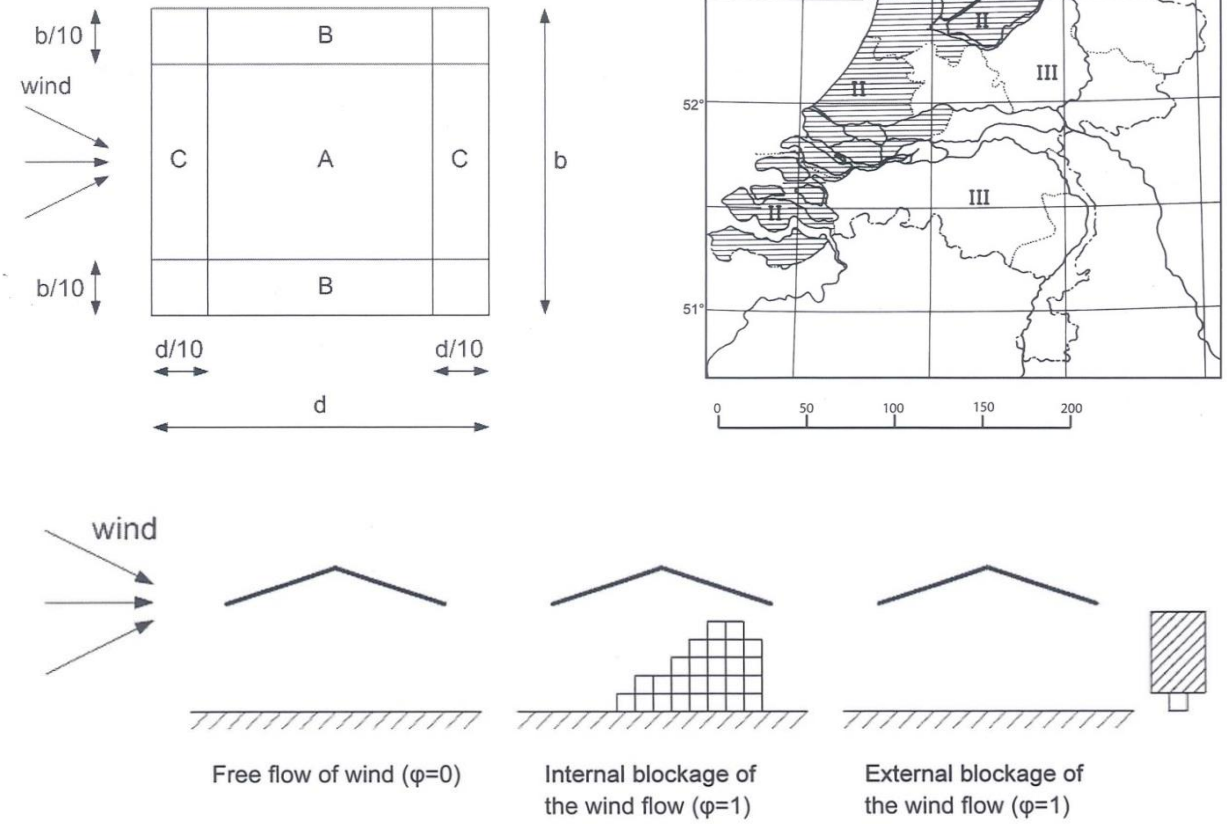
De bouwwerkfactor $c_s c_d$ houdt rekening met het feit dat de wind een ongelijke belasting uitoefent op het bouwwerk. Windvlagen zorgen ervoor dat stukken van het gevel- of dakoppervlak zwaarder belast worden dan de rest. De norm NEN-EN-1991-1-4 geeft aan dat in de meeste gevallen een waarde van 1,0 aangehouden mag worden (Raaij et al., 2014). In dat geval wordt uitgegaan van een extreme windbelasting op het totale beschouwde oppervlak. Indien grote oppervlakten beschouwd worden mag de bouwwerkfactor gereduceerd worden tot een minimum waarde van 0,85.

De krachtcoëfficient c_f is afhankelijk van de vorm van het bouwwerk. De waarde van de krachtcoëfficient verschilt per zogenaamde windzone van de schil van het bouwwerk. Voor het ontwerp van een overkappingsconstructie van een tribune gelden de regels die horen bij

luifelconstructies met interne en/of externe blokkade van de wind, zie Figuur 13 onder. Het dakvlak wordt hierbij opgedeeld in de zones A, B en C (zie Figuur 13, linksboven). Voor luifelconstructies is de krachtcoëfficient ook afhankelijk van de dakhelling α.

De extreme stuwdruk q_p is niet alleen afhankelijk van de bouwwerkhoogte z_e, maar ook van de locatie en directe omgeving van het bouwwerk. Nederland is onderverdeeld in drie windgebieden, zie Figuur 13 rechtsboven. De windgebieden zijn verder onderverdeeld in drie soorten omgevingen, namelijk: 'rural', 'urban' en 'coastal', oftewel landelijk, stedelijk en kustgebied.

Met het referentie oppervlak A_{ref} wordt het oppervlak bedoeld waar de windbelasting op werkt. Voor het ontwerp van een vakwerkligger als onderdeel van een overkappingsconstructie is dit het dakvlak.



Figuur 13. Linksboven: Dakoppervlak verdeeld in windzones. Rechtsboven: Nederland verdeeld in windgebieden. Onder: Blokkadefactor φ voor drie verschillende gevallen. (Raaij et al., 2014)

- **Sneeuwbelasting:** $F_{sneeuw} = \mu_i * C_e * C_t * s_k$

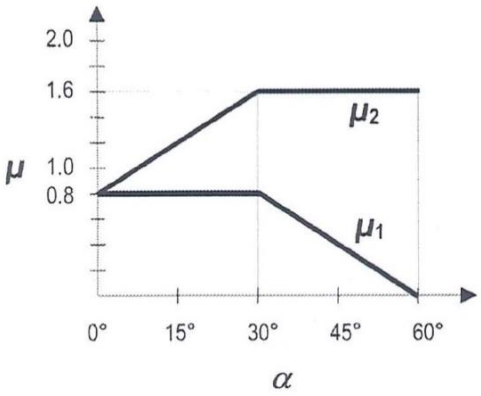
Waarbij:

μ_i = vormcoëfficient [-]
C_e = blootstellingscoëfficient = 1,0 [-]
C_t = warmtecoëfficient = 1,0 [-]
s_k = karakteristieke sneeuwbelasting op de grond = 0,7 kN/m^2

In Nederland mag voor zowel de blootstellingscoëfficient, c_e, als de warmtecoëfficient, c_t, de waarde 1,0 gebruikt worden. Dit staat aangegeven in de Nationale Bijlage van NEN-EN 1991-1-3.

De karakteristieke sneeuwbelasting op de grond, s_k, is eveneens vastgelegd in de Nationale Bijlage van deze norm en bedraagt in Nederland 0,7 kN/m^2 (De Vries et al., 2013).

Er wordt onderscheid gemaakt tussen 2 vormcoëfficienten: μ_1 en μ_2. Vormcoëfficient μ_2 wordt toegepast indien er sprake is van ophoping van sneeuw. Indien dat niet het geval is wordt, logischerwijs, μ_1 toegepast. Verder is de vormcoëfficient ook afhankelijk van de dakhelling α, zoals te zien is in Figuur 14**.**



Figuur 14. Vormcoëfficienten voor bepaling van de sneeuwbelasting (Raaij et al., 2014)

- **Belasting door goederen & personen:**

De grootte van deze verticale variabele belasting hangt af van het gebruik van het gebouw. Kenmerkend voor deze belasting is het feit dat de grootte gedurende de levensduur van het bouwwerk, sterk kan variëren. Deze belasting houdt onder andere rekening met onderhoudswerkzaamheden, waarbij op een gedeelte van het dak een relatief grote belasting werkt (Raaij et al., 2014).

Er dient rekening gehouden te worden met twee vormen van belasting door goederen en personen, namelijk de gelijkmatig verdeelde belasting q_k en de puntlast Q_k. De gelijkmatig verdeelde belasting werkt op een oppervlak van maximaal 10 m^2 en de puntlast op een oppervlak van 0,1x0,1 m^2. Het oppervlak dat belast wordt, dient zo ongunstig mogelijk gekozen te worden.

Voor daken hangt de grootte van deze belastingen af van de toegankelijkheid van het dak en van de dakhelling α.

2.4.4 Fundamentele belastingcombinaties

De beschreven belastingen dienen gecombineerd te worden in de zogenaamde fundamentele combinaties (FCs). Zowel voor toetsing van de Ultimate Limit State (ULS) als van de Serviceability Limit State (SLS) dienen deze combinaties opgesteld te worden.

De permanente belastingen zijn altijd aanwezig. Indien deze belastingen als maatgevend worden aangenomen, dient de hoge veiligheidsfactor $\Upsilon_{G,hoog}$ toegepast te worden. Wanneer echter wordt uitgegaan van een maatgevende variabele belasting, wordt de lage veiligheidsfactor $\Upsilon_{G,laag}$ gebruikt.

Voor de variabele belastingen geldt dat er slechts een variabele belasting tegelijk zijn extreme waarde kan bereiken. De overige variabele belastingen dienen vermenigvuldigd te worden met de gelijktijdigheidsfactor Ψ_0. Deze gelijktijdige waarden van de variabele belastingen, Ψ_0*Q komen in iedere belastingcombinatie voor. De waarde van Ψ_0 is voor zowel wind- als sneeuwbelasting als belasting door goederen en personen gelijk aan 0. Dit betekent dat de gelijktijdige waarden van deze belastingen uit de belastingcombinatie verdwijnen. De reden dat de gelijktijdigheidsfactoren 0 zijn, is dat bijvoorbeeld bij zeer harde wind de sneeuw van het dak wordt geblazen, waardoor er alleen sprake is van extreme windbelasting . Ook wordt er aangenomen dat onderhoudswerkzaamheden niet worden uitgevoerd indien er sprake is van extreme sneeuwval of extreme wind.

De fundamentele belastingcombinaties zijn:

FCI. Permanente belasting is maatgevend

FCII. Windbelasting is maatgevend

a. Neerwaartse windbelasting
b. Opwaartse windbelasting

FCIII. Sneeuwbelasting is maatgevend

FCIV. Belasting door goederen & personen is maatgevend
a. Gelijkmatige verdeelde belasting door goederen & personen
b. Belasting door goederen & personen in de vorm van een puntlast

FCI. Permanente belasting is maatgevend

$\gamma_{G,hoog} * G$ '+' $\Sigma \gamma_Q * Q_i * \Psi_{0,i}$

$1,5 * G_{dak}$ '+' $1,5 * G_{E.G.}$ '+' $1,65 * Q_{wind} * 0$ '+' $1,65 * Q_{sneeuw} * 0$ '+' $1,65 * Q_{g\&p} * 0$

$1,5 * G_{dak}$ '+' $1,5 * G_{E.G.}$

FCII. Windbelasting is maatgevend

a. Neerwaartse windbelasting

$\gamma_{G,laag} * G$ '+' $\gamma_Q * Q_{wind,neer}$ '+' $\Sigma \gamma_Q * Q_i * \Psi_{0,i}$

$1,3 * G_{dak}$ '+' $1,3 * G_{E.G.}$ '+' $1,65 * Q_{wind,neer}$ '+' $1,65 * Q_{sneeuw} * 0$ '+' $1,65 * Q_{g\&p} * 0$

$1,3 * G_{dak}$ '+' $1,3 * G_{E.G.}$ '+' $1,65 * Q_{wind,neer}$

b. Opwaartse windbelasting

$\gamma_{G,gunst} * G$ '+' $\gamma_Q * Q_{wind,op}$ '+' $\Sigma \gamma_Q * Q_i * \Psi_{0,i}$

$0,9 * G_{dak}$ '+' $0,9 * G_{E.G.}$ '+' $1,65 * Q_{wind,op}$ '+' $1,65 * Q_{sneeuw} * 0$ '+' $1,65 * Q_{g\&p} * 0$

$0,9 * G_{dak}$ '+' $0,9 * G_{E.G.}$ '+' $1,65 * Q_{wind,op}$

FCIII. Sneeuwbelasting is maatgevend

$\gamma_{G,laag} * G$ '+' $\gamma_Q * Q_{sneeuw}$ '+' $\Sigma \gamma_Q * Q_i * \Psi_{0,i}$

$1,3 * G_{dak}$ '+' $1,3 * G_{E.G.}$ '+' $1,65 * Q_{sneeuw}$ '+' $1,65 * Q_{wind} * 0$ '+' $1,65 * Q_{g\&p} * 0$

$1,3 * G_{dak}$ '+' $1,3 * G_{E.G.}$ '+' $1,65 * Q_{sneeuw}$

FCIV. Belasting door goederen & personen is maatgevend

a. Gelijkmatige verdeelde belasting door goederen & personen
(LET OP! Verschillende variaties met betrekking tot de plaats van de belasting)

$\gamma_{G,laag} * G$ '+' $\gamma_Q * q_{g\&p,A}$ '+' $\Sigma \gamma_Q * Q_i * \Psi_{0,i}$

$1,3 * G_{dak}$ '+' $1,3 * G_{E.G.}$ '+' $1,65 * q_{g\&p,A}$ '+' $1,65 * Q_{wind} * 0$ '+' $1,65 * Q_{sneeuw} * 0$

$1,3 * G_{dak}$ '+' $1,3 * G_{E.G.}$ '+' $1,65 * q_{g\&p,A}$

b. Belasting door goederen & personen in de vorm van een puntlast
(LET OP! Verschillende variaties met betrekking tot de plaats van de belasting)

$\gamma_{G,laag} * G$ '+' $\gamma_Q * Q_{g\&p,F}$ '+' $\Sigma \gamma_Q * Q_i * \Psi_{0,i}$

$1,3 * G_{dak}$ '+' $1,3 * G_{E.G.}$ '+' $1,65 * Q_{g\&p,F}$ '+' $1,65 * Q_{wind} * 0$ '+' $1,65 * Q_{sneeuw} * 0$

$1,3 * G_{dak}$ '+' $1,3 * G_{E.G.}$ '+' $1,65 * Q_{g\&p,F}$

2.5 Toetsingsregels

De constructie van een bouwwerk moet gecontroleerd te worden naar sterkte, stabiliteit en stijfheid. Deze controles worden uitgevoerd in de vorm van Unity Checks (UCs). Er worden ook eisen gesteld aan de verbindingen van het vakwerk. In deze paragraaf wordt beschreven hoe de normaalkracht- en de knikcapaciteit bepaald moeten worden en welke eisen er gesteld worden aan de vakwerkligger.

2.5.1 Sterkte: Normaalkrachtcapaciteit

Een kenmerk van *zuivere* vakwerkconstructies, zoals beschreven in paragraaf 2.3.1, is dat er alleen normaalkrachten optreden in de staven. Dit betekent dat de staven op het gebied van sterkte alleen getoetst hoeven te worden op normaalkracht. Om deze controle uit te kunnen voeren, moet eerst de normaalkrachtcapaciteit van de staven berekend worden. De capaciteit geeft aan wat de maximale kracht is die een staaf kan afdragen.

De rekenwaarde van de normaalkrachtcapaciteit wordt als volgt berekend:

$$\mathbf{N_{Rd}} = \frac{\mathbf{A} * \mathbf{f_y}}{\mathbf{\Upsilon_{M,0}}}$$

Met: $\Upsilon_{M,0}$ = materiaalfactor van staal = 1,0

De bijbehorende Unity Check is:

$$\mathbf{UC\_N} = \frac{\mathbf{N_{Ed}}}{\mathbf{N_{Rd}}} \leq \mathbf{1,0}$$

Waarbij N_{Ed} de rekenwaarde is van de normaalkracht die optreedt in de staaf.

2.5.2 Stabiliteit: Knikcapaciteit

Er wordt onderscheid gemaakt tussen twee vormen van instabiliteit van de staven. Dit zijn kip- en knikinstabiliteit. Voor dit onderzoek worden enkel buisprofielen beschouwd, waardoor er geen sprake is van kippen. Deze instabiliteitsvorm wordt dan ook niet verder behandeld.

De knikstabiliteit van de op druk belaste staven moet wel getoetst worden. Hiervoor dient eerst de knikcapaciteit van de op druk belaste staven bepaald te worden. Met de knikcapaciteit wordt de weerstand tegen knikken aangeduid.

De knikcapaciteit van een op druk belast element is de normaalkrachtcapaciteit vermenigvuldigd met de zogenaamde knikfactor χ. De knikfactor wordt berekend met onderstaande formule. Om de grootte van de knikfactor van een op druk belaste staaf te kunnen bepalen, zijn de imperfectiefactor en de relatieve slankheid van deze staaf nodig.

$$\boldsymbol{\chi} = \frac{\mathbf{1}}{\boldsymbol{\phi} + \sqrt{(\boldsymbol{\phi}^2 - \boldsymbol{\lambda}_{\mathbf{rel}}^2)}}$$

$$\boldsymbol{\phi} = \mathbf{0,5} * [\mathbf{1} + \boldsymbol{\alpha} * (\boldsymbol{\lambda}_{rel} - \mathbf{0,2}) + \boldsymbol{\lambda}_{\mathbf{rel}}^2]$$

Met: α = imperfectiefactor
λ_{rel} = relatieve slankheid

De imperfectiefactor α hangt af van de instabiliteitscurve van de doorsnede van de staaf, ook wel de knikkromme genoemd. De knikkrommen zijn opgesteld aan de hand van beproevingen en geven het instabiliteitsgedrag van het het op druk belaste element weer (Abspoel et al., 2013). Bij buisprofielen wordt er onderscheid gemaakt tussen de productiewijze en de staalsoort. Voor dit onderzoek worden warmgewalste buizen van staalsoort S355 beschouwd. In Tabel 2 zijn de knikkromme en imperfectiefactor, die hierbij horen weergegeven.

Tabel 2. Knikkrommen en imperfectiefactoren voor op druk belaste elementen van staalsoort S355

Doorsnede	Begrenzingen	Knik om de as	Knikkromme	Imperfactiefactor α
Buisprofiel	Warmvervaardigd	Elke as	a	0,21

Bron: Abspoel, R., De Vries, P.A. & Bijlaard, F.S.K., 2013.

De relatieve slankheid λ_{rel} geeft de verhouding weer tussen de slankheid van het element en de grensslankheid. Om de relatieve slankheid te bepalen, moeten dus eerst de slankheid en de grensslankheid berekend worden. De benodigde formules en toelichting van deze begrippen zijn hieronder weergegeven.

$$\boldsymbol{\lambda_{rel} = \frac{\lambda}{\lambda_e}} \qquad \boldsymbol{\lambda = \frac{l_{buc}}{i}} \qquad \boldsymbol{\lambda_e = \pi * \sqrt{(\frac{E}{f_y})}}$$

$$\boldsymbol{i = \sqrt{(\frac{I}{A})}} \qquad \boldsymbol{I = \frac{1}{64} * \pi * (d^4 - (d - t)^4)}$$

Met: λ = slankheid; λ_e = grensslankheid; I = traagheidsmoment van een cirkel; l_{buc} = kniklengte; i = traagheidsstraal; A = oppervlakte van de doorsnede; E = elasticiteitsmodulus; f_y = vloeispanning; d = diameter; t = wanddikte

De grensslankheid is de slankheid waarbij de knikspanning gelijk is aan de vloeispanning. De grensslankheid is dan ook een materiaaleigenschap. De slankheid daarentegen kan verschillen per element en hangt af van de kniklengte l_{buc} en de traagheidsstraal i. Hierbij is het van belang dat de kniklengte op de juiste wijze wordt bepaald. Voor de kniklengtes van onderdelen van een vakwerk, waarbij alle verbindingen scharnierend zijn, worden de volgende regels gehanteerd:

- Randstaven in het vlak van het vakwerk: $l_{buc} = l_{sys}$
- Randstaven uit het vlak van het vakwerk: l_{buc} = ongesteunde lengte
- Wandstaven in en uit het vlak: $l_{buc} = l_{sys}$

Zodra de knikfactor berekend is, kan ook de knikcapaciteit bepaald worden. De rekenwaarde van de knikcapaciteit is:

$$\boldsymbol{N_{b,Rd} = \frac{\chi * A * f_y}{\Upsilon_{M,1}}}$$

Met: $\Upsilon_{M,1}$ = modelfactor voor stabiliteitsberekeningen = 1,0

De bijbehorende Unity Check is:

$$\boldsymbol{UC\_Nb = \frac{N_{Ed}}{N_{b,Rd}} \leq 1,0}$$

Waarbij N_{Ed} de rekenwaarde is van de normaaldrukkracht waar het element door belast wordt.

2.5.3 Stijfheid: Verticale vervormingen

Naast eisen op gebied van sterkte en stabiliteit worden er ook eisen gesteld aan de stijfheid van een constructie. Bij deze eisen wordt onderscheid gemaakt tussen de maximaal toegestane verticale en horizontale verplaatsing van de constructie. Voor een dakconstructie die niet belast wordt door horizontale krachten (de windkracht resulteert in een opwaartse of neerwaartse belasting), is er geen sprake van horizontale verplaatsingen van de constructie. In dat geval hoeft dan ook alleen een controle van de verticale verplaatsing uitgevoerd te worden.

Voor een vakwerkligger die onderdeel uitmaakt van een dakconstructie geldt de volgende eis met betrekking tot de maximaal toegestane verticale verplaatsing:

Verticale verplaatsingseis: $w \leq 0{,}004 * L_{rep}$

Waarbij: L_{rep} = lengte van de overspanning ofwel twee keer de lengte van de uitkraging (De Vries et al., 2013)

2.5.4 Eisen met betrekking tot de verbindingen

Een uitgebreide toetsing van de verbindingen reikt buiten de strekking van dit onderzoek aangezien dit pas in de masteropleiding wordt behandeld. Aan de hand van algemene toetsen kunnen er echter toch controles uitgevoerd worden. Op die manier wordt dit onderdeel, ondanks de beperkte kennis van verbindingen, toch verwerkt in het computermodel.

Er worden eisen gesteld aan de afmetingen van de staven en de hoek tussen de randstaven en de diagonalen. De eisen die gesteld worden aan de verbindingen, dienen ertoe de sterkte van de verbindingen te verzekeren (Wardenier et al., 2010). De eis die gesteld wordt aan de hoek tussen randstaaf en diagonaal komt voort uit de minimale ruimte die nodig is om een goede lasverbinding te verkrijgen. Daarbij is deze eis ook van belang voor de krachtsverdeling binnen de vakwerkligger. Door de hoek niet te klein te kiezen, wordt voorkomen dat de horizontale component van de kracht in de diagonaal onnodig groot wordt. Indien hier geen rekening mee gehouden zou worden, zouden de randstaven extra belast worden wat resulteert in een ongewenste situatie. Een overzicht van de eisen die gesteld worden aan staafafmetingen en de hoek is weergegeven in Tabel 3.

Tabel 3. Eisen met betrekking tot de verbindingen

	Beschrijving	Eis
Staafafmetingen	Per staafgroep: Verhouding diameter/wanddikte	$\frac{d}{t} \leq 50{,}0$
	Verhouding diameter wandstaaf / diameter randstaaf	$0{,}2 \leq \frac{d_{wandstaaf}}{d_{randstaaf}} \leq 1{,}0$
	Verhouding wanddikte randstaaf / wanddikte wandstaaf	$\frac{t_{randstaaf}}{t_{wandstaaf}} \geq 2{,}0$
Hoek	Begrenzing hoek tussen randstaaf en diagonaal (en daarmee ook hoek tussen diagonaal en staander)	$30° \leq \theta \leq 60°$

Bron: G.J., Wardenier, Packer, J.A., J. & Zhao, X.-L. & Van der Vegte (2010).

2.6 Optimalisatie aanpakken

Er zijn verschillende aanpakken mogelijk om een optimalisatievraagstuk op te lossen. Een belangrijk kenmerk van een aanpak is het type probleem, of de typen problemen, waarvoor de aanpak geschikt is. Er worden drie probleemtypen gedefinieerd: *continuous problems, discrete problems* en *mixed continuous-discrete problem* (Straathof et al., 2008 p. 12).

Een allesomvattende aanpak, die voor alle probleemtypen toepasbaar is, is de *brute force approach*. Hierbij worden alle mogelijke waarden van de ontwerpvariabelen met elkaar gecombineerd. Alle mogelijke opties worden doorgerekend en vervolgens wordt het beste ontwerp geselecteerd. Deze aanpak is echter niet praktisch indien er sprake is van een lange rekentijd. In dat geval is het zaak dat een slimmere aanpak wordt toegepast. Hieronder worden drie bekende optimalisatie aanpakken beschreven.

Een bekende optimalisatie aanpak voor problemen met continue ontwerpvariabelen is de *gradient-based optimization method* (Straathof et al., 2008 p. 12). De gradient-based aanpak maakt gebruik van de afgeleide van de zogenoemde *objective function*. Deze functie berekent de te optimaliseren grootheid. Door de richting van de afgeleide te berekenen, bepaalt het optimalisatie algoritme of de ontwerpvariabelen vergroot of verkleind moeten worden om een beter resultaat te verkrijgen. Een nadeel van de gradient-based optimization method is het risico dat er als resultaat een lokaal optimum gevonden wordt i.p.v. het globale optimum (Straathof et al., 2008 p. 13 & Varoquaux, n.d.).

Een optimalisatie aanpak die beter is in het bepalen van het globale optimum is het *genetic search algorithm* (Straathof et al., 2008 p. 13). Deze aanpak is geschikt voor alle probleemtypen. Het genetic search algorithm is gebaseerd op het idee van natuurlijke selectie ook wel aangeduid met de term *survival of the fittest*. De startpopulatie wordt gevormd door een aantal mogelijke uitkomsten. Een uitkomst bestaat uit een set waarden van de ontwerpvariabelen. De mogelijke uitkomsten worden beoordeeld op hun *fitness*. De criteria hiervoor zijn afhankelijk van de eisen waaraan de uitkomst dient te voldoen: de constraints. Binnen de startpopulatie worden paartjes gevormd. Deze paartjes maken *nakomelingen,* oftewel nieuwe uitkomsten. Deze nieuwe uitkomsten bestaan uit een mix van de waarden van de ontwerpvariabelen van de *ouders*. Vervolgens wordt de nieuwe generatie beoordeeld op hun *fitness.* Dit proces van evolutie wordt herhaald totdat de *fitness*-score van de *ouders* niet meer overtroffen wordt door die van de nieuwe generatie of totdat wordt voldaan aan een vooraf gedefinieerde eis. (Jacobson, 2012).

Een andere discrete optimalisatie methode is de *particle swarm optimization method.* Deze methode is ontstaan vanuit een studie naar patronen van zwermen vogels en scholen vissen. De methode is net als het genetic search algorithm geschikt voor alle probleemtypen. Een andere overeenkomst met het genetic search algorithm is dat er gestart wordt met een aantal mogelijke uitkomsten (Hu, 2006). Iedere uitkomsten is een punt binnen de ontwerpruimte. De ontwerpruimte omvat alle mogelijke uitkomsten van het optimalisatieprobleem. Bij deze methode wordt de optimale uitkomst de *target value* genoemd. Het algoritme dat hoort bij de particle swarm optimization houdt bij welk punt, welke uitkomst, zich het dichtst bij de *target value* bevindt. Dit wordt de *global best value* of *gBest* genoemd. De andere punten die gedefinieerd zijn, bewegen in de richting van deze *gBest.* De snelheid waarmee de punten bewegen is afhankelijk van hun afstand tot de locatie van de *gBest.* Hoe groter de afstand, hoe hoger de snelheid. Van ieder punt wordt een *personal best value, pBest,* bijgehouden. De waarde van de *gBest* verandert zodra de *pBest* van een mogelijke uitkomst zich dichterbij de *target value* bevindt. Op die manier komt, volgens een iteratief proces, de *gBest* steeds dichterbij de *target value* te liggen. Dit proces gaat door totdat de *target value* of het maximum aantal iteraties bereikt is. (McCullock, 2012).

Het verschil tussen continue en discrete optimalisatie methoden is dat discrete methoden vaak beter zijn in het bepalen van het globale optimum. Daar staat tegenover dat continue methoden in de regel een stuk sneller een optimum kunnen vinden (Straathof et al., 2008 p. 13).

In de SciPy library (SciPy, 2016a) zijn optimalisatie algoritmen opgenomen die werken volgens verschillende optimalisatie aanpakken. Twee van deze optimalisatie algoritmen, een continue en een discrete, worden met elkaar vergeleken. De andere algoritmen zijn vooraf afgevallen wegens te beperkte controle op het verloop van de optimalisatie.

De *scipy.optimize.minimize* functie maakt gebruik van de gradient-based optimization method. Deze functie is dan ook geschikt voor continue optimalisatie problemen. Het is mogelijk om bounds en constraints mee te geven aan de functie (SciPy, 2016c). Op die manier kan de ontwerpruimte begrensd worden. Ook kan de tolerantie ingesteld worden (SciPy, 2016c), waardoor het detail waarin geoptimaliseerd wordt naar wens bepaald kan worden. Deze mogelijkheden zorgen ervoor dat het optimalisatieproces goed te sturen is. Door constraints mee te geven, wordt verzekerd dat de uitkomst voldoet aan de gestelde eisen. Daarbij kan door de bounds dichterbij elkaar te leggen of door een hogere tolerantie in te stellen de rekentijd beperkt worden.

Een andere optimalisatie functie uit de SciPy library is *scipy.optimize.differential_evolution.* Deze functie is een genetic search algorithm en kan zodoende continue en discrete ontwerpvariabelen verwerken. Ook aan deze functie kunnen bounds meegegeven worden en ook kan de tolerantie ingesteld worden (SciPy, 2016b). Het is echter niet mogelijk om constraints mee te geven (SciPy, 2016b). Dit betekent dat, indien deze functie toegepast wordt, de toetsing van de constraints lost van het optimalisatie algoritme geïntegreerd moet worden in het computermodel.

3 Methodiek

In dit hoofdstuk wordt beschreven hoe de informatie uit het voorgaande hoofdstuk wordt verwerkt in het computermodel. In paragraaf 1 wordt de ontwerpvraag, die als uitgangspunt dient bij dit onderzoek, beschreven. In de daaropvolgende paragrafen worden de modules die voor het computermodel ontwikkeld worden, behandeld. Deze modules zijn: de ontwerpmodule, de module ter bepaling van belastingen en FCs, de rekenmodule en de toetsingsmodule.

3.1 Input van de gebruiker: Ontwerpgegevens

Om een computermodel te ontwikkelen dat bepaalt welk ontwerp optimaal is voor een vakwerkligger, is er informatie nodig over de ontwerpvraag. Deze informatie, de ontwerpgegevens, vormt de input voor het computermodel. De ontwerpgegevens geven onder andere informatie over de gewenste afmetingen van de constructie die ontworpen moet worden en over de locatie.

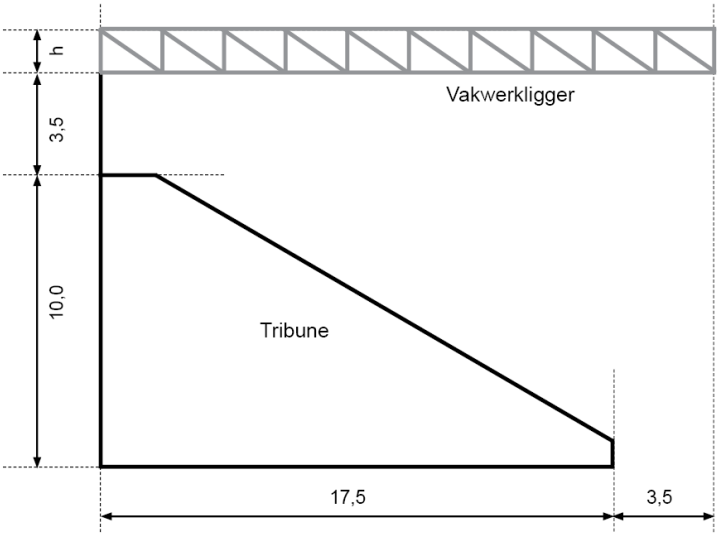
Voor dit rapport wordt één ontwerpvraag en dus ook één set ontwerpgegevens beschouwd. Het computermodel wordt echter wel zo ontwikkeld dat de ontwerpgegevens kunnen variëren en het computermodel dus bruikbaar is voor verschillende ontwerpvragen.

De ontwerpvraag die beschouwd wordt is: *"Welk ontwerp voor een uitkragende, vlakke vakwerkligger als onderdeel van de overkapping van een tribune is optimaal?".* De ontwerpgegevens die bij deze ontwerpvragen horen, zijn opgesteld aan de hand van het ontwerp van de bestaande overkapping van de Leo Halle Tribune van de club Go Ahead Eagles uit Deventer (zie Figuur 15 en Figuur 17). In deze paragraaf wordt toegelicht welke ontwerpgegevens het model nodig heeft om de optimalisatie uit te voeren en welke waarden gehanteerd worden voor dit onderzoek.

Figuur 15. Leo Halle Tribune (RTV Oost, 2015)

Leo Halle Tribune

De Leo Halle Tribune is onderdeel van het verbouwde stadion van de club *Go Ahead Eagles* uit Deventer. De tribune, die een dubbele capaciteit heeft ten opzichte van de oude Leo Halle tribune, werd in oktober 2015 in gebruik genomen. Het dak van de tribune bestaat uit uitkragende, vlakke vakwerkliggers van staal (Bartels, 2015) afgedekt met, naar het lijkt, trapeziumvormige dakplaten. De afmetingen en specificaties van de vernieuwde Leo Halle Tribune worden gebruikt als uitgangspunt voor de ontwerpvraag van dit onderzoek. Tabel 4 geeft een overzicht van deze ontwerpgegevens. Figuur 16 laat in doorsnede de situatie zien. De vakwerkligger die in dit figuur is afgebeeld, is slechts een voorbeeld met willekeurige hoogte en configuratie.



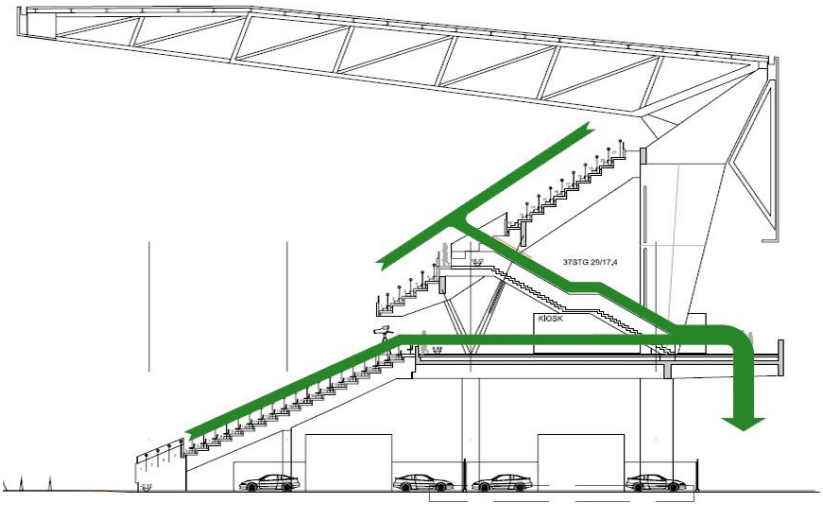
Figuur 16. Schematisch dwarsdoorsnede Leo Halle Tribune

Tabel 4. Ontwerpgegevens voor de overkapping van een tribune

Ontwerpgegevens		Invulling voor dit onderzoek
Tribune afmetingen	Capaciteit	2800 zitplaatsen
	Grondoppervlak	circa. 1300 m^2
	Breedte	75,0 m
	Lengte tribune	17,5 m
	Overspanning dakligger (span)	lengte tribune + 20% = 21,0 m
	Hoogte_zitgedeelte	10,0 m
	Vrije hoogte bovenaan	3,5 m
	Vrije hoogte_totaal	13,5 m
	H.o.h.-afstand liggers	8,0 m
Constructieve gegevens	Vorm van het dakvlak	Rechthoekig
	Type ligger en plaats in totale constructie	Uitkragende ligger; centrale positie (geen randligger)
	Mechanica-schema	Eenzijdig ingeklemde ligger met belastingen die aangrijpen in de knooppunten
	Vorm van de totale constructie	Blok met drie open en een gesloten zijde
Materiaal-gegevens	Materiaal_vakwerk	Staal
	Staalsoort	S355
	Staafprofiel	Buis met verschillende afmetingen (d en t)
	Productievorm	Warmgewalst
	Dakbedekking	Trapeziumvormige verzinkte staalplaat met gordingen
Overig	Locatie	Deventer
	Windgebied	3
	Omgeving	Urban

Bronnen: RTV Oost, 2015; Bartels, 2015 & MSP Dak en Wand, 2015

Figuur 17. Leo Halle Tribune in zijaanzicht (De Stentor, 2015)



Figuur 18. Vergelijkbaar ontwerp voor tribune en overkapping (De Architect, 2012)

3.2 Ontwerpmodule

In de ontwerpmodule wordt, zoals de naam als aangeeft, het ontwerp van de vakwerkligger opgesteld. In deze paragraaf wordt beschreven welke vakwerkconfiguraties in het computermodel worden verwerkt. Ook wordt toegelicht hoe het ontwerp van de vakwerkligger is geparametriseerd. Hierbij wordt uiteengezet hoe de plaats van de knooppunten bepaald wordt en hoe aangegeven wordt waar de staven zich bevinden. Ook wordt de wijze van nummering van de knopen en de elementen beschreven.

3.2.1 Vakwerkconfiguraties

Voor dit onderzoek wordt aangenomen dat de vakwerkligger die geoptimaliseerd dient te worden, zich gedraagt als een *zuiver vakwerk*. Deze term is in paragraaf 2.3.1 toegelicht. In die paragraaf zijn tevens drie configuraties voor vlakke vakwerkliggers geïntroduceerd. Deze drie configuraties, de Howe truss, de Pratt truss en de Warren truss, worden opgenomen in het computermodel. Deze configuraties zijn niet gekozen, omdat verwacht wordt dat dit de meest optimale configuraties zijn. Er is gekozen om de configuraties eenvoudig te houden, omdat de nadruk van dit onderzoek op het proces ligt. Zodoende is besloten om alleen vakwerkliggers te beschouwen waarvan de randstaven horizontaal en parallel lopen. De variatie wordt verkregen door verschillende toepassingen van de diagonalen en de staanders.

3.2.2 Parametrisatie van de vakwerkligger

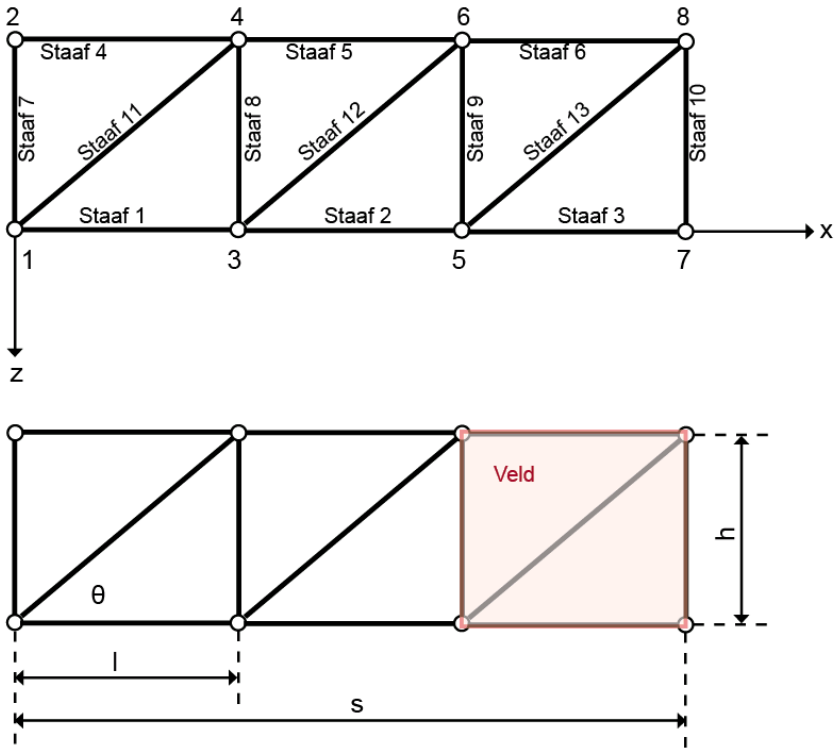
Om de verschillende configuraties te verwerken in het computermodel is het van belang de vakwerkligger te parametriseren. De plaats van de knooppunten wordt aangegeven door coördinaten in een xyz-assenstelsel. Vervolgens worden de staven als verbinding van twee knooppunten toegevoegd. Hierbij wordt gebruik gemaakt van het feit dat een vakwerkligger is opgebouwd uit *velden*, waarvan de configuratie zich herhaalt (zie Figuur 19 onder). Het aantal velden wordt in het computermodel verwerkt als een ontwerpvariabelen en staat dus niet vast. Dankzij het herhalende ritme van de velden is het mogelijk om met algoritmes de coördinaten van de knooppunten te bepalen en de staven toe te voegen. Aan de hand van een voorbeeld wordt dit verder toegelicht.

In Figuur 19 is een vakwerkligger weergegeven met een totale lengte s (span) en een hoogte h. De lengte staat vast en is onderdeel van de input van het model. De hoogte van de vakwerkligger is een ontwerpvariabele, waarvan de waarde door optimalisatie wordt bepaald. De vakwerkligger is een N-ligger met stijgende diagonalen, oftewel een vakwerkligger met configuratie Howe truss.

De voorbeeld vakwerkligger is opgebouwd uit 3 velden, ieder met een lengte l (l = s / aantal velden). De coördinaten van de knooppunten worden uitgedrukt in termen van de parameters h en l, waarbij knooppunt 1 zich altijd in de oorsprong van het assenstelsel bevindt. De knopen met een oneven nummer bevinden zich op de onderrandstaaf en die met even nummer op de bovenrandstaaf. De precieze coördinaten die horen bij dit voorbeeld, zijn weergeven in Tabel 5. De staven worden toegevoegd door aan te geven in welk knooppunt de staven starten en eindigen. Daarnaast krijgt iedere staaf ook een type mee. De typen die worden onderscheiden zijn: 1. Randstaven, 2. Staanders en 3. Diagonalen. De nummering van de staven verloopt per staaftype en van links naar rechts. Allereerst worden de onder- en bovenrandstaven genummerd. Gevolgd door de staanders en tot slot de diagonalen. Hier is voor gekozen, omdat er per staaftype sprake is van een ritme in het verloop van de start- en eindpunten van de staven. In

Tabel 6 is de informatie van de staven van de voorbeeld ligger weergegeven.

Voor meer inzicht in de wijze waarop de parametrisatie is uitgevoerd, wordt verwezen naar de code van het computermodel. Deze is toegevoegd aan dit rapport in bijlage F.



Figuur 19. Parametrisatie van een vakwerkligger

Tabel 5. Knooppunten van de voorbeeld vakwerkligger

Knoopnummer	X-coördinaat	Y- coördinaat	Z- coördinaat
1	0	0	0
2	0	0	h
3	L	0	0
4	L	0	h
5	2*L	0	0
6	2*L	0	h
7	3*L	0	0
8	3*L	0	h

Tabel 6. Staven van de voorbeeld vakwerkligger

Staafnummer	Startpunt	Eindpunt	Type
1	1	3	1
2	3	5	1
3	5	7	1
4	2	4	1
5	4	6	1
6	6	8	1
7	1	2	2
8	3	4	2
9	5	6	2
10	7	8	2
11	1	4	3
12	3	6	3
13	5	8	3

3.3 Module ter bepaling van de belastingen en de fundamentele combinaties

In dit hoofdstuk wordt de grootte van de belastingen die werken op de overkappingsconstructie bepaald. Hierbij wordt gebruik gemaakt van de formules die zijn behandeld in paragraaf 2.4. Er wordt toegelicht hoe de verschillende belastingen verwerkt worden in het computermodel en welke aannames en begrenzingen hierbij gehanteerd worden. Tot slot wordt beredeneerd welke FCs naar verwachting maatgevend zijn en opgenomen worden in het computermodel.

Bij de verdeling van de belastingen over de knooppunten wordt aangenomen dat de belastingen aangrijpen in de knooppunten van de bovenrandstaaf. Deze aanname is geldig zolang de dakbedekking ter plaatse van deze knooppunten verbonden is met de vakwerkligger.

Een uitzondering is de verdeling van de belasting ten gevolge van het eigen gewicht. Bij deze belasting is, wordt aangenomen dat het gewicht van een staaf zich verdeeld over de twee knooppunten die door de staaf met elkaar worden verbonden. Dit houdt in dat de helft van de staaflast afgedragen wordt naar het zogenaamde startpunt en de andere helft naar het eindpunt.

3.3.1 Constructieve betrouwbaarheid

In paragraaf 2.4.1 is besproken welke veiligheidsfactoren toegepast dienen te worden bij de controle van de ULS en de SLS. De veiligheidsfactoren voor toetsing van de SLS bedragen 1,0. Indien zowel de ULS en de SLS getoetst worden, zal het computermodel per ontwerp twee keer 3D Truss aan moeten roepen. Dit heeft als gevolg dat de rekentijd sterk toeneemt. Aangezien dit onderzoek in beperkte tijd uitgevoerd dient te worden, is het noodzakelijk dat de rekentijd beperkt blijft. Om deze reden is er gekozen gekozen om ook bij de controle van de SLS de veiligheidsfactoren van de ULS toe te passen. Dit betekent dat deze controle te streng uitgevoerd wordt. Door de eisen die horen bij de controle van de SLS, de stijfheidseisen, aan te passen wordt dit gecorrigeerd. Deze aanpak volstaat mits de eisen behorende bij de SLS niet maatgevend zijn. Dit dient dan ook gecontroleerd te worden. Indien deze eisen wel maatgevend blijken, dient de controle van de SLS opnieuw uitgevoerd te worden met de juiste veiligheidsfactoren. De aanpassing van de stijfheidseis wordt verder toegelicht in paragraaf 3.5.1.

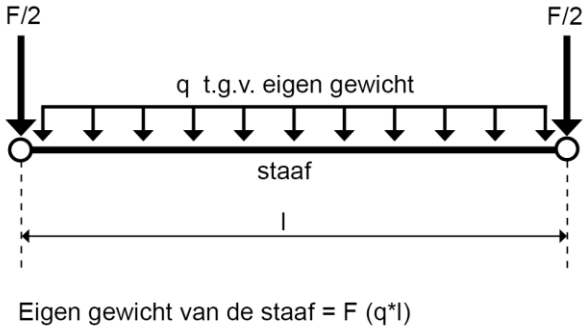
3.3.2 Permanente belastingen

De permanente belastingen die werken op de overkappingsconstructie zijn de rustende belasting van de dakbedekking en het eigen gewicht van het vakwerk, zoals besproken in paragraaf 2.4.2. Voor beide belastingen wordt hier toegelicht van welke waarden wordt uitgegaan bij het ontwikkelen van het computermodel.

Dakbedekking
Als dakbedekking voor de overkapping van de tribune worden verzinkte trapeziumvormige staalplaten toegepast. Om te zorgen dat de constructie zich gedraagt als zuiver vakwerk, is het van belang dat de krachten aangrijpen in de knooppunten. Om die reden is ervoor gekozen om gordingen toe te passen die bevestigd worden aan de vakwerkligger ter plaatse van de knooppunten. Het gewicht van de verzinkte staalplaat incl. gordingen en bevestigingen is: 0,15 kN/m^2 (Raaij et al., 2014).

Een belangrijk kenmerk van de trapeziumvormige staalplaten is de maximale overspanning van 6,0 meter, die deze staalplaten kunnen overbruggen (Raaij et al., 2014). Dit betekent dat de velden van de vakwerkligger maximaal 6,0 meter lang mogen zijn.

Eigen gewicht vakwerkligger
Het eigen gewicht van de vakwerkligger wordt berekend door het volume van iedere staaf te bepalen. Door dit vervolgens te vermenigvuldigen met het soortelijk gewicht van staal (zie paragraaf 2.4.2) wordt het eigen gewicht in kg gevonden. Dit kan gemakkelijk omgerekend worden naar de gewenste eenheid kN. De belasting ten gevolge van het eigen gewicht wordt per staaf afgedragen naar de twee knooppunten die door de staaf met elkaar worden verbonden (zie Figuur 20).



Figuur 20. Verdeling eigen gewicht over de knooppunten

3.3.3 Variabele belastingen

De variabele belastingen die van belang zijn bij het ontwerpen van de overkappingsconstructie zijn de wind- en de sneeuwbelasting en de belasting door goederen en personen. Ook deze drie belastingen zijn besproken in paragraaf 2.4.3, waarbij de formules voor het berekenen van de grootte van deze belastingen zijn toegelicht. Hieronder wordt per variabele belasting toegelicht welke waarden van de parameters uit deze formules beschouwd worden bij dit onderzoek.

Windbelasting
De grootte van de windbelasting is afhankelijk van enkele ontwerpgegevens. De locatie van het bouwwerk en de soort omgeving zijn van invloed, maar ook de hoogte van het gebouw speelt een rol. Deze ontwerpgegevens verschillen per ontwerpvraag en zijn dan ook vast onderdeel van de input van de gebruiker, zoals beschreven in paragraaf 3.1.

De formule waarmee de windbelasting berekend wordt, is behandeld in paragraaf 2.4.3. De waarden van de parameters die beschouwd worden bij dit onderzoek worden hier toegelicht.
De bouwwerkfactor $c_s c_d$ brengt het effect van de ongelijkheid van de windbelasting in rekening. Voor dit onderzoek wordt een enkele dakligger beschouwd. Daarom moet voor de bouwwerkfactor de waarde van 1,0 gebruikt worden. Het is namelijk een realistische mogelijkheid dat een windvlaag, en dus een extreme windbelasting, op het gedeelte van het dakvlak werkt dat zijn belasting afdraagt aan de beschouwde dakligger.

De dakligger die beschouwd wordt binnen dit onderzoek heeft een centrale positie in de totale dakconstructie. Zodoende zijn bij bepaling van de waarde van de krachtcoëfficient c_f alleen de zones A en C van het dakvlak van belang (zie paragraaf 2.4.3).

De waarde van de krachtcoëfficient c_f is ook afhankelijk van de dakhelling α. In het computermodel worden de krachtcoëfficienten opgenomen, die horen bij dakhellingen tussen de 0 en 10 graden en de zones A en C. De bijbehorende krachtcoëfficienten zijn weergegeven in Tabel 7. Voor het optimalisatievraagstuk van de vakwerkligger wordt een dakhelling van 0 graden aangehouden. De waarde van de dakhelling is niet variabel. Deze beperking is gerechtvaardigd, omdat de ligger altijd horizontaal is georiënteerd. Ontwerpen waarbij de helling van de ligger gelijk is

aan de dakhelling zijn vanwege de beperkte tijdsduur van dit onderzoek buiten beschouwing gelaten. Dit betekent dat de dakhelling alleen invloed heeft op de grote van de windbelasting. Hierbij geldt: hoe groter α, hoe groter de windbelasting. Er kan dan ook geconcludeerd worden dat een dakhelling van 0° in dit geval de voorkeur heeft. Indien bij eventueel vervolgonderzoek wel ontwerpen beschouwd worden waarbij de helling van de ligger gelijk is aan de dakhelling, is het van belang dat de dakhelling als variabele wordt meegenomen in het optimalisatievraagstuk.

Tabel 7. Krachtcoëfficient voor verschillende dakhellingen en voor blokkadefactor 1,0.

Dakhelling α	Blokkade factor φ	c_f voor zone	
		A	C
0°	Maximum all φ	+0,5	+1,1
	Minimum φ = 1	-1,5	-2,2
5°	Maximum all φ	+0,8	+1,3
	Minimum φ = 1	-1,6	-2,5
10°	Maximum all φ	+1,2	+1,6
	Minimum φ = 1	-2,1	-2,7

Bron: Raaij, B.P.M., Soons, F.A.M. & Wagemans, L.A.G., 2014.

De extreme stuwdruk is afhankelijk van de bouwwerkhoogte. Binnen dit onderzoek worden hoogtes van 10 m tot en met 20 m beschouwd. Er is gekozen om slechts een beperkte range van bouwwerkhoogtes te verwerken in het computermodel, omdat de onderzoekstijd zeer beperkt is. Daarnaast wordt het implementeren van de overige, reeds bekende, data niet gezien als waardevol. Er wordt echter wel voor gezorgd dat het model gemakkelijk uitgebreid kan worden met deze informatie als dit voor toekomstig gebruik gewenst is.

Het referentie oppervlak dat hoort bij de beschouwde dakligger heeft een grootte van s * a m^2 (overspanning * h.o.h.-afstand). De grootte van de windbelasting is echter niet voor dit gehele referentie oppervlak gelijk vanwege de aanwezigheid van de verschillende windzones van het dakvlak. Dit maakt de verdeling van de windbelasting complexer dan die van de overige belastingen. Het computermodel houdt rekening met de aanwezigheid van de windzones en zorgt ervoor dat de windbelasting op de juiste wijze wordt verdeeld over de knooppunten. De validatie van deze code wordt besproken in paragraaf 0.

Sneeuwbelasting

De grootte van de sneeuwbelasting wordt berekend met de formule, die beschreven is in paragraaf 2.4.3. De enige parameter uit deze formule, waarvan de waarde nog niet bepaald is, is de vormcoëfficient. Deze is afhankelijk van de dakhelling α en de mogelijkheid tot ophoping van sneeuw. Voor dit onderzoek wordt een horizontale vakwerkligger beschouwd. Hierbij wordt aangenomen dat de verticale constructie die de oplegging vormt voor deze ligger niet boven de dakconstructie uitsteekt. De vorm van het dak biedt zodoende geen mogelijkheid tot ophoping van sneeuw. Verder is onder het kopje *windbelasting* reeds besproken dat de overkappingsconstructie een maximale dakhelling van 10 graden kan aannemen. Dit betekent dat vormcoëfficient en de bijbehorende waarde die gebruikt dient te worden $\mu_1 = 0{,}8$ is (zie Figuur 14 in paragraaf 2.4.3).

Belasting door goederen & personen

In paragraaf 2.4.3 is beschreven dat de grootte van deze belastingen bij daken afhankelijk is van de toegankelijkheid van het dak en van de dakhelling α. Binnen dit onderzoek worden alleen daken beschouwd die niet toegankelijk zijn en waarbij de dakhelling begrensd is en ligt tussen de 0 en 10 graden.

De grootte van de twee vormen van de belasting door goederen & personen is in dit geval:

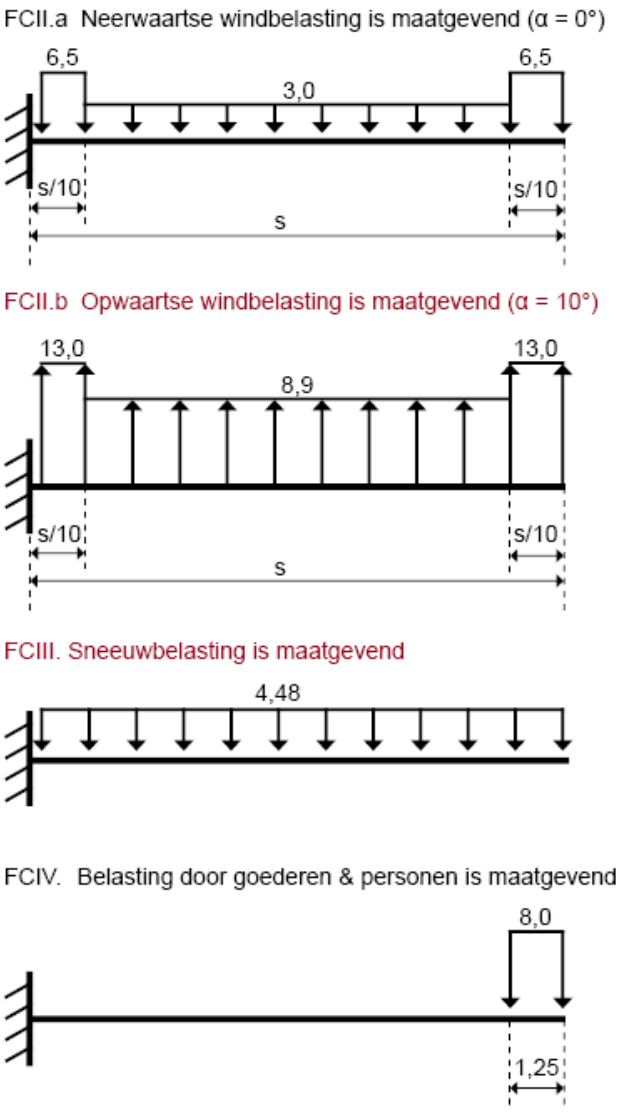
$q_k = 1{,}0$ kN/m^2 en $Q_k = 1{,}5$ kN

3.3.4 Bepaling maatgevende fundamentele combinaties

De fundamentele combinaties zijn opgesteld in paragraaf 2.4.4. Per FC treden verschillende staafkrachten op. Zodoende ontstaan er ook verschillende sets van Unity Checks (zie paragraaf 2.5). Voor alle sets geldt dat de UCs niet groter dan 1,0 mogen zijn. Deze eis wordt in het computermodel opgenomen door de FCs te verwerken als constraints, oftewel beperkingen van het ontwerpgebied. Iedere FC levert een set constraints waaraan het optimale ontwerp dient te voldoen.

Voor dit onderzoek worden vanwege de beperkte tijd niet alle FCs behandeld. Om aan te tonen dat het computermodel om kan gaan met verschillende belastingsituaties worden alleen de FCs, die naar verwachting maatgevend zijn voor het optimalisatievraagstuk van dit onderzoek, verwerkt in het model.

De verschillende belastingcombinaties worden met elkaar vergeleken aan de hand van de geschematiseerde weergave in Figuur 21. De wijze waarop de waarden van de belastingen bepaald zijn, is terug te vinden bijlage A. De permanente belasting is bij de vergelijking van de FCs buiten beschouwing gelaten, omdat deze belasting bij iedere FC gelijk is. Daarbij kan vooraf gemakkelijk opgemerkt worden dat de FC waarbij de permanente belasting maatgevend is gekozen, niet maatgevend zal zijn voor het ontwerp van de vakwerkligger. Dit kan opgemerkt worden door het lage gewicht van de dakbedekking en het lage eigen gewicht van de staven van het vakwerk.



Figuur 21. Fundamentele combinaties geschematiseerd

De visualisatie van de belastingcombinaties (zie Figuur 21) maakt inzichtelijk dat FCII.b en FCIII naar verwachting de maatgevende FCs zijn. Deze twee FCs worden verwerkt in het computermodel en beschouwd bij het optimalisatievraagstuk van de vakwerkligger.

De maatgevende FCs zijn:

FCII.b Opwaartse windbelasting is maatgevend

0,9 * G ‘+’ 1,65 * $Q_{wind,op}$

FCIII Sneeuwbelasting is maatgevend

1,3 * G ‘+’ 1,65 * Q_{sneeuw}

3.3.5 Belastingsvormen en bijbehorende parameters

De grootte van de belastingen is afhankelijk van verschillende variabelen, zoals de locatie van het bouwwerk, de dakhelling of de onderlinge hart-op-hart afstand tussen de vakwerkliggers die samen de gehele overkappingsconstructie vormen. In Tabel 8 is aangegeven welke variabelen van invloed zijn op de grootte van de belastingen.

Tabel 8. Belastingen en parameters die van invloed zijn op de grootte

Belasting	Afhankelijk van:	Toegepaste waarden voor dit onderzoek:
Dakbedekking	Gewicht per m^2 Overspanning s H.o.h.-afstand a	Golfplaat: 0,15 kN/m^2 S = 21,0 m a = 8,3 m
Eigen gewicht	Soortelijk gewicht staal Per staaftype: Oppervlak A Lengte l	78,5 kN/m^3 A en l zijn variabel
Windbelasting	Windgebied 1, 2, 3 Soort gebied rural, urban, coastal Bouwwerkhoogte z_e Windrichting Dakhelling α	Windgebied 3 (Deventer) Soort gebied = urban (stedelijk) z_e = 13,5 + hoogte vakwerk (h) Tegen dichte gevel (neerwaarts) of vanaf open zijde (opwaarts). α = 0°
Sneeuwbelasting	Sneeuwbelasting per m^2 Dakhelling α Vorm van het dak i.v.m. kans op ophoping van sneeuw	0,56 kN/m^2 α = 0° geen ophoping mogelijk → μ_1 = 0,8
Belasting door goederen & personen	q_k Q_k Functie Toegankelijkheid Dakhelling α	1,0 kN/m^2 1,5 kN Dak Niet toegankelijk α = 0° á 10°

3.4 Rekenmodule

In paragraaf 2.2.3 is besproken dat het programma 3D Truss als rekenkern voor het computermodel wordt gebruikt. De werking van het programma is reeds besproken in deze paragraaf. Ook is aangegeven dat voor de koppeling van Python en 3D Truss gebruik gemaakt wordt van de eindrapporten van Hoogendoorn (2015) en Koning (2015). Hierbij is aangegeven dat de manier waarop Koning 3D Truss runt in batch mode twee nadelen met zich meebrengt. Ten eerste is het niet zeker dat 3D Truss na de wachttijd klaar is met het wegschrijven van de resultaten naar de outputfile. Als 3D Truss er, om wat voor reden dan ook, langer over doet dan de geschatte tijd, klapt het programma eruit. Ten tweede is het goed mogelijk dat 3D Truss minder tijd nodig heeft dan geschat wordt. Met name indien het aantal variabelen beperkt is, zal 3D Truss (veel) sneller klaar zijn. Dit betekent dat de snelheid van het computermodel onnodig vertraagd wordt.

Voor dit onderzoek zijn bovenstaande nadelen zeer ongewenst, omdat een hoge robuustheid en rekensnelheid van grote waarde zijn voor het computermodel. Om die reden is er een alternatief gezocht voor de manier waarop 3D Truss aangestuurd wordt. Bij deze alternatieve wijze van batch mode running wordt het commando '*call*' gebruikt in plaats van '*Popen*'. Hierbij wacht Python tot 3D Truss klaar is en gaat dan pas verder met het runnen van de rest van de code. Op die manier krijgt 3D Truss exact de juiste hoeveelheid tijd die het nodig heeft. Er is geen risico dat het computermodel eruit klapt en er hoeft niet onnodig lang gewacht te worden. Dit resulteert in een hogere robuustheid en hogere rekensnelheid van het model.

3.5 Toetsingsmodule

Er moeten een aantal toetsen uitgevoerd worden om te controleren of de vakwerkligger voldoet aan alle eisen. De eisen die gesteld worden aan het ontwerp van de vakwerkligger, zijn beschreven in paragraaf 2.5. De maximale waarde van de Unity Checks (UCs) is 1,0 . Indien de waarde van een UC ver onder de 1,0 ligt is er sprake van overdimenisonering en is er wellicht ruimte voor optimalisatie.

De volgende controles worden opgenomen in het computermodel:

- Controle van de normaalkracht
- Controle van de knikstabiliteit
- Controle van de verticale verplaatsingen
- Controle van de verbindingen

De staafkrachten die optreden verschillen per fundamentele belastingcombinatie. Er worden twee FCs opgenomen in het computermodel. Voor beide FCs dienen alle bovenstaande controles uitgevoerd te worden. Deze eisen worden in computermodel verwerkt als zogenoemde *constraints*.

3.5.1 Aanpassing stijfheidseis

De verplaatsingen worden binnen dit onderzoek berekend mét toepassing van de partiële factoren aan de belastingkant, zoals is toegelicht in paragraaf 3.3.1. Zodoende dient de stijfheidseis die beschreven is in paragraaf 2.5.3 aangepast te worden. Hierbij wordt gebruik gemaakt van een *gewogen veiligheidsfactor*, die afhankelijk is van de verhouding tussen de permanente en de variabele belasting. Deze verhouding verschilt per fundamentele belastingcombinatie. De wijze waarop de grootte van de gewogen veiligheidsfactor berekend is, is terug te vinden in bijlage B.

De gewogen veiligheidsfactor die opgenomen wordt in het computermodel heeft een waarde van 1,56. De stijfheidseis wordt hiermee:

Aangepaste verticale verplaatsingseis: $w_{aangepast} \leq 1{,}56 * 0{,}004 * L_{rep}$

$w_{aangepast} \leq 0{,}01248 * s$

4 Optimalisatie

Bij dit onderzoek wordt het ontwerp van de vakwerkligger geoptimaliseerd naar kosten. In paragraaf 1 wordt de *objective function*, de functie die de totale kosten berekent, beschreven.

In paragraaf 2 wordt de meest geschikte optimalisatie aanpak voor het optimalisatievraagstuk van dit onderzoek gekozen. Hierna wordt in paragraaf 3 beschreven welke bounds en startwaarden van de ontwerpvariabelen gehanteerd worden en hoe deze zijn bepaald.

4.1 Objective function

De objective function is de functie die de te optimaliseren grootheid berekent. Bij dit onderzoek zijn dit de totale kosten van de vakwerkligger. Er zijn verschillende kostenposten die bijdragen aan de totale prijs van de ligger. Bij dit onderzoek worden de staafkosten en de kosten van de verbindingen beschouwd. Andere kostenposten worden buiten beschouwing gelaten.

In deze paragraaf wordt van de staaf- en de verbindingskosten beschreven waar de kosten van afhangen. Ook wordt per onderdeel aangegeven welke prijs bij dit onderzoek aangenomen wordt. Deze prijzen zijn slechts ter indicatie. Ze kunnen per aannemer sterk verschillen. Zodoende moeten bij het gebruik van dit model de prijzen geanalyseerd en eventueel opnieuw vastgesteld worden.

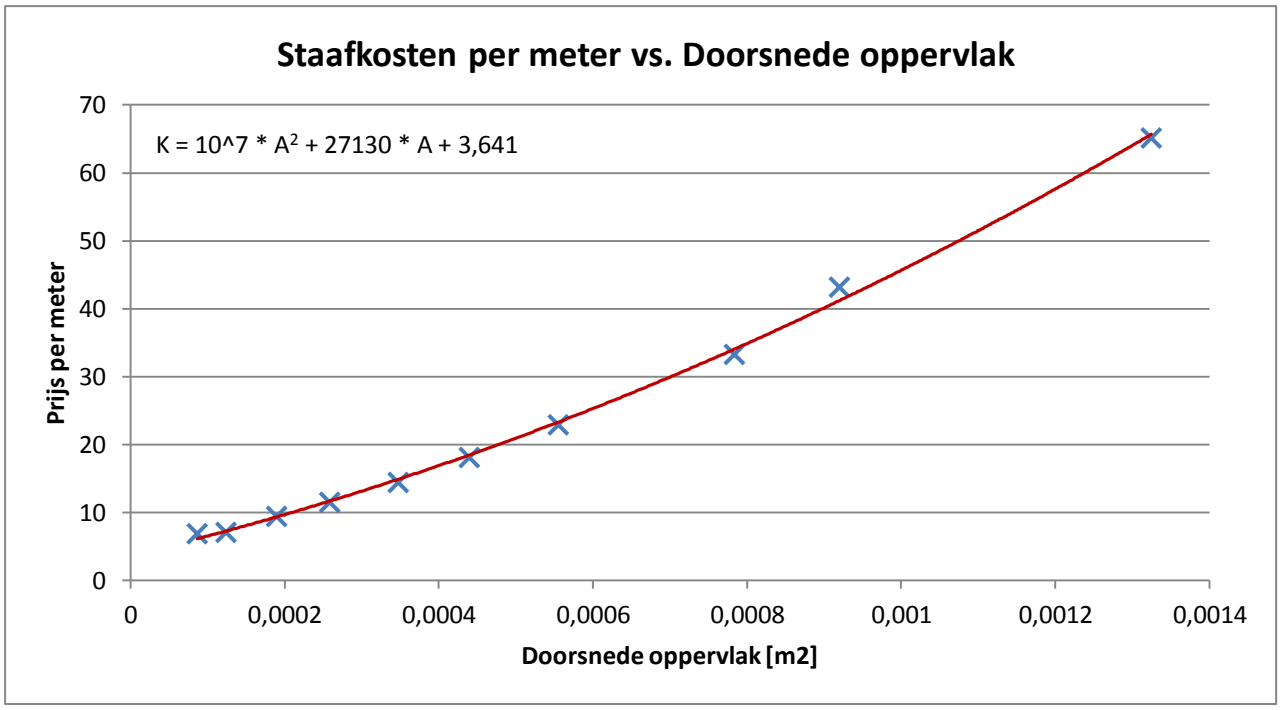
4.1.1 Staafkosten

De kosten van de staven zijn afhankelijk van het doorsnede oppervlak en de bijbehorende prijs per meter. Het is dan ook gewenst om een formule te bepalen waarmee de kosten per meter staaf berekend kunnen worden aan de hand van het doorsnede oppervlak. In het eindrapport van Koning wordt beschreven hoe zo'n formule afgeleid kan worden (2015, p. 15). Koning geeft aan dat bij deze afleiding gebruikt gemaakt is van de kosten per kg, zoals deze gepubliceerd zijn op *staalprijzen.nl* (Staalprijzen, 2015). De kosten die op deze site gevonden werden, kwamen echter niet overeen met de datapunten uit de grafiek van de Koning. De correctheid van de formule voor de staafkosten, die Koning afleid in haar rapport, kon daardoor niet gecontroleerd worden. Daarom is gekozen om deze afleiding nogmaals uit te voeren met de informatie van de site *staalprijzen.nl*. De datapunten en de bijbehorende trendline zijn weergegeven in Grafiek 1. De formule van deze trendline bleek inderdaad niet overeen te komen met de formule van Koning. Voor dit onderzoek wordt gebruik gemaakt van het verband tussen het doorsnede oppervlak van de staven en de prijs per meter, dat gevonden is met behulp van de trendline uit Grafiek 1. De formule die hierbij hoort en die verwerkt wordt in het computermodel is:

$$\mathbf{K_{staaf}} = \mathbf{10^7} * \mathbf{A^2} + \mathbf{27130} * \mathbf{A} + \mathbf{3,641}$$

Waarbij: K_{staaf} = kosten van de staaf [€/m]
A = doorsnede oppervlak van de staaf [m^2]

Grafiek 1. Plot van de staafkosten uitgezet tegen het doorsnede oppervlak incl. trendline



4.1.2 Kosten van de verbindingen

Bij dit onderzoek wordt onderscheid gemaakt tussen twee soorten verbindingen, namelijk lasverbindingen en systeemverbindingen. De kenmerken van deze twee verbindingsvormen zijn besproken in paragraaf 2.3.4. Per verbindingsvorm wordt hieronder toegelicht waar de kosten van afhankelijk zijn en hoe ze binnen dit onderzoek worden berekend.

Lasverbindingen

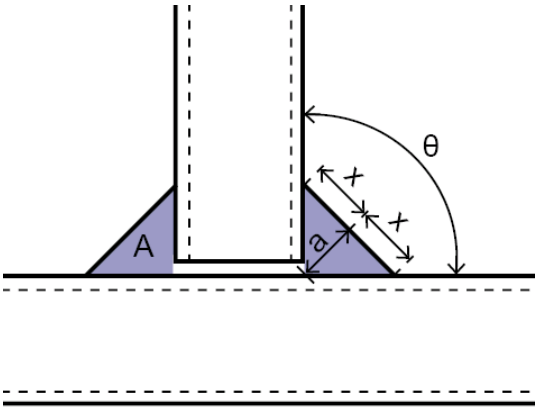
Binnen dit onderzoek wordt aangenomen dat de randstaven bij toepassing van lasverbindingen, over de hele lengte van de ligger doorlopen. Deze aanname is voor dit onderzoek gerechtvaardigd, aangezien de lengte van de vakwerkligger die geoptimaliseerd wordt de maximale lengte met betrekking tot transport niet overschrijdt. Deze ligger kan dus in zijn geheel vervoerd worden en alle verbindingen kunnen in de fabriek vervaardigd worden. Voor dit onderzoek wordt verder geen aandacht besteed aan liggers met lengtes boven de maximum transportlengte, die in delen vervoerd zouden moeten worden. Het wordt echter wel aangeraden om bij een eventueel vervolgproject te onderzoeken welke invloed het splitsen van de ligger zou hebben op de kosten.

Bij de beschrijving van lasverbindingen in paragraaf 2.3.4 zijn zes knooptypen gedefinieerd. In het computermodel worden drie configuraties voor de vakwerkligger opgenomen, zoals besproken in paragraaf 3.2.1. Niet alle zes de knooptypen komen voor bij deze drie configuraties. Daarom worden ook niet alle knooptypen in het computermodel verwerkt. De knooptypen die wel in het computermodel worden opgenomen, zijn de T-, K- en N-knopen.

De kosten voor de verbindingen worden per knooptype bepaald. De prijs wordt hierbij afhankelijk gesteld van het volume van de lasnaad dat nodig is om de staven te verbinden. Er wordt aangenomen dat het volume van de lasnaad evenredig is aan het aantal manuren dat nodig is om de las te leggen.

Het volume van de lasnaad wordt berekend door de omtrek van de te verbinden staaf te vermenigvuldigen met de dwarsdoorsnede van de las. Hierbij wordt de dikte van de las aangeduid met de keeldoorsnede a, die afhankelijk is van de wanddikte t van de staaf. In Figuur 22 is een voorbeeld weergegeven van een lasverbinding tussen een staander en een randstaf met de bijbehorende parameters.

Het doorsnedeoppervlak A valt niet te bepalen zonder informatie over de hoek tussen de te verbinden staven en informatie over eventuele voorbewerking. Bij dit onderzoek wordt ter vereenvoudiging aangenomen dat geldt: x = a.



Figuur 22. Lasdoorsnede incl. parameters

De benodigde afmeting van keeldoorsnede a wordt als volgt bepaald (Abspoel et al., 2013):

$$2 * a * l_{las} * f_{w,u;d} \geq t * l_{las} * f_{y;d}$$

Voor staalsoort S355:

$$2 * a * 510 \geq t * 355$$
$$a \geq t * 0,35$$

De volgende berekeningen zijn gebruikt om de kosten van de lasverbindingen te bepalen:

$$A_{las} = 2 * (\frac{1}{2} * a * x) = a^2 = (t * 0,35)^2$$
$$K_{las} = P_{las} * l_{las} * A_{las} = O_{staaf} * (t * 0,35)^2$$

Waarbij: P_{las} = prijs van de lasnaad [€/mm^3] = € 0,15* per mm^3

**Deze prijs is slechts bedoeld ter indicatie van de kosten. De schatting is bepaald op basis van gegevens uit het eindrapport van L. Koning (2015, p. 16).*

Systeemverbindingen

Bij het vaststellen van de kosten van systeemverbindingen wordt, net als bij lasverbindingen, gebruik gemaakt van een classificatie van knooptypen. De classificatie die hier gehanteerd wordt is echter minder uitgebreid dan die van de knooptypen bij lasverbindingen. Er wordt slechts gekeken naar het aantal staven dat door de systeemverbinding met elkaar verbonden moet worden. Een belangrijk verschil ten opzichte van de lasverbindingen is dat de randstaven in dit geval geen doorlopende staven zijn, maar een lengte hebben die gelijk is aan de lengte van de velden.

Bij het opstellen van de kosten van systeemverbindingen worden de kosten van het benodigde materiaal buiten beschouwing gelaten. Deze keuze is gemaakt, omdat de kosten slechts ter indicatie dienen om het optimalisatieproces te kunnen onderzoeken. Een gedetailleerde beschrijving van de kosten is hiervoor niet nodig.

Classificatie van de knooptypen naar aantal staven dat samenkomt en een schatting* van de bijbehorende kosten:

- 2 staven 50 euro
- 3 staven 100 euro
- 4 staven 150 euro

** De schatting van de kosten is gemaakt op basis van gegevens uit het eindrapport van L. Koning (2015, p. 16).*

4.2 Keuze optimalisatie aanpak

In deze paragraaf worden allereerst de kenmerken toegelicht van het optimalisatievraagstuk van dit onderzoek. Hierbij wordt beschreven naar welke ontwerpvariabelen gekeken wordt. Ook wordt beschreven welke eisen er aan de optimalisatie aanpak gesteld worden. Op basis van de kenmerken en eisen wordt vervolgens beredeneerd welke optimalisatie aanpak het meest geschikt is en in het computermodel geïmplementeerd zal worden.

Er zijn 3 discrete ontwerpvariabelen te benoemen, die een rol spelen bij het optimalisatievraagstuk van de vakwerkligger. Dit zijn de configuratie, het aantal velden waar de vakwerkligger uit opgebouwd wordt en het type verbinding dat wordt toegepast. Er worden drie configuraties (zie paragraaf 3.2.1) en twee verbindingstypen (zie paragraaf 2.3.4) bekeken. Naast het drietal discrete variabelen worden er nog 5 continue variabelen beschouwd. Dit zijn de staafafmetingen van de randstaven en de wandstaven en de hoogte van de vakwerkligger.Het optimalisatievraagstuk van de vakwerkligger is dus een probleem met zowel continue als discrete ontwerpvariabelen. Dit wordt een mixed continuous-discrete problem genoemd.

Het beschouwde optimalisatievraagstuk is een probleem met veel ontwerpvrijheid, doordat er een groot aantal ontwerpen mogelijk is. De mogelijke oplossingen van het probleem lopen gemakkelijk tot in de duizenden bij combinatie van alle 8 ontwerpvariabelen. Stel dat er voor iedere ontwerpvariabele 5 mogelijkheden zijn, dan resulteert dit in 5^8 = 390625 mogelijke uitkomsten. Gezien de rekentijd van de rekenkern 3D Truss voor vergelijkbare ontwerpen (Koning, 2015 p.12) kan het erg lang duren om al deze mogelijke ontwerpen door te rekenen. Het is dan ook niet gewenst om de brute force aanpak te gebruiken, zeker niet als er bij een vervolgproject meer uitbreidingen worden toegevoegd aan het computermodel, zoals extra profielen, configuraties, belastingcombinaties, etc..

Gezien de beperkte tijd die beschikbaar is voor dit onderzoek, is het noodzakelijk om een optimalisatie aanpak toe te passen die in relatief korte tijd het optimum kan bepalen. Als uitgangspunt zou het gewenst zijn om in een paar uur een optimum te kunnen vaststellen. Op die manier kan het computermodel meerdere malen per dag gerund worden. Dit biedt de mogelijkheid om op dezelfde dag de resultaten te analyseren, eventuele bugs uit de code te halen en het model opnieuw te runnen. Ook kunnen de instellingen van het optimalisatie algoritme meerdere malen aangepast en getest worden zonder dat hier veel tijd voor nodig is.

In paragraaf 2.6 zijn vier optimalisatie aanpakken met elkaar vergeleken. Uit deze vergelijking bleek dat continue aanpakken over het algemeen sneller zijn in het bepalen van het optimum dan discrete. Hier staat tegenover dat bij continue aanpakken het risico aanwezig is dat er een lokaal optimum wordt gevonden in plaats van het globale optimum. In paragraaf 2.6 zijn ook twee optimalisatie algoritmen, een continue en een discrete, met elkaar vergeleken. Bij deze vergelijking is aangetoond dat het continue optimalisatie algoritme beter te sturen is. Ook is vastgesteld dat aan het discrete algoritme geen constraints meegegeven kunnen worden. Dit betekent dat bij toepassing van dit algoritme, de toetsing van het ontwerp los geïntegreerd moet worden in het computermodel.

Met oog op de beperkte tijdsduur van dit onderzoek is beredeneerd dat de voordelen van een continue aanpak zwaarder wegen dat het nadeel met betrekking tot het vinden van een lokaal optimum. Er is dan ook gekozen de gradient-based optimization method toe te passen. Aangezien deze aanpak alleen geschikt is voor continue ontwerpvariabelen en er bij het optimalisatievraagstuk van dit onderzoek ook sprake is van discrete ontwerpvariabelen, dient het vraagstuk opgesplitst te worden. Het gemengde vraagstuk wordt opgesplitst in een continu hoofdprobleem en een discreet subprobleem. Voor het discrete subprobleem wordt de brute force approach gehanteerd. Deze aanpak is haalbaar in beperkte tijd, vanwege het geringe aantal discrete ontwerpvariabelen.

Het continue hoofdprobleem en het discrete subprobleem worden gekoppeld door toepassing van een aantal for-loops. Deze for-loops doorlopen alle mogelijke combinaties van de

discrete ontwerpvariabelen. Iedere combinatie wordt doorgegeven aan het continue hoofdprobleem dat vervolgens de optimalisatie van de continu ontwerpvariabelen uitvoert.

Indien het model uitgebreid wordt en er meer discrete variabelen worden toegevoegd of de rekentijd sterk toeneemt, zal wellicht een andere aanpak gekozen moeten worden. Bij een eventueel vervolgproject moet hier rekening mee gehouden worden.

4.3 De ontwerpvariabelen: bounds en startwaarden

De ontwerpvariabelen die een rol spelen bij het optimalisatievraagstuk van dit onderzoek worden begrensd, zoals is toegelicht in paragraaf 1.2. Daardoor wordt de ontwerpruimte, de ruimte die alle mogelijke oplossingen bevat, verkleind. Dit maakt het mogelijk om in de beperkte tijd die voor dit onderzoek beschikbaar is, tot een uitkomst van het optimalisatievraagstuk te komen en het proces te onderzoeken.

In deze paragraaf wordt toegelicht hoe de boven- en ondergrenzen, ofwel de bounds van de ontwerpvariabelen opgesteld zijn. Ook wordt beschreven welke startwaarden gehanteerd worden voor de continue ontwerpvariabelen. In Tabel 9 is een overzicht weergegeven van de ontwerpvariabelen en de bijbehorende bounds en startwaarden, zoals deze verwerkt zijn in het computermodel.

Tabel 9. Ontwerpvariabelen die variëren binnen het optimalisatievraagstuk

	Ontwerpvariabelen	Bounds	Startwaarden	Eenheid
Discreet	Configuratie	'Howe', 'Pratt', 'Warren'	-	-
	Aantal velden	4 ≤ nfields ≤ 20	-	-
	Type verbinding	'welded, 'system'	-	-
Continu	Diameter randstaven	114 – 570	380	mm
	Wanddikte randstaven	4,8 – 24,0	16	mm
	Diameter wandstaven	78 – 390	260	mm
	Wanddikte wandstaven	2,4 – 12,0	8	mm
	Hoogte van de vakwerkligger	s/20 ≤ h ≤ s/5 Voor s = 21,0: 1,05 ≤ h ≤ 4,2	1,5	m

De upperbound van het aantal velden waaruit de vakwerkligger wordt opgebouwd, is opgesteld i.v.m. de verdeling van de windbelasting. Het aantal knopen waarover de windbelasting verdeeld wordt, hangt af van het aantal velden waaruit de vakwerkligger wordt opgebouwd. De verdeling aan de uiteinden van de ligger wordt bepaald door de grootte van de randzones van het dak ten opzichte van de lengte van de velden. Om de tijd die nodig is om aan te tonen dat het computermodel om kan gaan met verschillende verdelingen van de windbelasting, te beperken, is de upperbound van 20 velden opgesteld. De lowerbound van deze ontwerpvariabele is nodig om te voorkomen dat overspanning van de dakbedekking te groot wordt. De afstand die de dakplaten moeten overspanning is gelijk aan de afstand tussen de knopen van de randstaven. Bij dit onderzoek worden trapeziumvormige verzinkte staalplaten toegepast als dakbedekking. Deze staalplaten hebben een maximale overspanning van 6,0 m. Een vakwerkligger met een totale lengte van 21,0 m dient zodoende uit minimaal 4 velden te bestaan (21,0 / 6,0 ≈ 4).

Voor het type verbinding dat wordt toegepast wordt onderscheid gemaakt tussen lasverbindingen en systeemverbindingen. De kenmerken van deze verbindingsvormen en de kosten die hieraan gekoppeld zijn, zijn besproken in paragraaf 2.3.4 respectievelijk 4.1.2.

De bounds van de staafafmetingen zijn opgesteld aan de hand van een *trial and error approach.* De eerste schatting van de bounds is gedaan op basis van een handberekening, waarbij

het benodigde doorsnede oppervlak is uitgerekend. De bounds zijn aangepast door deze mee te geven aan de optimalisatiefunctie en de uitkomst te analyseren. Indien de optimalisatiefunctie tegen de bounds aanliep of binnen de meegegeven bounds niet aan de eisen kon voldoen, zijn de bounds bijgesteld. De bounds die uiteindelijk zijn aangehouden, zijn weergegeven in Tabel 9.

De upperbound van de hoogte van de vakwerkligger komt voort uit de afbakening van de bouwwerkhoogte i.v.m. de bepaling van de windbelasting. De maximale bouwwerkhoogte is 20,0 m. Met een vrije hoogte van 13,5 m (zie ontwerpgegevens in paragraaf 3.1) betekent dit dat de vakwerkligger maximaal 6,5 m hoog mag zijn (20,0 – 13,5 = 6,5 m). Deze upperbound wordt bijgesteld met oog op de slankheid van de ligger. Ook de lowerbound van de hoogte van de ligger is vanuit dit perspectief bepaald. De lowerbound is nodig ter beperking van de slankheid van de ligger. De lower- en upperbound die gehanteerd worden zijn: s/20 respectievelijk s/5 m. Voor dit onderzoek, waarbij s= 21,0m resulteert dit in een lower- en upperbound van: 1,05 respectievelijk 4,2 m.

De startwaarden van de continue ontwerpvariabelen zijn, net als de bounds van de staafafmetingen, opgesteld aan de hand van een trial and error approach. Hierbij is naar voren gekomen dat het van belang is dat het ontwerp dat gevormd wordt door de combinatie van de startwaarden, het startpunt genoemd, voldoet aan de constraints. Een startpunt dat voldoet aan de constraints wordt een *feasible startpoint* genoemd.

5 Programmatuur beschrijving

In dit hoofdstuk wordt beschreven hoe het computermodel in elkaar zit. De aannames en begrenzingen die gehanteerd zijn bij het ontwikkelen van het model worden in paragraaf 1 benoemd.

Aan de hand van een drietal *flow diagrams* wordt vervolgens de werking van het computermodel uitgelegd in paragrafen 2 t/m 4. De stappen die doorlopen worden en de keuzes die hierbij gemaakt moeten worden, worden beschreven.

Het hoofdstuk eindigt met de validatie van het model in paragraaf 5.

5.1 Aannames en Begrenzingen

Bij het ontwikkelen van het computermodel is gebruik gemaakt van aannames en begrenzingen om het onderzoeksgebied af te bakenen. Op die manier wordt gezorgd dat het onderzoek binnen de beperkte tijdsduur afgerond kan worden. De aannames en begrenzingen die gehanteerd zijn, zijn hieronder in de vorm van een opsomming weergegeven.

<u>Aannames:</u>

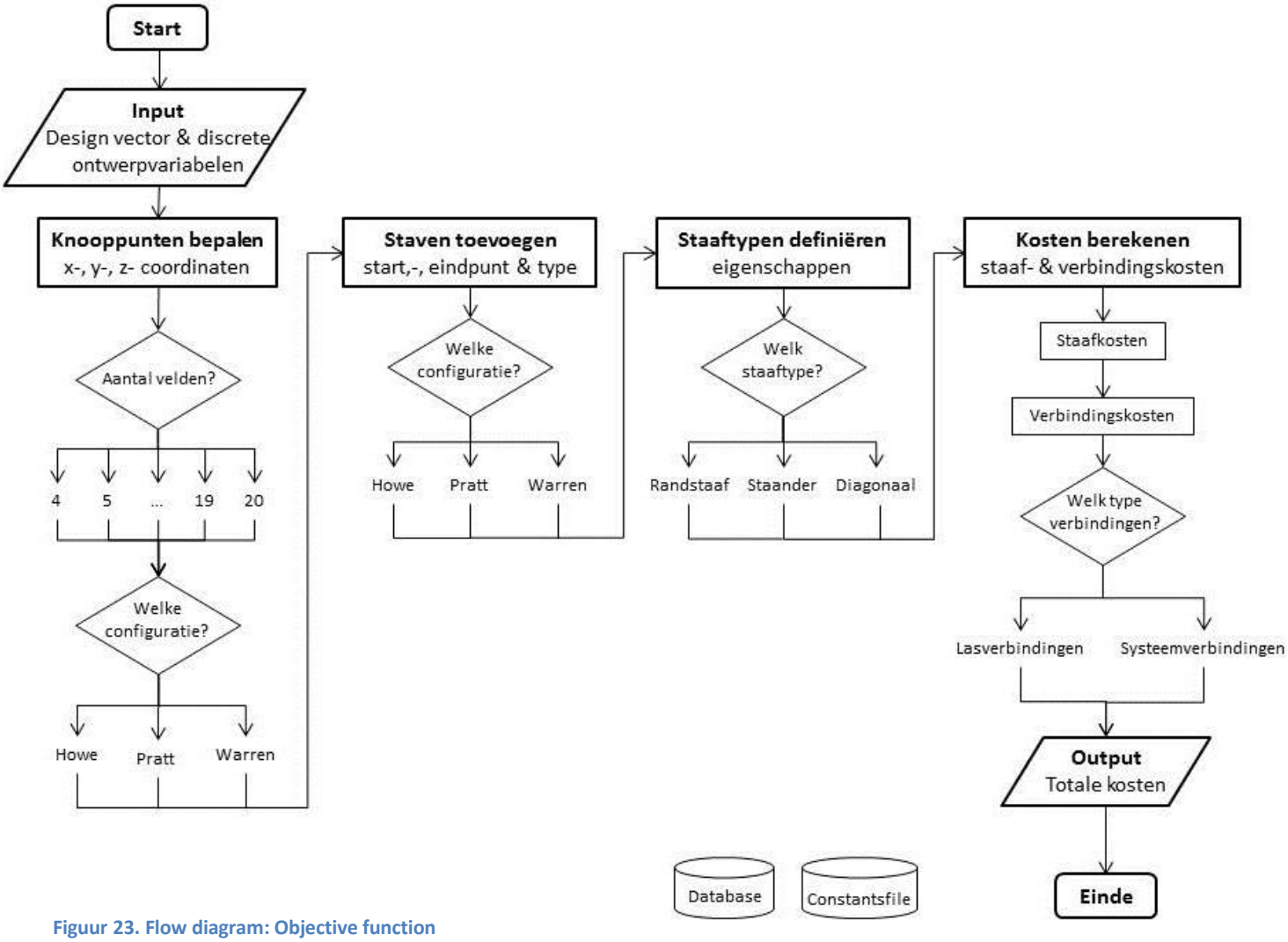
- Stabiliteit van dakconstructie in zijn geheel wordt geregeld.
- Randstaven van de vakwerkligger worden in de knopen zijwaartse ondersteund.
- Constructie gedraagt zich als een zuiver vakwerk.
- Alle belastingen werking in verticale (z-)richting.
- Er is alleen sprake van statische belasting (dynamische belasting wordt niet beschouwd).
- De belasting grijpt aan in de knooppunten van de bovenrandstaaf (met uitzondering van de belasting door het eigen gewicht).
- Verplaatsingen zijn niet maatgevend; berekeningen worden uitgevoerd mét veiligheidsfactoren (zowel SLS als ULS), waarbij de stijfheidseis wordt aangepast.
- Bouwwerkfactor $c_s c_d$, die gebruikt wordt om de windbelasting te bepalen, is 1,0.
- Bij lasverbindingen lopen de randstaven door over de hele lengte van de ligger.

<u>Begrenzingen:</u>

- Alleen vlakke vakwerkliggers worden bekeken.
- De randstaven van de vakwerkliggers lopen horizontaal en parallel.
- Het betreft bouwwerken in Nederland.
- Dak is niet toegankelijk.
- Ophoping van sneeuw is niet mogelijk, dus vormcoëfficient μ_1 kan gebruikt worden.
- Totale hoogte van het bouwwerk ligt tussen de 10 en 20 meter.
- De dakhelling is 0 graden.
- Blokkadefactor is 1,0.
- Er worden 3 configuraties beschouwd: Howe, Pratt en Warren truss.
- Het maximum aantal velden van de vakwerkligger is 20.
- Het minimum aantal velden van de vakwerkligger is 4.
- Er worden 2 staafgroepen met eigen afmetingen behandeld: wandstaven en randstaven.
- Er wordt 1 type profiel beschouwd: buisprofiel.
- Er wordt 1 staalsoort beschouwd: S355.
- Er worden 2 belastingcombinaties opgenomen in het model: opwaartse windbelasting is maatgevend en sneeuwbelasting is maatgevend.
- Er worden geen eisen gesteld met betrekking tot de overlap of de gap van verbindingen (alleen eisen met betrekking tot de verhouding van staafafmetingen).

5.2 Objective function

Het flow diagram in Figuur 23 maakt inzichtelijk welke input de *objective function* nodig heeft en welke stappen worden doorlopen om de output te genereren. De objective function geeft als output de te optimaliseren grootheid, het *object*. Binnen dit onderzoek wordt het ontwerp van de vakwerkligger geoptimaliseerd naar kosten. Aan de hand van het flow diagram in Figuur 23 wordt een korte toelichting gegeven van de stappen die worden doorlopen en de keuzes die hierbij gemaakt moeten worden.



Figuur 23. Flow diagram: Objective function

Database & Constantsfile

Het model heeft informatie nodig om de benodigde berekeningen uit te kunnen voeren. Deze informatie is opgeslagen in een database en een constantsfile. In de database is de informatie met betrekking tot de profielen en staalsoorten van de staven opgenomen. Voor dit onderzoek worden warmgewalste buizen van staalsoort S355 beschouwd, zoals beschreven in paragraaf 1.2. De afmetingen van de buizen kunnen varieëren binnen bepaalde grenzen. Met het oog op uitbreiding en om de toepassingsmogelijkheden van het computermodel te vergroten is gekozen om de staafgegevens in een database te verwerken. Andere profielen en staalsoorten kunnen gemakkelijk aan deze database toegevoegd worden. De database bevat de volgende geometrische staafeigenschappen:

- Doorsnedeoppervlak A

- Lengte l
- Volume V
- Eigen gewicht F_self
- Traagheidsmoment I

De ontwerpgegevens, die per ontwerpvraag verschillen en door de gebruiker ingevoerd worden in het computermodel (zie paragraaf 3.1), worden opgeslagen in de constantsfile. De gegevens zijn zo gemakkelijk terug te vinden in het model en kunnen naar behoeven aangepast worden.

Alle informatie die in de database en in de constantsfile verwerkt is, kan vanuit alle modules van het computermodel opgevraagd worden. Door de informatie op deze manier in het model op te nemen, hoeven de gegevens niet aan iedere functie als input meegegeven te worden. Hierdoor wordt het model een stuk overzichtelijker en kunnen aanpassing gemakkelijk doorgevoerd worden.

Input

De objective function heeft als input de waarden van de continue ontwerpvariabelen nodig. Deze variabelen worden verzameld in een lijst: de zogenaamde design vector. De keuze voor een lijst in plaats van losse variabelen is gebaseerd op het feit dat optimalisatie algoritmen vaak met lijsten werken.

Naast de design vector heeft de objective function ook informatie nodig over de discrete ontwerpvariabelen. Het gaat hierbij om combinatie van discrete variabelen waarvoor de kosten berekend moeten worden. De design vector en de combinatie van de discrete variabelen worden door het optimalisatie algoritme aan de objective function meegegeven.

<u>Ontwerpmodule</u>

Knooppunten bepalen

Deze functie bepaalt de x-,y- en z-coördinaten van de knooppunten. Afhankelijk van de configuratie variëren het aantal en de plaats van de knooppunten. De configuratie die toegepast wordt, dient dan ook aan de functie meegegeven te worden. Er zijn drie configuraties verwerkt in het computermodel. Deze configuraties, de parametrisatie van de vakwerkligger en nummering van de knooppunten is terug te vinden in paragraaf 3.2.

Staven toevoegen

Nadat de locatie van de knooppunten bepaald is, worden de staven toegevoegd als elementen tussen twee knooppunten. Wederom hangen het aantal en de plaats van de staven af van de configuratie. Ook in deze functie is daarom de keuze van de configuratie opgenomen. Voor iedere staaf wordt een start- en een eindpunt gedefinieerd. Daarbij wordt ook aan iedere staaf een staaftype gekoppeld. Per staaftype variëren de afmetingen en bijbehorende eigenschappen. De wijze waarop de staven ingedeeld zijn en de bijbehorende nummering is, net als die van de knooppunten, terug te vinden in paragraaf 3.2.

Staaftypen

Bij dit onderzoek wordt onderscheid gemaakt tussen drie staaftypen. Voor ieder staaftype moeten de staafafmetingen aangegeven worden. Deze informatie wordt meegegeven in de vorm van de design vector, zoals hierboven is besproken. Vervolgens kunnen alle benodigde gegevens van de gedefinieerde staaftypen, uit de database worden getrokken.

De drie staaftypen die in het model zijn opgenomen, zijn als volgt:

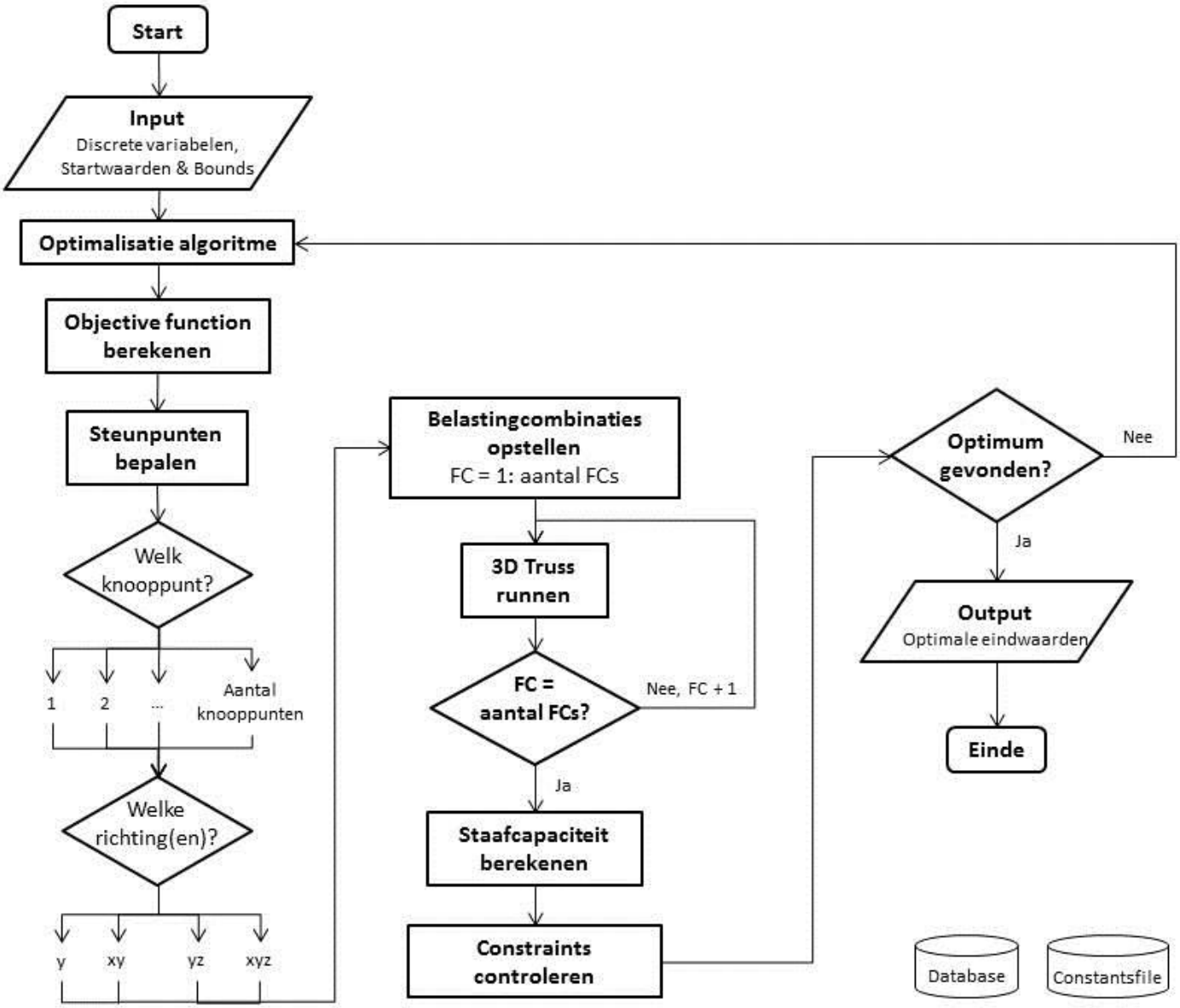
1 = randstaven (zowel boven- als onderrandstaven)
2 = staanders (als onderdeel van de staafgroep: wandstaven)
3 = diagonalen (als onderdeel van de staafgroep: wandstaven)

Kosten berekenen
Aan de hand van het ontwerp dat is opgesteld binnen de ontwerpmodule, worden de totale kosten van de vakwerkligger berekend. Hierbij wordt onderscheid gemaakt tussen de staafkosten en de kosten van de verbindingen. De kosten van de verbindingen zijn afhankelijk van de keuze van het verbindingstype. Er kan gekozen worden om las- of systeemverbindingen toe te passen. Deze keuze is dan ook opgenomen in de functie waarbinnen de verbindingskosten berekend worden.

Output
De output van de objective function zijn de totale kosten van de vakwerkligger. Het optimalisatie algoritme bepaalt op basis van deze output hoe de design vector veranderd moet worden. Het doel hierbij is het bepalen van de design vector waarvoor de output van de objective function minimaal is.

5.3 Optimalisatie algoritme

In deze paragraaf wordt het flow diagram van het optimalisatie algoritme behandeld. Dit flow diagram is weergegeven in Figuur 24. De stappen die doorlopen worden om tot het optimale ontwerp van de vakwerkligger te komen worden hieronder toegelicht.



Figuur 24. Flow diagram: Optimalisatie algoritme

Input
Het optimalisatie algoritme wordt aangeroepen voor alle mogelijke combinaties van de discrete variabelen. De combinatie van discrete variabelen, waarvoor het algoritme de continue ontwerpvariabelen moet optimaliseren, maakt dan ook deel uit van de input. Naast informatie betreffende de discrete ontwerpvariabelen heeft het optimalisatie algoritme ook informatie nodig over de continue ontwerpvariabelen. Per continue variabele moet een startwaarde meegegeven worden. Ook verwacht het optimalisatie algoritme per variabele een set bounds, die aangeeft binnen welke grenzen naar de optimale waarde gezocht moet worden.
De continue ontwerpvariabelen en de bijbehorende startwaarden en bounds die voor dit onderzoek zijn opgesteld, zijn besproken in paragraaf 4.3.

Optimalisatie algoritme
Het optimalisatie algoritme gaat op zoek naar de optimale combinatie van de continue ontwerpvariabelen binnen de opgegeven bounds. Hierbij worden de continue ontwerpvariabelen verwerkt in een lijst, de design vector. De waarden van de design vector worden door het optimalisatie algoritme bij iedere iteratie aangepast totdat de combinatie waarvoor de output van de objective function minimaal is, is gevonden.

Per iteratie doorloopt het optimalisatie algoritme een aantal stappen. Deze stappen worden hieronder verder toegelicht.

Steunpunten bepalen
Deze functie verwerkt de input van de gebruiker met betrekking tot de steunpunten. De knooppunten waar de steunpunten zich bevinden moeten aangegeven worden en ook de richting waarin beweging verhinderd wordt, is nodig. Binnen dit onderzoek wordt aangenomen dat de vakwerkligger in alle knooppunten zijwaarts ondersteund wordt. Dit betekent dat beweging in de y-richting in alle knooppunten wordt verhinderd. Aan de hand van deze aanname zijn vier typen steunpunten gedefinieerd. Deze zijn als volgt:

- Steunpunt waarbij beweging in y-richting wordt verhinderd
- Steunpunt waarbij beweging in x- en y-richting wordt verhinderd
- Steunpunt waarbij beweging in y- en z-richting wordt verhinderd
- Steunpunt waarbij beweging in x-, y- en z-richting wordt verhinderd

De ontwerpvraag die behandeld wordt bij dit onderzoek geeft aan dat in knooppunten 1 en 2 beweging in alle drie de richtingen wordt verhinderd. In alle overige knooppunten wordt, zoals hierboven beschreven, beweging in de y-richting verhinderd.

<u>Module ter bepaling van de belastingen en fundamentele combinaties</u>
Belastingcombinaties opstellen
Binnen deze module wordt de totale belasting per knooppunt berekend. De verschillende belastingen worden gecombineerd in de fundamentele combinaties. Hierbij worden ook de veiligheidsfactoren in rekening gebracht. In het computermodel zijn de twee naar verwachting maatgevende, fundamentele combinaties verwerkt. Deze zijn als volgt:

FC2b = Opwaartse windbelasting is maatgevend
FC3 = Sneeuwbelasting is maatgevend

Bij het berekenen van de totale rekenwaarde van de belasting wordt gebruik gemaakt van de functie die voor de verschillende belastingen de grootte berekend. Deze functie maakt hierbij gebruik van de ontwerpgegevens. Alleen de belastingen die van invloed zijn op de twee hierboven genoemde FCs zijn opgenomen in het model. De volgende belastingen worden beschouwd:

- Belasting ten gevolge van het gewicht van de dakbedekking

- Belasting ten gevolge van het eigen gewicht van de vakwerkligger
- Windbelasting (opwaarts)
- Sneeuwbelasting

Rekenmodule

3D Truss runnen

Het programma 3D Truss heeft informatie nodig om de vakwerkligger door te kunnen rekenen. Deze informatie, de input, wil het programma ontvangen in een specifiek format en als .TRS-file. Voor een uitgebreide beschrijving van het gewenste format wordt verwezen naar het eindrapport van Hoogdoorn (2015, p. 19). De wijze waarop de informatie weggeschreven wordt naar de .TRS-file is als volgt:

```
>> with open('filename', "w") as f:
>> f.write( str(...) + '\n')
```

De informative die weggeschreven moet worden, wordt hierbij geplaatst tussen de haken van str(...). Verder staat '\n' voor *enter*. Het volgende stukje informatie wordt dus op een nieuwe regel in de .TRS-file weggeschreven.

3D Truss wordt in batch mode gerund vanuit Python. De wijze waarop dit gedaan wordt, is reeds beschreven in paragraaf 3.4.

Nadat 3D Truss de ligger heeft doorgerekend, schrijft het programma de resultaten bij in de inputfile. Deze resultaten dienen ingelezen te worden in Python, zodat ze gebruikt kunnen worden bij het controleren van de constraints. Bij het inlezen van de output van 3D Truss wordt gebruik gemaakt van de functie *ConfigParser.ConfigParser()*. Deze aanpak is toegepast naar het voorbeeld van Koning en wordt toegelicht in haar eindrapport (2015, p. 68).

Loop ter verwerking van de FCs

De grootte en richting van de totale belasting die op de vakwerkligger werkt, verschilt per FC. Hierdoor verschillen ook de staafkrachten en knoopverplaatsingen die optreden. Het ontwerp moet, als vanzelfsprekend, alle mogelijke belastingcombinaties aan kunnen. Het is dan ook van belang dat de verschillende FCs doorgerekend worden en dat de optredende krachten en verplaatsingen van deze FCs gecontroleerd worden. Dit wordt gedaan door toepassing van een loop. Deze loop zorgt ervoor dat de FCs doorgerekend worden en dat de output van 3D Truss voor deze FCs wordt opgeslagen. De staafkrachten en knoopverplaatsingen worden verwerkt in een lijst met optredende belastingen respectievelijk een lijst met optredende verplaatsingen. Deze lijsten worden vervolgens gebruikt bij het controleren van de constraints.

Toetsingsmodule

Staafcapaciteit berekenen

Om het ontwerp van de vakwerkligger te kunnen toetsen, moet eerst de capaciteit van de staven berekend te worden. De normaalkrachtcapaciteit wordt voor alle staven berekend en de knikcapaciteit alleen voor de op druk belaste staven.

Constraints controleren

Voor de toetsing van de vakwerkligger is een set constraints opgesteld. Deze constraints beperken het ontwerpgebied waarbinnen het optimale ontwerp gevonden moet worden. De uitkomst van het optimalisatie algoritme dient te voldoen aan alle constraints. De eisen die gesteld worden aan het ontwerp van de vakwerkligger en die als constraints in het computermodel verwerkt zijn, zijn besproken in paragraaf 2.5.

Terugkoppeling

Aan het einde van de serie stappen, die per iteratie uitgevoerd wordt, vindt een terugkoppeling plaats. Indien het optimum gevonden is, is het optimalisatie algoritme klaar en worden de optimale

waarden van de ontwerpvariabelen als output gegeven. Als het optimum nog niet gevonden is of als het ontwerp niet voldoet aan de constraints wordt teruggekoppeld naar het optimalisatie algoritme om de volgende iteratie te starten. Het algoritme past de waarden van de continue ontwerpvariabelen aan en doorloopt opnieuw de serie stappen.

5.4 Discrete ontwerpvariabelen

In de voorgaande paragraaf is het flow diagram van het optimalisatie algoritme behandeld. Hierbij is aangegeven dat het optimalisatie algoritme als input de combinatie van discrete ontwerpvariabelen nodig heeft, waarvoor de continue ontwerpvariabelen geoptimaliseerd moeten worden. De discrete variabelen worden volgens de brute force approach verwerkt in het optimalisatie vraagstuk. Met behulp van een aantal for loops worden alle combinaties doorlopen. Het flow diagram dat de werking van deze for loops visualiseert, is weergegeven in Figuur 25.

Voor dit onderzoek zijn drie discrete ontwerpvariabelen gedefinieerd. Per discrete variabele zijn een eindig aantal keuzes mogelijk. Hieronder staan de discrete ontwerpvariabelen aangegeven met het bijbehorende aantal keuzes.

- Het aantal velden 17 keuzes
- De configuratie 3 keuzes
- Het verbindingstype 2 keuzes

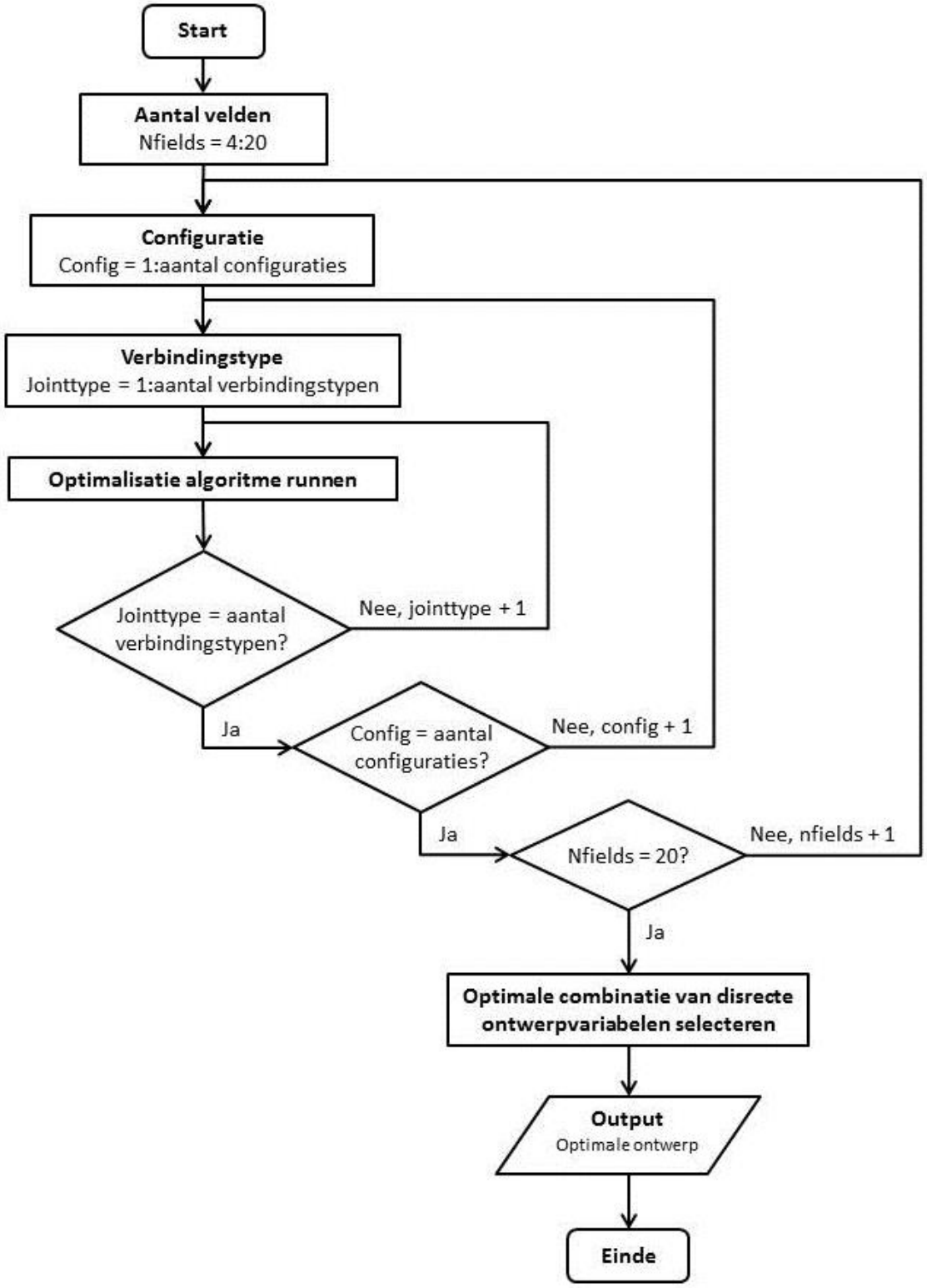
De for loops combineren alle keuzes van de drie discrete variabelen met elkaar. De eerste combinatie wordt verkregen door de eerste keuze voor het aantal velden te combineren met de eerste keuze voor de configuratie en de eerste keuze voor het verbindingstype. Voor deze combinatie van discrete variabelen wordt het optimalisatie algoritme aangeroepen. Vervolgens wordt de vraag gesteld of alle verbindingstypen doorlopen zijn. Aangezien dit de eerste combinatie was en er twee keuzes zijn voor het verbindingstype, is het antwoord op deze vraag *Nee*. Waarop de eerste keuze voor het verbindingstype wordt vervangen door de tweede en er een terugkoppeling plaatsvindt in het stroomschema. Het optimalisatie algoritme wordt opnieuw aangeroepen en krijgt als input de nieuwe combinatie van discrete variabelen mee. Zodra het optimalisatieproces voor deze combinatie is afgerond, wordt wederom gevraagd of alle verbindingstypen doorlopen zijn. Dit keer is het antwoord op deze vraag *Ja*. Er wordt een tweede vraag gesteld, namelijk of alle configuraties doorgelopen zijn. Het antwoord op deze vraag is *Nee.* De eerste keuze voor de configuratie wordt vervangen door de tweede en ook hier vindt een terugkoppeling plaats. Dit keer wordt het optimalisatie algoritme echter niet direct aangeroepen. Eerst wordt wederom de eerste keuze voor het verbindingstype geselecteerd.

De tweede en derde keuze voor de configuratie worden op dezelfde manier als hierboven is beschreven, gecombineerd met beide verbindingstypen. Volgens dit principe worden ook alle keuzes voor het aantal velden doorlopen. Iedere keer dat een nieuwe keuze voor het aantal velden wordt geselecteerd, wordt deze gecombineerd met de eerste keuzes voor de configuratie en het verbindingstype. Waarna eerste beide verbindingstypen doorlopen worden en vervolgens de drie configuraties.

Optimale combinatie van discrete ontwerpvariabelen selecteren & Output van het model

Voor iedere combinatie van de discrete ontwerpvariabelen wordt het optimalisatie algoritme aangeroepen. Zodra het optimale ontwerp dat hoort bij de specifieke combinatie van discrete variabelen gevonden is, wordt een nieuwe combinatie opgesteld en wordt het optimalisatieproces herhaald. In totaal zijn er (17*3*2=) 102 combinaties van de discrete variabelen mogelijk. Dit betekent dat het optimalisatie algoritme 102 keer aangeroepen wordt. Nadat het optimalisatie algoritme de optimale waarden van de continue ontwerpvariabelen heeft bepaald voor alle 102 mogelijke combinaties, wordt de combinatie geselecteerd waarvoor de kosten minimaal zijn. Deze optimale combinatie van de discrete ontwerpvariabelen met de bijbehorende waarden van de

continue ontwerpvariabelen, die door het optimalisatie algoritme zijn bepaald, vormen het optimale ontwerp. Als output geeft het computermodel de waarden van alle ontwerpvariabelen, die horen bij dit optimale ontwerp.



Figuur 25. Flow diagram: For loops ter verwerking van de discrete variabelen

5.5 Validatie

De werking van het computermodel is gecontroleerd met behulp van handberekeningen en het programma MatrixFrame. In bijlage C is de validatie van het eerste model te vinden. De wijze waarop de code vanuit Python weggeschreven wordt naar een .TRS-file, wordt hierbij gecontroleerd. De output van 3D Truss is vergeleken met die van MatrixFrame. Uit deze vergelijking blijkt dat de output van beide programma's overeenkomt. Zodoende is geconcludeerd dat de wijze waarop de inputfile voor 3D Truss geschreven wordt, correct is.

In bijlage D zijn handberekeningen opgenomen, die gebruikt zijn om (een verouderde versie van) het computermodel te valideren. De verschillende onderdelen van het computermodel zijn één voor één gecontroleerd door de output van het model te vergelijken met de resultaten van de handberekeningen. Iedere keer dat het model is aangepast of uitgebreid, is de werking van het model opnieuw gecontroleerd met nieuwe handberekeningen en/of met behulp van MatrixFrame. Het bleek veel tijd te kosten om de handberekeningen presentabel uit te werken. Aangezien de wijze waarop de code gevalideerd is niet veranderd is gedurende het onderzoek, is dan ook gekozen een enkele versie van de handberekeningen op te nemen in de bijlagen.

6 Resultaten

De resultaten van het onderzoek worden in dit hoofdstuk gepresenteerd. In de eerste paragraaf wordt het verloop van de totale kosten voor iedere combinatie van configuratie en verbindingstype besproken. Hierbij wordt onderzocht wat de invloed is van de verhouding tussen de staaf- en de verbindingskosten op het verloop van de totale kosten.

In paragraaf 2 wordt het optimale ontwerp dat door het computermodel is bepaald, behandeld. Er wordt gecontroleerd of er daarwerkelijk sprake is van een optimum door de ontwerpvariabelen één voor één aan te passen.

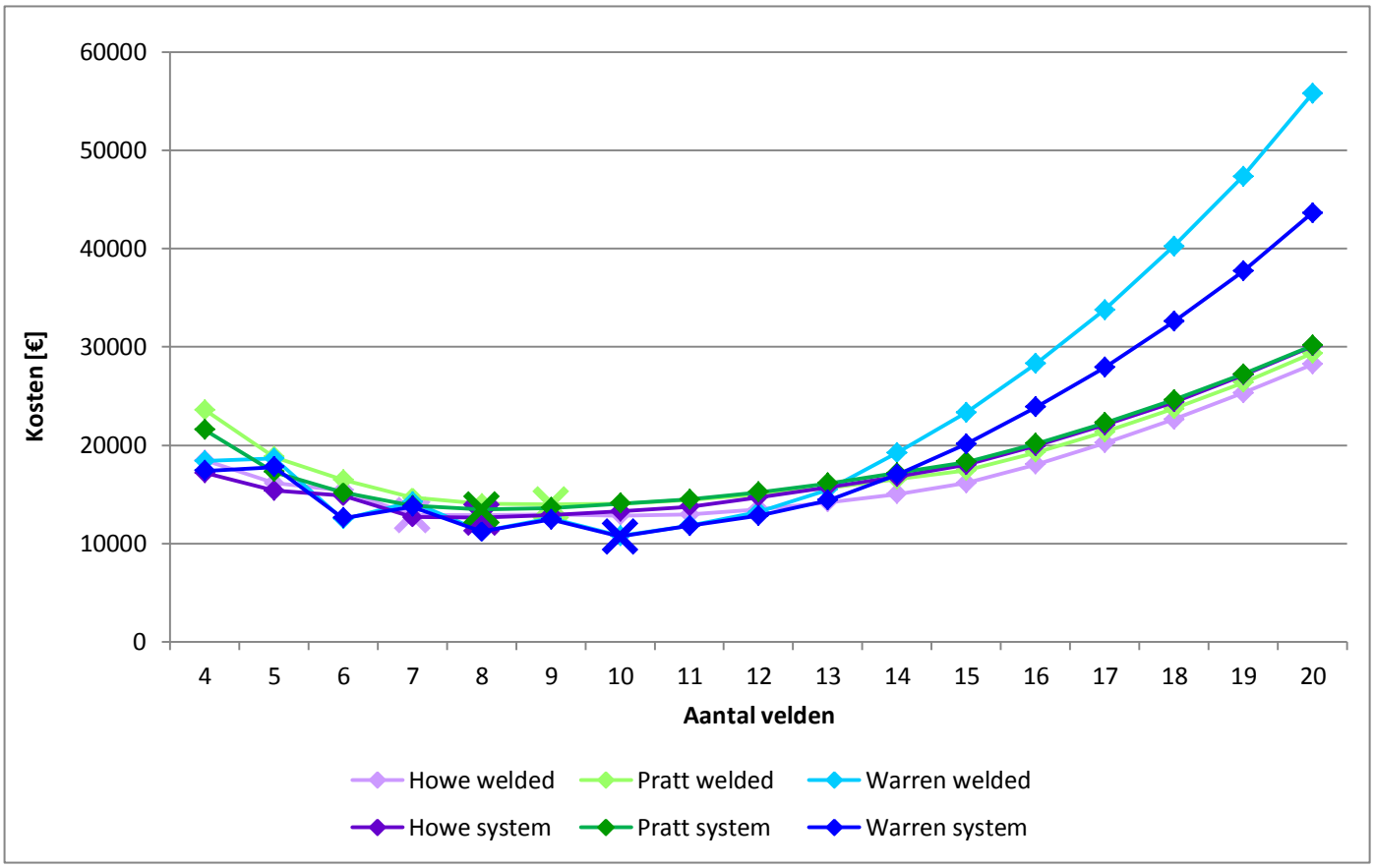
Vervolgens wordt in paragraaf 3 aan de hand van het verloop van de objective function van het optimale ontwerp, de werking van het optimalisatie algoritme inzichtelijk gemaakt.

Tot slot wordt in paragraaf 4 gecontroleerd of het optimale ontwerp voldoet aan alle constraints. Hierbij wordt ook onderzocht welke FC en welke constraints maatgevend zijn voor de dimensionering van de vakwerkligger.

6.1 Verband tussen Totale kosten en Aantal Velden van de vakwerkligger

Het computermodel heeft nog geen zes uur nodig gehad om alle 102 combinaties van de discrete variabelen te optimaliseren. In Grafiek 2 zijn de totale kosten van de vakwerkligger per combinatie van configuratie en verbindingstype, uitgezet tegen het aantal velden waaruit de ligger wordt opgebouwd. Te zien is dat de lijnen voor het gedeelte tot 14 velden, erg dicht bij elkaar liggen en dat het verloop redelijk vlak is. Verder valt op dat de kosten vanaf 13 velden voor de configuratie Warren truss ineens sterk toenemen. Hieruit kan geconcludeerd worden dat het niet verstandig is om deze configuratie te kiezen indien meer dan 13 velden worden toegepast.

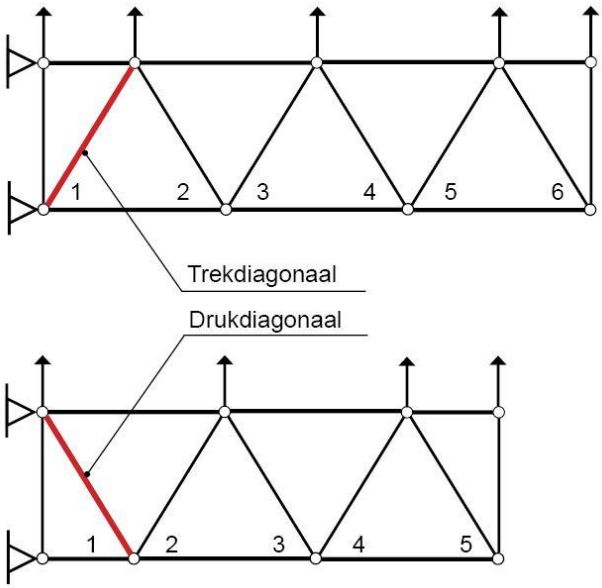
Grafiek 2. Totale kosten vs. Aantal velden (per combinatie van configuratie en verbindingstype)



Om beter inzicht te krijgen in het verloop van de kosten rondom de optima van de verschillende combinaties is een gedeelte van Grafiek 3 uitvergroot weergegeven in Grafiek 2. In deze uitvergroting worden vier dingen zichtbaar.

Ten eerste is in Grafiek 2Grafiek 3 te zien dat het verloop dat hoort bij de configuratie Warren sprongen vertoont. Deze sprongen duiden op een verschil tussen een Warren truss met een even en met een oneven aantal velden. Dit verschil kan verklaard worden door deze twee ontwerpen met elkaar te vergelijken.

Uit de resultaten van het computermodel is gebleken dat de belastingcombinatie waarbij de opwaartse windbelasting als maatgevend is gekozen, bepalend is voor de dimensionering van de ligger. Dit wordt in paragraaf 6.4 van dit hoofdstuk inzichtelijk gemaakt. In Figuur 26 zijn de twee ontwerpen van een vakwerkligger weergegeven, die belast wordt door de maatgevende FC. Beide liggers zijn Warren trusses. De ene ligger bestaat uit een even aantal velden en de andere uit een oneven aantal. In de figuur is bij beide liggers de zwaarst belaste diagonaal in het rood aangegeven. Te zien is dat in het geval dat de ligger bestaat uit een even aantal velden, deze diagonaal een trekdiagonaal is. Indien de ligger uit een oneven aantal velden bestaat is deze diagonaal daarentegen een drukdiagonaal. In geval van drukbelasting moet er rekening gehouden worden met knikinstabiliteit. Dit is ongunstig ten aanzien van de dimensionering en daarmee ook ten aanzien van de kosten.



Figuur 26. Boven: Warren truss met even aantal velden Onder: Warren truss met oneven aantal velden

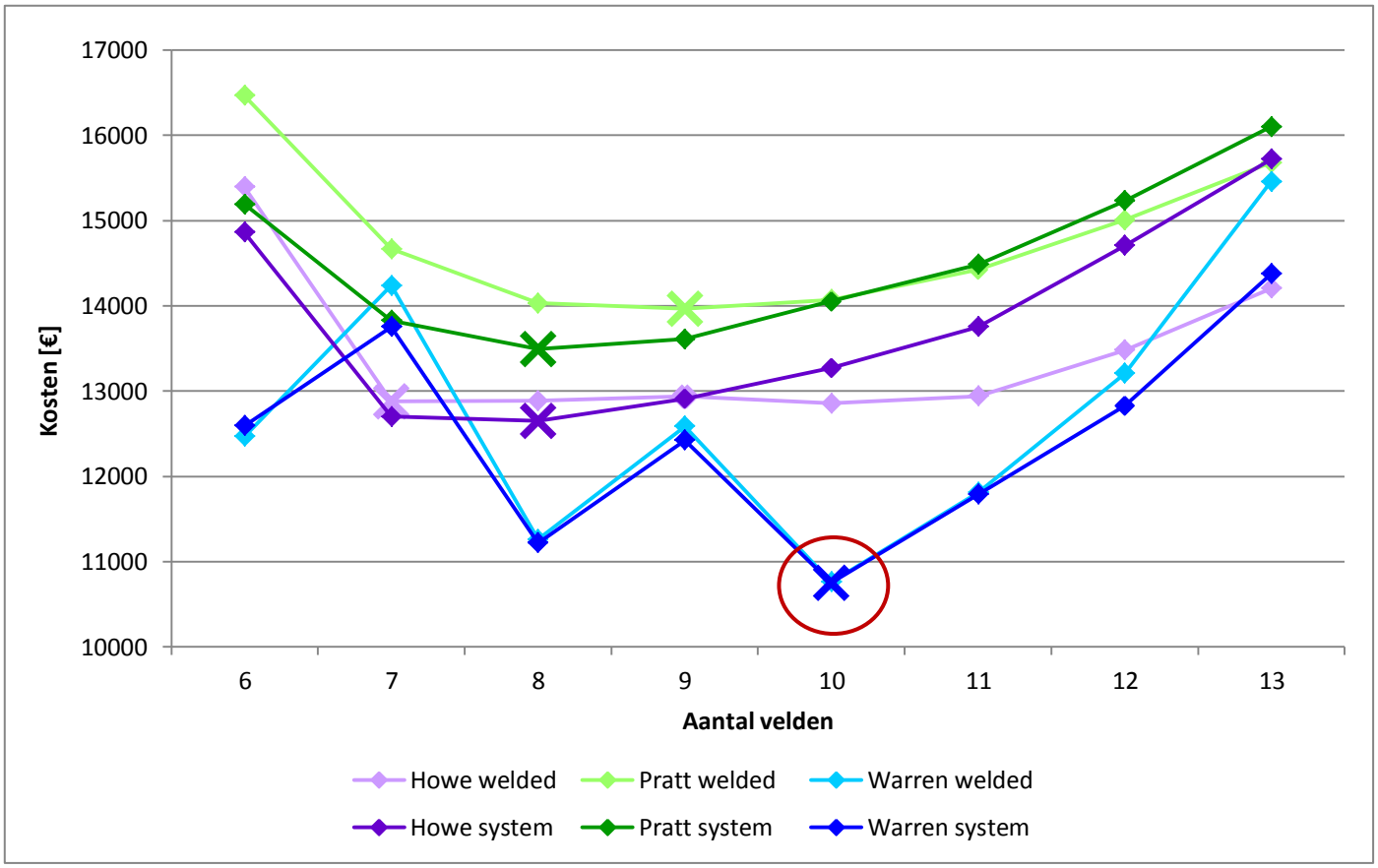
Ten tweede zijn de optima van de verschillende combinaties nu beter te onderscheiden. Voor iedere combinatie is het optimale aantal velden aangegeven met een kruis. Het optimale aantal velden varieert van 7 tot en met 10 velden. Voor iedere combinatie is bepaald hoe groot het kostenverschil is indien er een extra veld wordt toegepast. Hieruit is gebleken dat binnen de range van 7 tot en met 11 velden de kosten gemiddeld 1,8 % verschillen met een maximum van 4 % verschil. Het verloop van de kosten is voor dit gedeelte dus erg vlak. Buiten deze range is het kostenverschil per extra veld significant groter. Het gemiddelde verschil bedraagt 10 % en dit kan oplopen tot ruim 20 %. Op basis van deze gegevens kan geconcludeerd worden dat er het beste gekozen kan worden uit 7 tot en met 11 velden. Deze conclusie is overigens alleen geldig indien alle configuraties beschouwd worden.

Ten derde is te zien dat de kosten in de meeste gevallen het laagste zijn indien een Warren truss configuratie wordt toegepast. De viel te verwachten aangezien een Warren truss minder knopen heeft dan een Howe of Pratt truss. Daar staat tegenover dat de staafafmetingen van een Warren truss groter zijn, omdat de staven relatief gezien grotere krachten moeten afdragen. Doordat de verbindingskosten dominant zijn ten opzichte van de staafkosten, is een Warren truss goedkoper ondanks de grotere staafafmetingen.

Tot slot geeft Grafiek 2 weer welke combinatie van discrete variabelen het beste toegepast kan worden. Deze optimale combinatie is aangegeven met een rode cirkel. Het is duidelijk te zien dat het gaat om de configuratie Warren toegepast op een ligger bestaande uit 10 velden. Het verschil tussen de toepassing van las- en systeemverbindingen is echter zo klein, dat dit niet uit de grafiek af te lezen is. De combinatie Warren met systeemverbindingen blijkt nog geen 11 euro goedkoper dan die met lasverbindingen.

Om aan te tonen dat er sprake is van een significant verschil in kosten tussen het optimale ontwerp en de overige ontwerpen, is per ontwerp het kostenverschil ten opzichte van het optimum berekend. Hierbij wordt onderscheid gemaakt tussen de ontwerpen die vallen binnen de range van 7 tot en met 11 velden en de ontwerpen daarbuiten. Binnen deze range kosten de ontwerpen gemiddeld 22 % meer dan het optimale ontwerp. Dit verschil kan oplopen tot 40 %. Buiten de range van 7 tot en met 11 velden bedraagt het kostenverschil minimaal 25 % en gemiddeld 70 %! Dit betekent dat de ontwerpen, waarbij minder dan 7 of meer dan 11 velden toegepast worden, gemiddeld ruim 7500 euro duurder zijn dan het optimale ontwerp. Dit bedrag geldt voor een enkele vakwerkligger. Indien er meerdere vakwerkliggers worden toegepast, wat doorgaans het geval is bij vakwerkconstructies, kan dit bedrag enorm oplopen. Dit geeft aan dat de informatie die het computermodel levert erg waardevol kan zijn. Met name indien er naar een grote range van het aantal velden wordt gekeken.

Grafiek 3. Totale kosten vs. Aantal velden voor 6 t/m 13 (per combinatie van configuratie en verbindingstype)

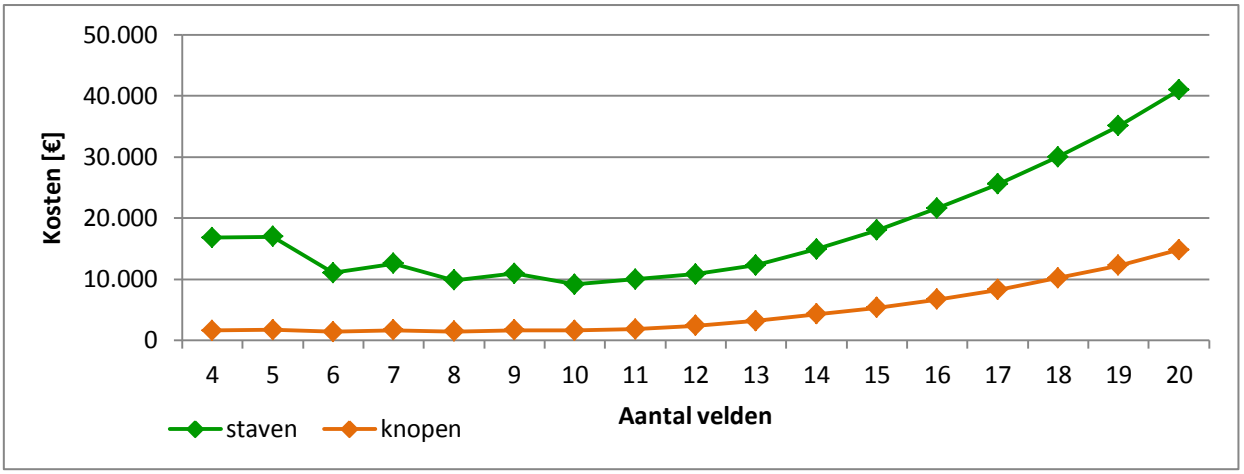


De conclusies die zojuist getrokken zijn naar aanleiding van het verloop van de totale kosten, zijn alleen van toepassing als de staaf- en verbindingskosten berekend zijn volgens de wijze die toegelicht is in paragraaf 4.1**.** In Grafiek 4 zijn de staaf- en verbindingskosten voor configuratie Warren en lasverbindingen afzonderlijk weergegeven. Het is duidelijk dat in dit geval de staafkosten dominant zijn en het verloop van de totale kosten bepalen. Dit geldt ook voor de andere combinaties van configuratie en verbindingstype. Wanneer de verhouding tussen de staaf- en de verbindingskosten verandert, verandert naar alle waarschijnlijkheid ook het verloop van de totale kosten.

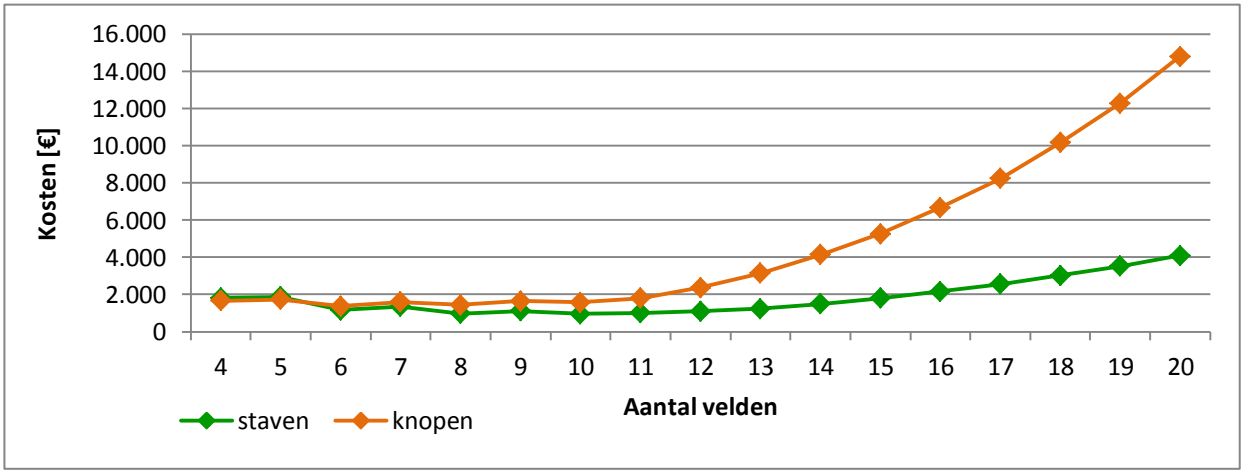
Het effect van een verandering van deze verhouding is onderzocht door de staafkosten te verkleinen met een factor 10. In Grafiek 5 en Grafiek 6 zijn wederom de staaf- en verbindingskosten afzonderlijk weergegeven. Dit keer voor configuratie Warren in combinatie met lasverbindingen respectievelijk systeemverbindingen. Uit deze grafieken blijkt dat het bij deze aangepaste kostenverhouding afhangt van het aantal velden en het verbindingstype of de staaf- of de

verbindingskosten dominant zijn. Indien lasverbindingen worden toegepast, zijn de verbindingskosten dominant voor ieder aantal velden met uitzondering van 4 en 5. In het geval dat systeemverbindingen toegepast worden, zijn de verbindingskosten alleen dominant voor liggers opgebouwd uit 8 tot en met 17 velden. Voor liggers met minder dan 8 of meer dan 17 velden zijn de staafkosten dominant. Het verband tussen de kosten van de systeemsverbindingen en het aantal toegepaste velden is lineair, zoals te zien is in Grafiek 6. Dit is logisch aangezien voor ieder extra veld dat wordt toegepast er een extra knoop nodig is. De kosten per systeemknoop zijn constant en zodoende nemen de verbindingskosten lineair toe met een toename van het aantal velden.

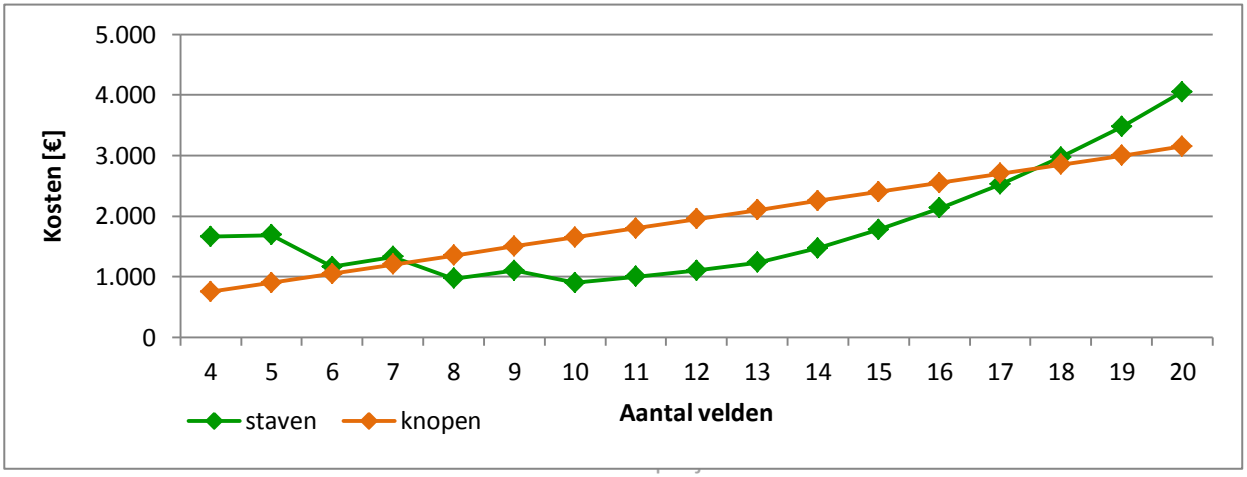
Grafiek 4. Gesplitste kosten vs. Aantal velden voor configuratie Warren met lasverbindingen



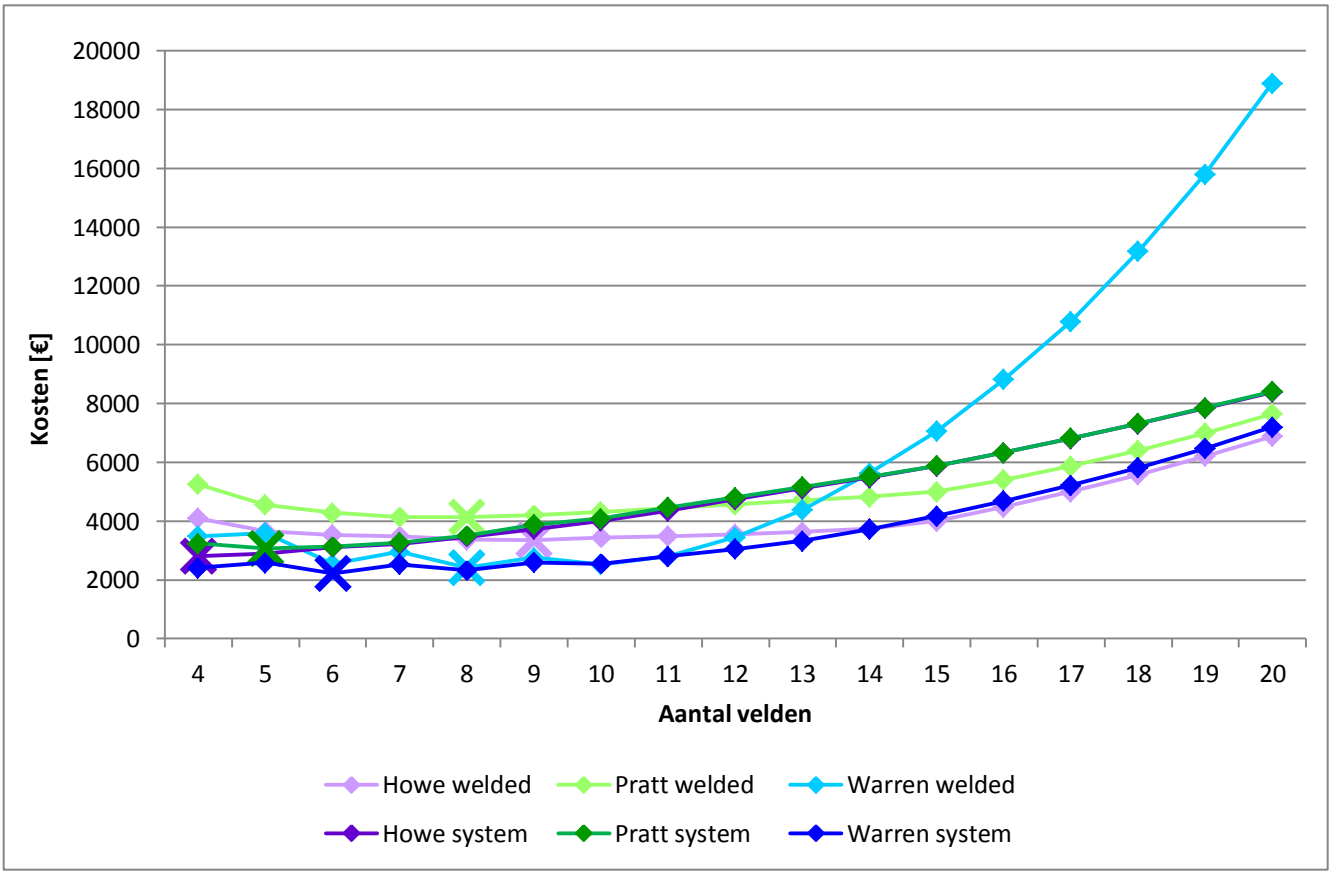
Grafiek 5. Gesplitste kosten vs. Aantal velden voor configuratie Warren en lasverbindingen. Staafkosten maal 1/10.



Grafiek 6. Gesplitste kosten vs. Aantal velden voor configuratie Warren en systeemverbindingen. Staafkosten maal 1/10.



Grafiek 7. Totale kosten vs. Aantal velden (per configuratie en verbindingstype). Staafkosten maal 1/10.



Het verloop van de totale kosten, waarbij de staafkosten zijn aangepast, uitgezet tegen het aantal velden van de vakwerkligger is voor iedere combinatie van configuratie en verbindingstype weergegeven in Grafiek 7. Naast het feit dat de kosten, als vanzelfsprekend, voor iedere combinatie van configuratie en verbindingstype een stuk lager zijn, vallen er nog drie dingen op.

Ten eerste is te zien dat het verloop van de totale kosten voor de combinaties waarbij systeemsverbindingen zijn toegepast, is veranderd ten opzichte van het verloop weergegeven in Grafiek 2. Bij de oorspronkelijke kostenverhouding vertoonden deze combinaties een parabolisch verloop. Na aanpassing van de kostenverhouding is er i.p.v. een parabolisch verloop een lineair verloop te zien. Dit verschil kan op een logische wijze verklaard worden. Voor de aanpassing van de kostenverhouding waren de staafkosten, met een parabolisch verloop, dominant. De verandering van de kostenverhouding heeft ertoe geleid dat de verbindingskosten dominant zijn geworden. Er is reeds toegelicht dat de kosten van de systeemsverbindingen lineair toenemen met een toename van het aantal velden. Nu deze verbindingskosten dominant zijn geworden, is het lineaire verloop bepalend voor het verloop van de totale kosten.

Het verloop van de totale kosten voor de combinaties waarbij lasverbindingen zijn toegepast, is niet zo sterk veranderd. Het enige verschil is dat het verloop vlakker is geworden.

Ten tweede is er een verschuiving van het optimale aantal velden zichtbaar. Ook hierbij is het verschil groter voor combinaties waarbij systeemverbindingen zijn toegepast dan voor die waarbij lasverbindingen zijn toegepast. Voor deze eerste groep combinaties varieert het optimale aantal velden van 4 tot 6, terwijl dit eerst van 8 tot 10 was. Er is dus sprake van een aanzienlijke verlaging van het optimale aantal velden.

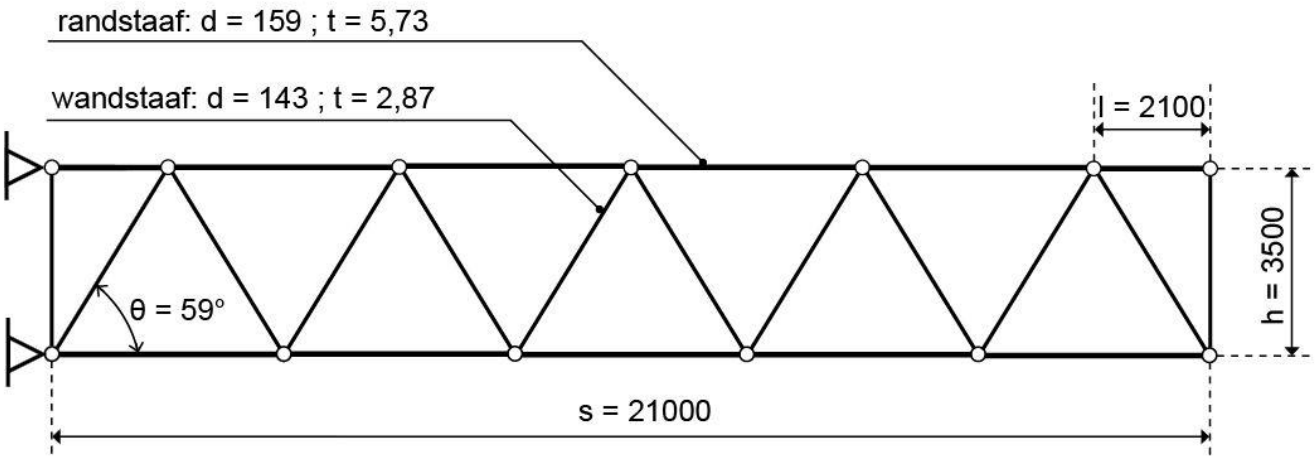
Ten derde valt het op dat de kosten alleen voor configuratie Warren in combinatie met lasverbindingen een sterke stijging vertonen. Dit terwijl deze stijging voor de aanpassing van de

kostenverhouding ook te zien was bij de combinatie van Warren met systeemverbindingen. Met behulp van de twee grafieken die de gesplitste kosten voor beide situaties weergeven (Grafiek 5 en Grafiek 6)**,** kan dit verschil verklaard worden. De staafkosten vertonen een parabolisch verloop, zoals te zien is in beide grafieken. Ook de kosten van lasverbindingen vertonen een parabolisch verloop. Indien lasverbindingen worden toegepast, is het verloop van de totale kosten dan ook altijd parabolisch. Het maakt hierbij niet uit of de staaf- of de verbindingskosten dominant zijn. In het geval dat systeemverbindingen worden toegepast, maakt het daarentegen wel verschil welke kosten dominant zijn. De kosten van systeemverbindingen vertonen, zoals reeds is besproken, een lineair verband. Indien de staafkosten dominant zijn, zoals bij Grafiek 2 het geval is, overwint het parabolische verloop van de staven het van het lineaire verloop van de systeemverbindingen. Door de verkleining van de staafkosten wordt het verloop van de totale kosten niet meer geheel door het parabolische verloop van de staven bepaald. Het lineaire verloop van de systeemverbindingen speelt nu ook een rol. In Grafiek 6 is te zien dat de kosten van de staven en die van de systeemverbindingen erg dicht bij elkaar liggen. Ze dragen allebei ongeveer evenveel bij aan het verloop van de totale kosten. Onder invloed van de kosten van de systeemverbindingen wordt het parabolisch verloop van de stavenkosten afgevlakt. De kostenstijging bij een toenemend aantal velden, zoals deze te zien is in Grafiek 2, is daardoor niet terug te zien in Grafiek 7.

6.2 Het optimale ontwerp voor de vakwerkligger

Het computermodel geeft als output het ontwerp van de vakwerkligger, waarvoor de kosten minimaal zijn: het optimale ontwerp. De waarden van de discrete ontwerpvariabelen die horen bij dit optimale ontwerp, zijn reeds besproken in de voorgaande paragraaf. Deze en de continue ontwerpvariabelen van het optimale ontwerp zijn weergegeven in Tabel 10. Daarbij is in Figuur 27 het optimale ontwerp afgebeeld. De kosten van dit optimale ontwerp zijn: € 10.750,-.

In deze paragraaf wordt onderzocht of het optimale ontwerp dat het model als output heeft gegeven daadwerkelijk een optimum is. Dit wordt gecontroleerd door de ontwerpvariabelen één voor één te veranderen en het aangepaste ontwerp door het computermodel te laten doorrekenen. Indien voor alle aanpassingen de kosten van het aangepaste ontwerp toenemen en/of dit ontwerp niet voldoet aan de constraints, kan geconcludeerd worden dat er inderdaad sprake is van een optimum.



Figuur 27. Het optimale ontwerp van de uitkragende vakwerkligger

Tabel 10. De ontwerpvariabelen van het optimale ontwerp

	Ontwerpvariabele	Optimale waarde	Eenheid
Discreet	Configuratie	Warren truss	-
	Aantal velden	10	-
	Type verbinding	systeemverbindingen	-
Continu	Diameter randstaven	159	mm
	Wanddikte randstaven	5,73	mm
	Diameter wandstaven	143	mm
	Wanddikte wandstaven	2,87	mm
	Hoogte van de vakwerkligger	3,52	m

Controle van het optimale ontwerp

De ontwerpvariabelen worden één voor één veranderd. Voor de discrete ontwerpvariabelen worden de twee andere configuraties, het andere verbindingstype en het optimale aantal velden plus en min 1 beschouwd. De continue ontwerpvariabelen worden aangepast door er 1% bij op te tellen of van af te trekken. In Tabel 11 is het effect van deze veranderingen weergegeven. De aangepaste waarden van de ontwerpvariabelen met de bijbehorende kostenverschillen zijn in deze tabel opgenomen. Ook is in de tabel te zien of de aangepaste ontwerpen voldoen aan de constraints. Indien een ontwerp niet voldoet aan de constraints, is de constraint die overschreden wordt ook in de tabel opgenomen.

Tabel 11. Aanpassing van de ontwerpvariabelen en effect van deze aanpassing

Aangepaste ontwerpvariabele	Aangepaste waarde	Kostenverschil [€]	Voldoet het ontwerp?	Ontwerp voldoet niet aan:
Configuratie	Howe	+ 3061	Ja	-
Configuratie	Pratt	+ 3061	Nee	Knik
Aantal velden	11	+ 341	Nee	Knik Hoek tussen rand- en wandstaaf
Aantal velden	9	- 273	Nee	Knik
Type verbinding	**Lasverbindingen**	**- 15**	**Ja**	**-**
Diameter randstaven	1,01 * $d_{randstaaf}$	+ 99	Ja	-
Diameter randstaven	0,99 * $d_{randstaaf}$	- 99	Nee	Knik
Wanddikte randstaven	1,01 * $t_{randstaaf}$	+ 92	Ja	-
Wanddikte randstaven	0,99 * $t_{randstaaf}$	- 91	Nee	Knik
Diameter wandstaven	1,01 * $d_{wandstaaf}$	+ 33	Nee	d/t-verhouding van de wandstaven
Diameter wandstaven	0,99 * $d_{wandstaaf}$	- 33	Nee	Knik
Wanddikte wandstaven	1,01 * $t_{wandstaaf}$	+ 31	Ja	-
Wanddikte wandstaven	0,99 * $t_{wandstaaf}$	- 31	Nee	d/t-verhouding van de wandstaven
Hoogte van de vakwerkligger	1,01 * h	+ 20	Ja	-
Hoogte van de vakwerkligger	0,99 * h	- 20	Nee	Knik

Opmerking: De diameter en wanddikte van zowel de rand- als de wandstaven evenals de hoogte van de vakwerkligger zijn terug te vinden in Tabel 10.

Met uitzondering van één ontwerp, worden voor alle aangepaste ontwerpen de kosten hoger en/of voldoen de ontwerpen niet aan de constraints. De enige uitzondering is het ontwerp waarbij lasverbindingen worden toegepast i.p.v. systeemverbindingen (dikgedrukt in tabel Tabel 11). Voor dit aangepaste ontwerp zijn de kosten 15 euro lager en wordt aan alle constraints voldaan. In eerste instantie lijkt het er door deze uitkomst op dat het computermodel een fout heeft gemaakt. Er is tenslotte een ander ontwerp gevonden dat voldoet aan de eisen en ook nog eens goedkoper is. Deze uitkomst kan echter op een logische manier verklaard worden.

Het kostenverschil bedraagt slechts 15 euro. Dit is minder dan 0,2 % van de kosten van het het optimale ontwerp dat door het computermodel is gevonden. Het optimalisatie algoritme heeft daarentegen een tolerantie van 1 % meegekregen. Dit betekent dat het algoritme het ontwerp geoptimaliseerd heeft, totdat het verschil in kosten kleiner werd dan € 107,50. Een kostenverschil kleiner dan deze 1 % geeft dan ook niet aan dat er een fout zit in het model of dat het gevonden ontwerp geen optimum is. Het optimalisatie algoritme heeft simpelweg niet de opdracht gekregen om in zoveel detail het ontwerp te optimaliseren.

Om de verklaring te onderbouwen is het computermodel nogmaals gerund voor twee combinaties van de discrete ontwerpvariabelen. Deze combinaties zijn: Warren truss, 10 velden en systeemverbindingen respectievelijk lasverbindingen. Dit keer is is het optimalisatie algoritme ingesteld op een tolerantie van 10^{-8} %. De geoptimaliseerde ontwerpen voor deze combinaties zijn als volgt:

Warren truss, 10 velden, Systeemverbindingen:
Kosten: € 10.641,-
Warren truss, 10 velden, Lasverbindingen:
Kosten: € 10.658,-

De optimale combinatie van discrete variabelen blijkt onveranderd. Doordat er echter een lage tolerantie is gehanteerd, zijn de waarden van de continue ontwerpvariabelen verder geoptimaliseerd dan eerst het geval was. De ontwerpvariabelen en de nieuwe optimale waarden weergegeven in Tabel 12.

Voor deze set ontwerpvariabelen is wederom het type verbindingen aangepast. Deze aanpassing resulteert dit keer wel in hogere kosten. Ook voldoet het aangepaste ontwerp niet meer aan de eis die gesteld wordt aan de hoek tussen de rand- en wandstaven.

Door de ontwerpvariabelen één voor één te veranderen, is gebleken dat er voor het ontwerp dat het computermodel als output heeft gegeven, inderdaad sprake is van het optimum. Het is belangrijk dat hierbij gerealiseerd wordt dat het detail waarmee dit optimale ontwerp bepaald is, afhangt van de tolerantie van het optimalisatie algoritme.

Tabel 12. De ontwerpvariabelen van het optimale ontwerp voor een tolerantie van 10e-8

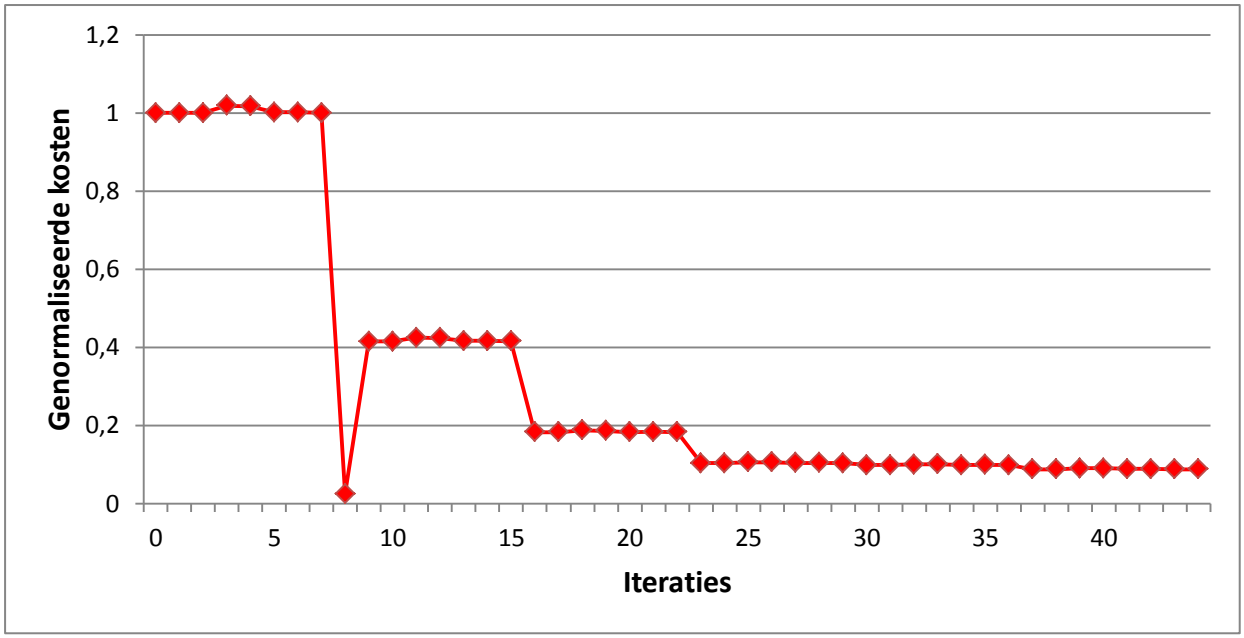
	Ontwerpvariabele	Optimale waarde	Eenheid
Discreet	Configuratie	Warren truss	-
	Aantal velden	10	-
	Type verbinding	systeemverbindingen	-
Continu	Diameter randstaven	155	mm
	Wanddikte randstaven	5,77	mm
	Diameter wandstaven	144	mm
	Wanddikte wandstaven	2,89	mm
	Hoogte van de vakwerkligger	3,65	m

6.3 Verloop van de objective function

In deze paragraaf wordt de wijze waarop het optimalisatie algoritme werkt toegelicht aan de hand van het verloop van de objective function van het optimale ontwerp. De genormaliseerde kosten zijn uitgezet tegen de iteraties in Grafiek 8.

In het verloop van de genormaliseerde kosten zijn duidelijke plateaus en sprongen te herkennen. De plateaus stellen de momenten voor waar het optimalisatie algoritme de continue ontwerpvariabelen één voor één een klein beetje aanpast. Door deze kleine aanpassingen te doen kan het optimalisatie algoritme voor iedere ontwerpvariabele bepalen of deze groter of kleiner moet worden om de kosten te verlagen. Vervolgens stelt het optimalisatie algoritme met behulp van deze informatie een geheel nieuwe set ontwerpvariabelen op. Deze nieuwe sets zijn te herkennen aan de sprongen in de grafiek. Deze sprongen zijn in het begin groot en duidelijk terug te zien in de grafiek, zoals bij iteraties 8, 9, en 16. Naarmate het optimalisatieproces vordert worden de sprongen steeds kleiner en zijn ze in de grafiek nauwelijks nog zichtbaar. In de data zijn ze echter nog wel duidelijk terug te vinden. De nieuwe sets worden ingezet bij iteraties 23, 30, 37 en 44. Opvallend is dat er na de sprong bij iteratie 8 geen plateau volgt. Dit komt doordat de constraints sterk overschreden worden bij het ontwerp, behorende bij deze set ontwerpvariabelen. Hierdoor is dit ontwerp ver buiten het ontwerpgebied komen te liggen. Het optimalisatie algoritme heeft dat direct opgemerkt en heeft zichzelf gecorrigeerd door de waarden van de ontwerpvariabelen bij te stellen.

Grafiek 8. Genormaliseerde kosten vs. Iteraties (10 velden, Warren truss, systeemverbindingen)



6.4 Constraints voor het optimale ontwerp

In paragraaf 2.5 zijn de eisen opgesteld waaraan het ontwerp van de vakwerkligger moet voldoen. Deze eisen zijn in de vorm van constraints verwerkt in het computermodel. Gedurende het optimalisatieproces veranderen de parameters waarvoor de constraints zijn opgesteld. In deze paragraaf wordt gecontroleerd of het optimale ontwerp, besproken in de voorgaande paragrafen, voldoet aan alle constraints. Daarnaast wordt onderzocht welke FC en welke constraints maatgevend zijn voor de dimensionering van de vakwerkligger. Dit wordt gedaan door het verloop van de parameters gedurende het optimalisatieproces van het optimale ontwerp, te bekijken.

Voor het optimale ontwerp zijn de waarden van de parameters, waarvoor de constraints zijn opgesteld, weergegeven in

Tabel 13. Per constraint wordt aangegeven wat de eis is en of het optimale ontwerp aan deze eis voldoet. In de tabel kan afgelezen worden dat het optimale ontwerp voldoet aan alle constraints. Zodoende kan geconcludeerd worden dat het optimalisatie algoritme op de juiste wijze omgaat met de constraints die worden meegegeven.

Er zijn drie constraints waarvoor de waarde van het optimale ontwerp zich heel dicht tegen de grens, zo niet op de grens bevindt. Deze constraints zijn dikgedrukt weergegeven in de tabel. Het gaat om de eisen met betrekking tot de knikkracht van de zwaarst belaste randstaaf, de d/t-verhouding van de wandstaven en de wanddikteverhouding van de rand- en de wandstaven. Het verloop van deze parameters gedurende het optimalisatieproces van het optimale ontwerp is weergegeven in Grafiek 9, Grafiek 10 en Grafiek 11. In bijlage E is het verloop van de overige parameters gedurende het optimalisatieproces van het optimale ontwerp weergegeven.

Tabel 13. Controle van de constraints voor het optimale ontwerp

Constraints	Waarde voor het optimale ontwerp	Controle
UC normaalkracht ≤ 1,0 van de zwaarst belaste randstaaf	0,917	Voldoet
UC normaalkracht ≤ 1,0 van de zwaarst belaste wandstaaf	0,73	Voldoet
UC knikkracht ≤ 1,0 **van de zwaarst belaste randstaaf**	**1,00**	**Voldoet**
UC knikkracht ≤ 1,0 voor de zwaarst belaste wandstaaf	0,99	Voldoet
UC verticale verplaatsing ≤ 1,0 Van het uiteinde van de ligger	0,87	Voldoet
$30° \leq \theta \leq 60°$	59°	Voldoet
$\frac{d_{randstaaf}}{t_{randstaaf}} \leq 50{,}0$	159 / 5,7 = 27,7	Voldoet
$\boldsymbol{\frac{d_{wandstaaf}}{t_{wandstaaf}} \leq 50{,}0}$	**144 / 2,8 = 50,0**	**Voldoet**
$0{,}2 \leq \frac{d_{wandstaaf}}{d_{randstaaf}} \leq 1{,}0$	144 / 159 = 0,90	Voldoet
$\boldsymbol{\frac{t_{randstaaf}}{t_{wandstaaf}} \geq 2{,}0}$	**5,7 / 2,8 = 2,0**	**Voldoet**

Het verloop van de UC van de knikkracht van de zwaarst belaste randstaaf en diagonaal is per FC weergegeven in Grafiek 9. In deze grafiek komen een aantal dingen terug die beschreven zijn bij het verloop van de objective function in de voorgaande paragraaf. Zo zijn er ook hier plateaus en sprongen te herkennen. Daarbij is ter hoogte van iteratie 8 een enorme piek te zien, waarbij de UC sterk wordt overschreden. Dit komt overeen met de duik van de objective function op dezelfde plek in het optimalisatieproces.

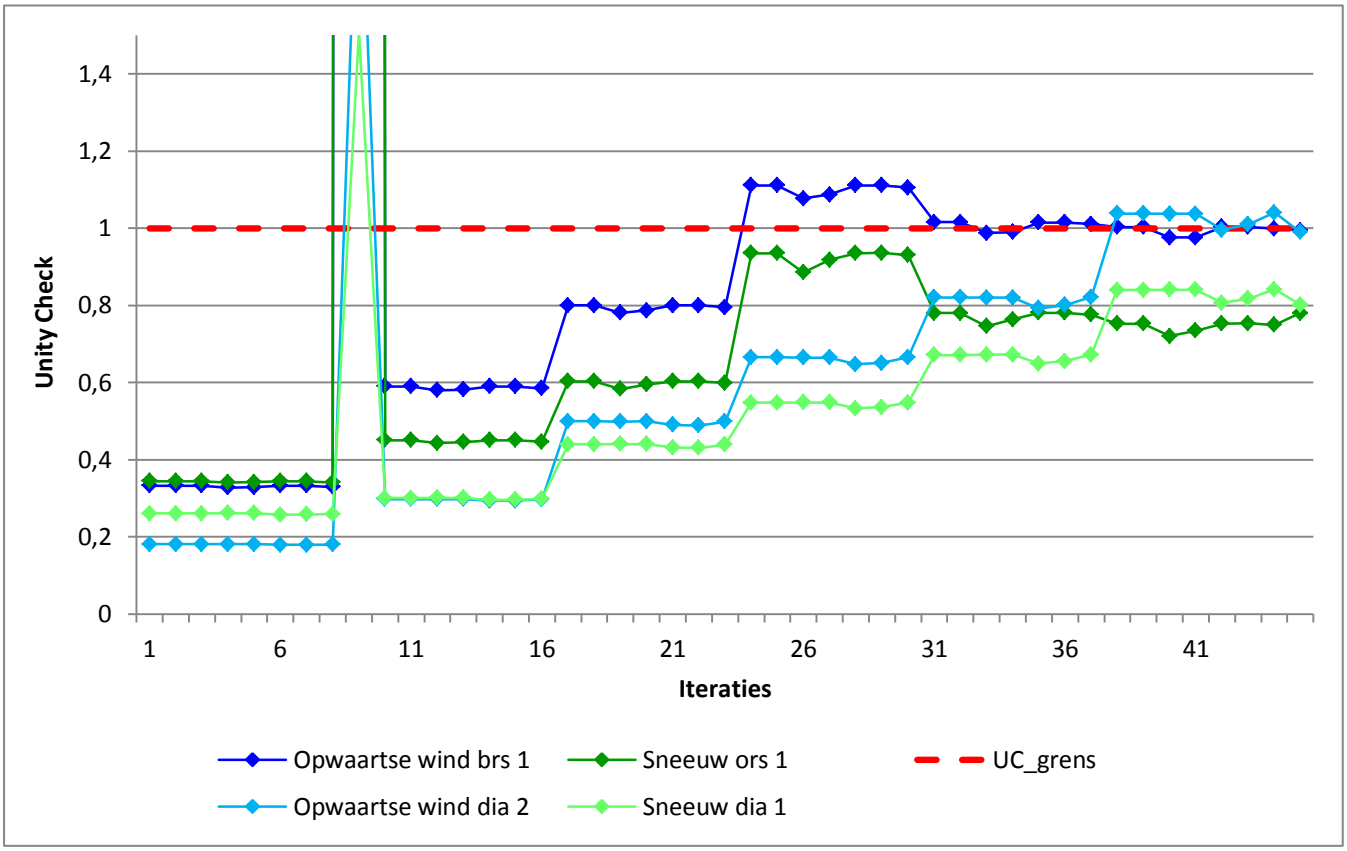
In Grafiek 9 is duidelijk zichtbaar dat de twee blauwe lijnen tegen het einde van het optimalisatieproces erg dicht langs de grens van de UC lopen. De blauwe lijnen horen bij de FC waarbij de opwaartse windbelasting als maatgevend is gekozen. Hoewel het in de grafiek niet duidelijk te zien is, kan uit Tabel 13 opgemaakt worden dat de donkerblauwe lijn het dichtst bij de grens eindigt. Hieruit kan geconcludeerd worden dat de UC voor de knikkracht bij belasting door de FC met opwaartse windbelasting bepalend is voor de dimensionering van de randstaven.

De dimensionering van de wandstaven wordt bepaald door de eisen die gesteld worden aan de verhoudingen van de staafafmetingen. Dit wordt inzichtelijk gemaakt in Grafiek 10 en Grafiek 11. In deze grafieken is het verloop van de d/t-verhouding respectievelijk de wanddikteverhouding weergegeven. In Grafiek 10 is te zien dat de groene lijn de grens dicht benadert en af en toe zelfs

overschrijdt. Deze groene lijn geeft de d/t-verhouding weer van de wandstaven. In Grafiek 11 is te zien dat ook de wanddikteverhouding om de grenswaarde heen beweegt.

Er is beredeneerd dat de randstaven gedimensioneerd worden op basis van de UC voor de knikkracht. De wanddikte van de wandstaven wordt vervolgens bepaald met behulp van de minimale waarde voor de wanddikteverhouding. De wanddikte van de randstaven dient minimaal 2 keer zo groot te zijn als die van de wandstaven. Aan de hand van de maximale verhouding van de diameter en de wanddikte kan vervolgens ook de diameter van de wandstaven bepaald worden. Hierbij mag de d/t-verhouding niet meer dan 50 bedragen.

Grafiek 9. Unity Check knikkracht van de zwaarst belaste randstaaf en diagonaal per FC (10 velden, Warren truss, systeemverbindingen)

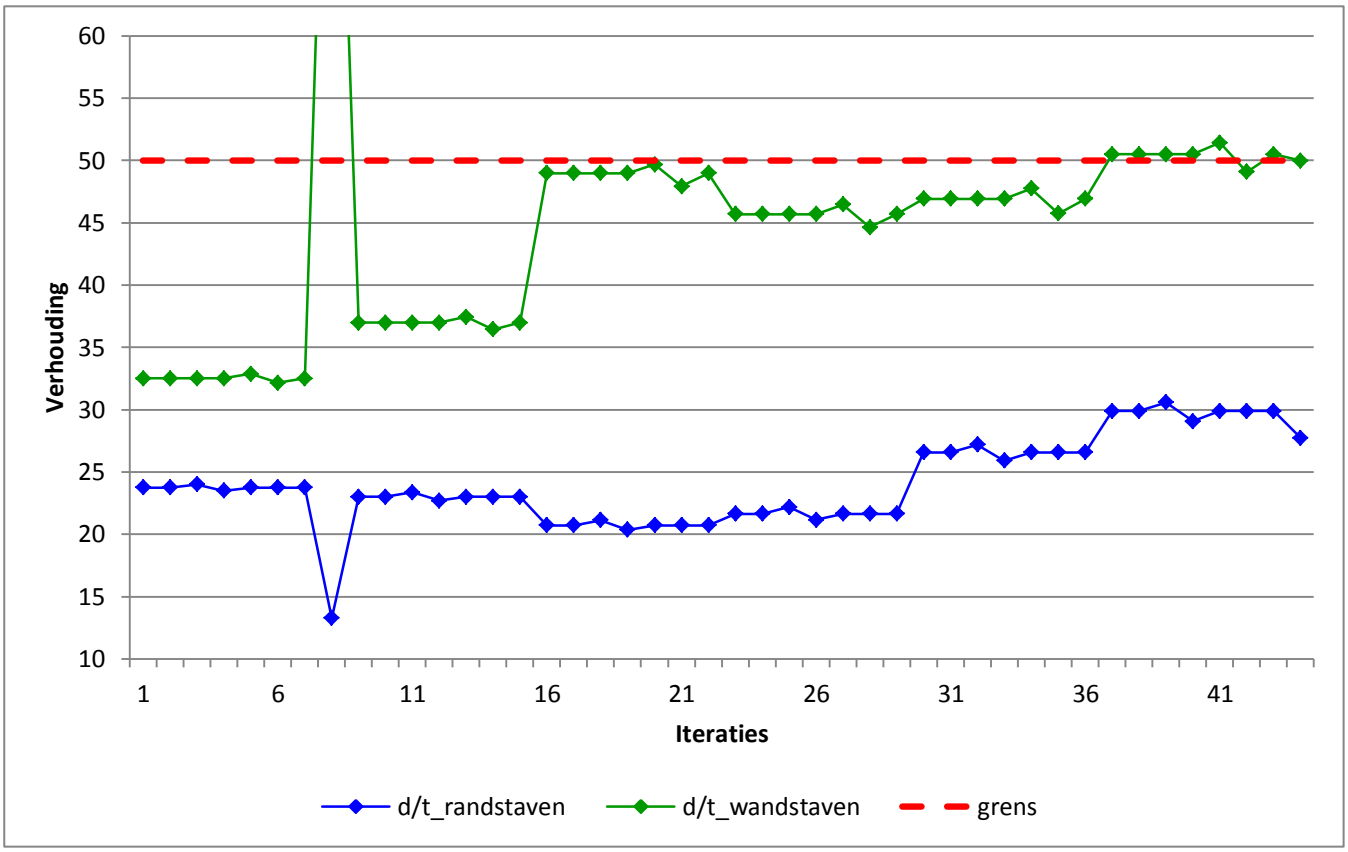


6.5 Invloed van verschillende startwaarden

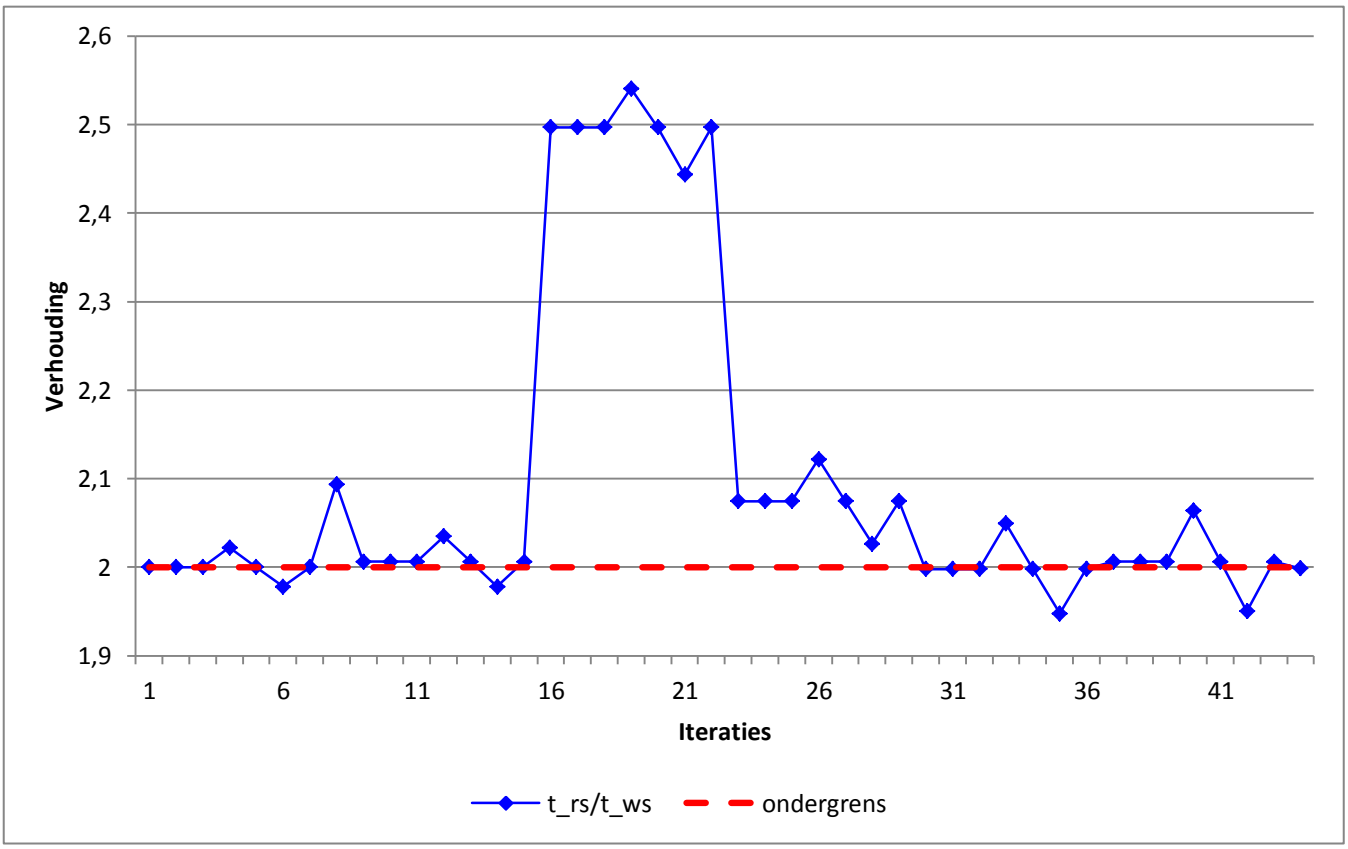
Er is onderzocht of verschillende startwaarden van de continue ontwerpvariabelen de uitkomst van het optimalisatie algoritme beïnvloeden. Het optimalisatie algoritme bepaalt voor alle mogelijke combinaties van de discrete variabelen het optimale ontwerp. Per combinatie zijn de kosten van de optimale ontwerpen die gevonden zijn voor twee verschillende sets van startwaarden, met elkaar vergeleken. Uit deze vergelijking blijkt dat het verschil in kosten gemiddeld 0,7 % bedraagt. Dit kan echter oplopen tot een verschil van 9 %. Het optimalisatie algoritme is ingesteld op een tolerantie van 1 %. Een kostenverschil dat groter is dan 1 % lijkt erop te wijzen dat er sprake is van een lokaal optimum. Er zijn immers, voor een specifieke combinatie van de discrete variabelen, twee verschillende *optimale* ontwerpen gevonden met een significant verschil in kosten.

Op basis van deze informatie kan echter niet met zekerheid vastgesteld worden of er sprake is van één of misschien zelfs meerdere lokale optima. Het zou interessant zijn om de resultaten van het continue optimalisatie algoritme dat bij dit onderzoek is toegepast, te vergelijken met die van een discreet optimalisatie algoritme. Dit is een interessant onderwerp voor een vervolgonderzoek.

Grafiek 10. Diameter/wanddikte-verhouding per staafgroep (10 velden, Warren truss, systeemverbindingen).



Grafiek 11. Wanddikteverhouding (10 velden, Warren truss, systeemverbindingen)



7 Conclusie

In dit hoofdstuk worden de vragen beantwoord die in de inleiding gesteld zijn. In de eerste paragraaf wordt de ontwerpvraag, waarvoor het computermodel is ontwikkeld, behandeld. Er wordt toegelicht welke conclusies met betrekking tot het ontwerp van de vakwerkligger getrokken kunnen worden op basis van de resultaten van het computermodel. Ook wordt het optimale ontwerp dat met dit model gevonden is, beschreven.

In de tweede paragraaf worden de twee deelvragen en de onderzoeksvraag beantwoord. Er wordt in twee delen antwoord gegeven op de onderzoeksvraag. In het eerste deel worden de beperkingen van het computermodel besproken waarna in het tweede deel de mogelijkheden worden beschreven.

7.1 De Ontwerpvraag

Aan de hand van een specifieke ontwerpvraag is de werking van het computermodel onderzocht. Deze ontwerpvraag luidt: “Wat is het optimale ontwerp voor een uitkragende, vlakke vakwerkligger die onderdeel uitmaakt van de overkapping van een tribune?”. Het antwoord op deze vraag is gevonden door het ontwikkelde computermodel te runnen en de resultaten te analyseren.

Op basis van deze resultaten zijn vier conclusies getrokken. Ten eerste is er een range bepaald voor het aantal velden dat het beste toegepast kan worden. Er is aangetoond dat er het beste gekozen kan worden uit 7 tot en met 11 velden. Binnen deze range is het verloop van de totale kosten uitgezet tegen het aantal velden erg vlak. Het kostenverschil per extra veld bedraagt gemiddeld slechts 1,8 %. Buiten de genoemde range nemen de totale kosten sneller toe. Het verschil per extra veld is gemiddeld 10 % en dit kan oplopen tot wel 20 %. Daarbij is aangetoond dat de kosten van de ontwerpen binnen deze range ten opzichte van het optimum gemiddeld 22 % hoger zijn met een maximum van 40 %. De kosten van de ontwerpen buiten de range van 7 tot en met 11 velden zijn ten opzichte van het optimum minimaal 25 % en gemiddeld 70 % hoger. Dit kan oplopen tot een verhoging van de kosten van meer dan 200 %!

Ten tweede is besproken dat de meest gunstige configuratie de Warren truss is. Er is echter gebleken dat de toepassing van een oneven aantal velden in combinatie met de configuratie Warren truss in sommige gevallen nadelig werkt en resulteert in hoge kosten. De voorkeur gaat dan ook uit naar het toepassen van de configuratie Warren truss in combinatie met een even aantal velden.

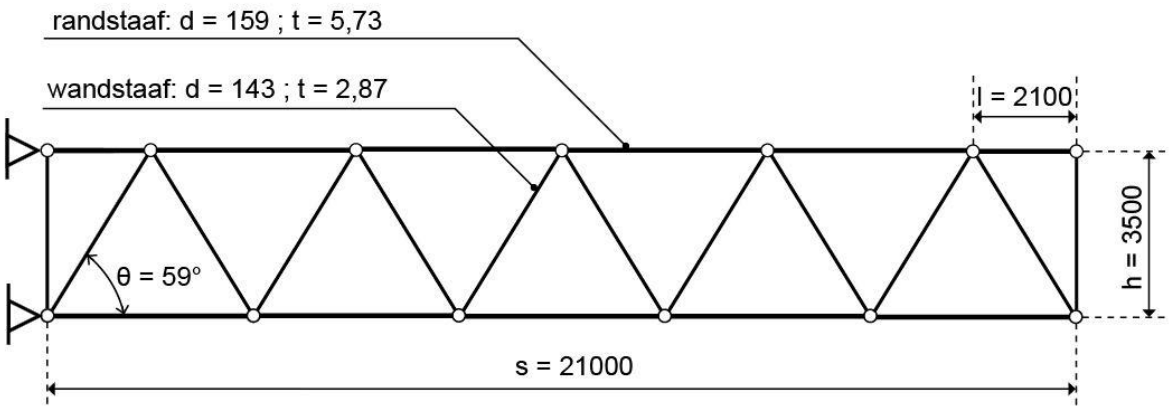
Ten derde is vastgesteld dat de fundamentele combinatie die de opwaartse windbelasting behandelt, de maatgevende FC is voor het ontwerp van de vakwerkligger. Dit betekent dat de sneeuwbelasting, die ook in het computermodel verwerkt is, geen invloed heeft op het ontwerp van de vakwerkligger.

Ten vierde is aangetoond dat de eis met betrekking tot de knikstabiliteit bepalend is voor de dimensionering van de randstaven. De dimensionering van de wandstaven wordt bepaald door de minimale wanddikteverhouding en door de maximale d/t-verhouding.

Het ontwerp van de vakwerkligger is naar stijfheid gecontroleerd aan de hand van een aangepaste stijfheidseis. Bij het opstellen van de aangepaste eis is aangegeven dat deze aanpak alleen geldig is zolang de stijfheidseis niet maatgevend is voor de dimensionering van het vakwerk. Deze voorwaarde is opgesteld om overdimensionering door toepassing van de te strenge stijfheidseis te voorkomen. Uit de vierde conclusie kan opgemaakt worden dat de dimensionering van de staven inderdaad niet bepaald wordt door de stijfheidseis. Dit betekent dat de wijze waarop de controle is uitgevoerd geldig is voor dit onderzoek.

Het optimale ontwerp dat het computermodel als antwoord geeft op de ontwerpvraag is weergegeven in Figuur 28. Voor de beschouwde ontwerpvraag is gebleken dat het beste een vakwerkligger toegepast kan worden met 10 velden, configuratie Warren en systeemverbindingen.

De waarden van de continue ontwerpvariabelen die bij dit optimale ontwerp horen zijn terug te vinden in de figuur.



Figuur 28. Het optimale ontwerp voor de beschouwde ontwerpvraag

7.2 Deelvragen & de Onderzoeksvraag

In deze paragraaf worden de twee deelvragen en de onderzoeksvraag beantwoord.

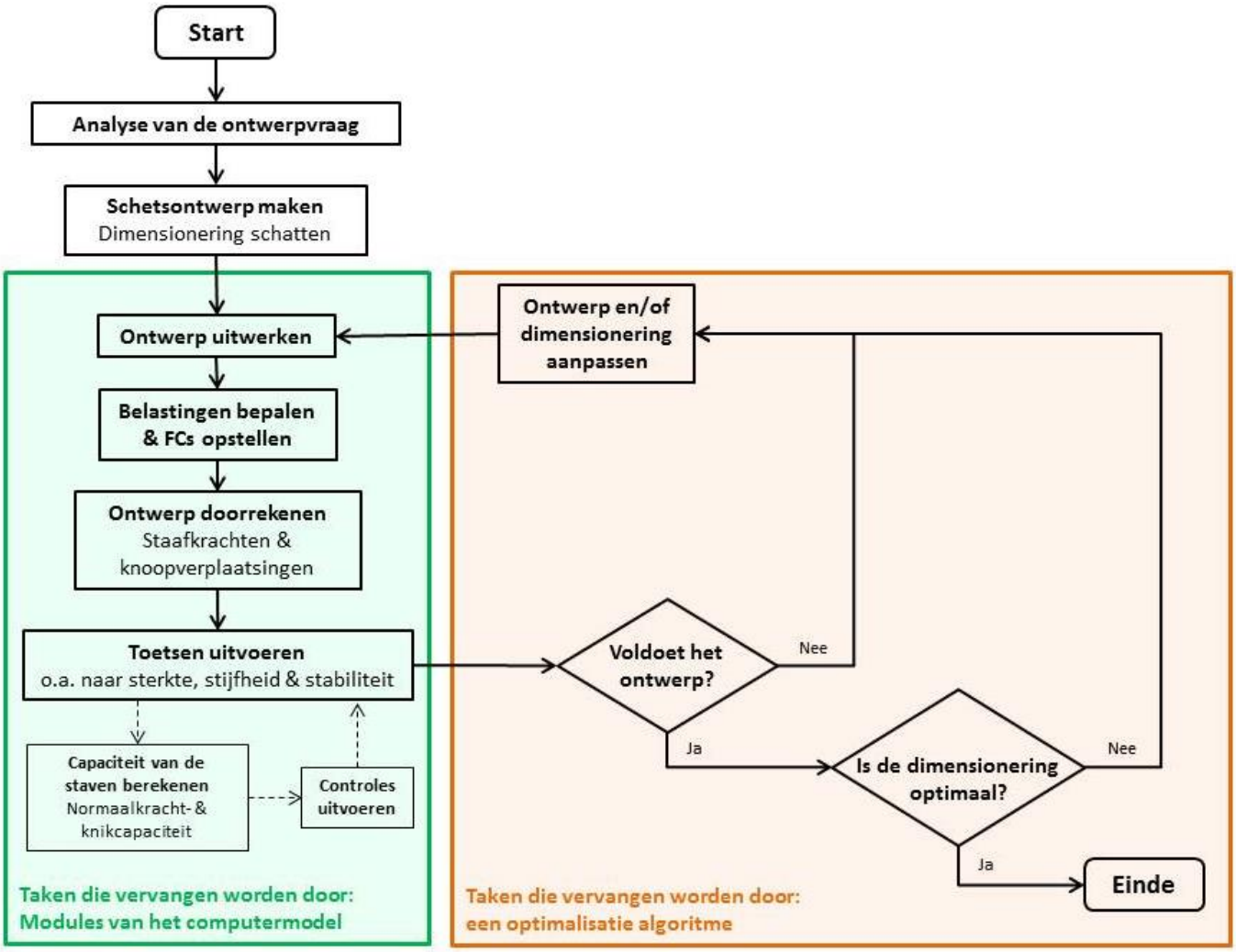
Deelvraag 1: Van Ingenieursaanpak naar Computermodel

"Welke modules voor het computermodel moeten er ontwikkeld worden om de stappen die een ingenieur doorloopt bij het ontwerpen van een vakwerkligger te automatiseren?"

Er is onderzocht hoe een ingenieur te werk gaat bij het ontwerpen van een vakwerkligger. Hierbij is vastgesteld dat een ingenieur tijdens dit ontwerpproces globaal gezien 7 stappen doorloopt. Deze stappen zijn weergegeven in het flow diagram van de ingenieursaanpak in Figuur 29. Aan de hand van dit flow diagram is de vertaalslag gemaakt van de ingenieursaanpak naar het computermodel. De twee kaders in Figuur 29 omvatten de stappen die door het computermodel geautomatiseerd worden. De eerste twee stappen van de ingenieursaanpak vallen buiten de kaders en worden zodoende niet door het computermodel uitgevoerd. Deze twee stappen zijn het analyseren van de ontwerpvraag en het maken van een schetsontwerp. Bij het analyseren van de ontwerpvraag worden de ontwerpgegevens opgesteld. Deze gegevens verschillen per ontwerpvraag en dienen aan het model meegegeven te worden als input. De stap waarbij het schetsontwerp wordt gemaakt, bepaalt in termen van het computermodel, het startpunt van het optimalisatieproces. Het optimalisatie algoritme kan niet zelf bepalen waar het moeten beginnen met optimaliseren. Informatie over dit startpunt dient dus, net als de ontwerpgegevens, als input aan het model meegegeven te worden.

De derde tot en met de zesde stap van de ingenieur vallen binnen het groene kader en zijn verwerkt in modules. De ontwerpmodule, de module ter bepaling van de belastingen en de FCs en de toetsingsmodule zijn hierbij zelf ontwikkeld. Voor de rekenmodule is gebruikt gemaakt van het programma 3D Truss dat de optredende staafkrachten en knoopverplaatsingen berekend.

Het oranje kader omvat de terugkoppeling, die het ontwerpproces een iteratief proces maakt. Er worden een paar vragen gesteld en afhankelijk van het antwoord wordt het ontwerp aangepast of wordt het proces beëindigd. Deze terugkoppelende stap wordt binnen het computermodel uitgevoerd door het optimalisatie algoritme. Het algoritme controleert of het ontwerp voldoet aan de eisen, ofwel de constraints en of het optimum gevonden is. Zolang het optimum niet gevonden is, vindt er een terugkoppeling plaats naar de ontwerpmodule, waarbij het algoritme een aangepaste set ontwerpvariabelen meegeeft als input.



Figuur 29. Flow diagram Ingenieursaanpak incl. vertaalslag naar Computermodel

Deelvraag 2: De keuze van de optimalisatie aanpak

"Welke optimalisatie aanpak is het meest geschikt voor het optimalisatievraagstuk van de vakwerkligger, zoals deze voor dit onderzoek beschouwd wordt?"

Het optimalisatievraagstuk dat behandeld is voor dit onderzoek, is een mixed continuous-discrete problem. Bij dit vraagstuk worden namelijk zowel continue als discrete ontwerpvariabelen beschouwd. Er is gekozen om het gemengde probleem op te splitsen in een continu hoofdprobleem en een discreet subprobleem, zodat de gradient-based optimization method toegepast kan worden.

Deze keuze is voor een groot deel gebaseerd op de beperkte duur van dit onderzoek, waardoor het is noodzakelijk is dat het optimum in relatief korte tijd bepaald kan worden. Uit de vergelijking van een viertal optimalisatie aanpakken is naar voren gekomen dat continue methoden, zoals de gradient-based optimization method, in de regel sneller het optimum kunnen vinden dan discrete methoden. Ook aangetoond is dat het optimalisatie algoritme scipy.optimize.minimize, dat werkt volgens de gradient-based optimization method, zeer goed te sturen is. Dit komt doordat het bij dit algoritme mogelijk is om onder andere bounds en constraints mee te geven en de tolerantie in te stellen. Door deze mogelijkheden kan het ontwerpgebied beperkt worden wat de rekentijd ten goede komt.

Voor het discrete subprobleem is gekozen om de brute force approach toe te passen. Deze aanpak is uitvoerbaar in beperkte tijd, omdat het aantal discrete ontwerpvariabelen gering is.

Onderzoeksvraag: Beperkingen van het computermodel

"Wat zijn de beperkingen van een computermodel dat zelfstandig bepaalt wat, economisch gezien, het optimale ontwerp is van een uitkragende, vlakke vakwerkligger, die onderdeel uitmaakt van de overkapping van een tribune?"

Uit dit onderzoek zijn zes beperkingen van het computermodel naar voren gekomen. De eerste beperking heeft te maken met het berekenen van de totale kosten. Er bestaat geen universele wijze waarop de kosten van de staven en de verbindingen berekend worden. Per aannemer verschilt deze aanpak. Daarbij kunnen de prijzen van de onderdelen in de tijd veranderen. Dit betekent dat het gedeelte van het computermodel dat de kosten berekent bij ieder gebruik gecontroleerd en eventueel aangepast dient te worden. Een volledige automatisering van de kostenbepaling is dan ook niet zonder meer mogelijk.

De tweede beperking heeft betrekking op de toepassingsmogelijkheden van het computermodel. Het model is ontwikkeld voor de ontwerpvraag van een uitkragende, vlakke vakwerkligger en is daarom alleen toepasbaar bij soortgelijke ontwerpvragen. Om de toepassingsmogelijkheden van het computermodel te vergroten, zijn er uitbreidingen en/of aanpassingen nodig. Zo kunnen er in het geval dat de ontwerpvraag, bijvoorbeeld, over een ligger op twee steunpunten gaat, configuraties toegevoegd worden die gespiegeld zijn in de middellijn. Ook indien de schaal van de ontwerpvraag sterk verschilt van die van de beschouwde vakwerkligger, zijn er aanpassingen nodig. In dat geval moet extra aandacht besteed worden aan het bijstellen van de startwaarden en de bounds van de continue ontwerpvariabelen.

Het feit dat het computermodel slechts een gering aantal ontwerpen kan aanleveren, wordt gezien als derde beperking. Het computermodel kan voor het optimale ontwerp alleen gebruik maken van de keuzes van de ontwerpvariabelen die zijn ingeprogrammeerd. Dit betekent dat indien er drie configuraties zijn opgenomen in het model, het optimale ontwerp altijd een van deze drie configuraties zal zijn. Dat het model zo werkt, is logisch, maar de beperking van de ontwerpvrijheid die hiermee gepaard gaat, blijft ongewenst. Er kunnen natuurlijk oneindig veel keuzes worden toegevoegd aan het model. Denk bijvoorbeeld aan andere configuraties of andere profielen. Het zou echter nog beter zijn als het model volledig zelfstandig zou kunnen bepalen waar de knooppunten en de staven het beste geplaatst kunnen worden en welke doorsnedevorm van de staven het gunstigste is. Zo'n zelfstandig werkend computermodel staat echter nog ver af van het huidige model. Het is ook de vraag of zo'n model ontwikkeld kan worden en betrouwbare resultaten kan leveren. Dit is een interessant onderwerp voor een vervolgonderzoek.

Aan de derde beperking kleeft een vierde beperking. De derde beperking geeft aan dat het computermodel begrensd wordt door de keuzemogelijkheden die zijn ingeprogrammeerd. Indien een architect of constructeur gebruik kan maken van zo'n computermodel om in beperkte tijd en voor relatief lage kosten het optimale ontwerp te bepalen, is het denkbaar dat creativiteit en innovatie worden geremd. Dit wordt gezien als de vierde beperking. Het ontwikkelen van een ontwerp dat niet in het model is opgenomen, vraagt namelijk om extra handelingen. Het computermodel zal uitgebreid moeten worden of een constructeur zal volgens de 'ouderwetse' ingenieursaanpak het ontwerp moeten maken. In beide gevallen leidt dit tot een langer ontwerpproces en tot hogere kosten.

De laatste twee beperkingen hebben te maken met de toegepaste optimalisatie aanpak. De gradient-based optimization method brengt als risico met zich mee dat er een lokaal optimum gevonden wordt in plaats van het globale optimum. Dit betekent dat het onzeker blijft of het optimale ontwerp dat met het computermodel gevonden wordt daadwerkelijk het optimum van optimalisatievraagstuk is. Daarbij is uit dit onderzoek gebleken dat het toegepaste optimalisatie algoritme niet altijd om kan gaan met een startpunt dat buiten het ontwerpgebied ligt. Dit betekent dat het van belang is dat vooraf gecontroleerd wordt of het startpunt dat wordt meegegeven aan het algortime voldoet aan de constraints. Aangezien dit van de ontwerpvraag afhangt, zal hier bij het huidige model altijd aandacht aan besteed moeten worden door de gebruiker.

Onderzoeksvraag: Mogelijkheden van het computermodel

"Wat zijn de mogelijkheden van een computermodel dat zelfstandig bepaalt wat, economisch gezien, het optimale ontwerp is van een uitkragende, vlakke vakwerkligger, die onderdeel uitmaakt van de overkapping van een tribune?"

Naast de beschreven beperkingen zijn er ook veel mogelijkheden van het computermodel te benoemen. In principe is alles wat niet als beperking genoemd is een mogelijkheid van het model. Vier van deze mogelijkheden worden hier toegelicht.

Allereerst maakt het gebruik van het computermodel het mogelijk om *human error* te voorkomen. Mensen maken fouten. Zeker in het geval dat berekeningen meerdere malen achter elkaar uitgevoerd moeten worden, is de kans aanwezig dat er ergens iets niet helemaal goed gaat. Door een computermodel deze berekeningen uit te laten voeren, wordt dit risico op fouten weggenomen. Hierbij is het uiteraard van belang dat het model bij het ontwikkelen gevalideerd wordt.

De tweede mogelijkheid heeft betrekking op de informatie die het computermodel aanbiedt. Aan de hand van een grafiek met het verloop van de totale kosten uitgezet tegen het aantal velden, is aangetoond dat er een significant verschil is tussen de kosten van het optimale ontwerp en van overige ontwerpen. Binnen de range van 7 tot en met 11 velden bedraagt dit verschil gemiddeld 22 %. Buiten deze range is het gemiddelde verschil 70 %. Een kostenbesparing van 70 % komt neer op een besparing van ruim 7500 euro per vakwerkligger. Indien er meerdere vakwerkliggers worden toegepast, wat vrijwel altijd het geval is, kan dit bedrag snel oplopen. De informatie die het computermodel levert over het optimale ontwerp kan zodoende erg waardevol zijn.

De derde mogelijkheid die besproken wordt, hangt samen met de vierde beperking. Als beperking werd genoemd dat creativiteit en innovatie geremd kunnen worden door de beschikbaarheid van een computermodel voor de optimalisatie van een vakwerkligger. Hier staat tegenover dat door gebruik te maken van zo'n computermodel het optimalisatieproces sneller verloopt. Doordat er minder tijd nodig is om het ontwerp te optimaliseren, is er ruimte om nieuwe mogelijkheden, zoals nieuwe configuraties, te ontwerpen en te onderzoeken. Dit betekent dus dat de beschikbaarheid van een computermodel juist creativiteit en innovatie mogelijk maakt. Het zal per situatie verschillen of de beperking of de mogelijkheid overheerst. Ook is dit afhankelijk van het budget en van het tijdsbestek waarbinnen het ontwerp gemaakt dient te worden.

Tot slot is het doel van het computermodel het bepalen van het ontwerp waarvoor de kosten minimaal zijn. De meest voor de hand liggende mogelijkheid is dan ook het besparen van kosten. De voorgaande mogelijkheden resulteren allemaal in zekere zin in kostenbesparing en/of kwaliteitsverhoging. Waarbij ook kwaliteitsverhoging opgevat kan worden als kostenbesparing. Er wordt tenslotte iets ontworpen met een hogere waarde voor dezelfde prijs.

Op basis van de mogelijkheden en beperkingen, die besproken zijn, kan geconcludeerd worden dat een computermodel dat zelfstandig het optimale ontwerp bepaalt voor een specifieke ontwerpvraag erg waardevol kan zijn.

In het volgende hoofdstuk worden suggesties gedaan over de mogelijke uitbreidingen, waardoor de toepassingsmogelijkheden van het huidige model vergroot kunnen worden. Ook worden aanbevelingen gedaan voor vervolgonderzoek.

8 Aanbevelingen & Uitbreidingsmogelijkheden

Op basis van het onderzoek dat gedaan is, worden vijf aanbevelingen gedaan over manieren waarop het computermodel verbeterd kan worden. Daarnaast worden suggesties gedaan over uitbreidingen voor het computermodel. Door het computermodel uit te breiden worden toepassingsmogelijkheden vergroot.

8.1 Aanbevelingen

De eerste aanbeveling heeft betrekking op de optimalisatie aanpak. Binnen dit onderzoek is gebruik gemaakt van een algoritme dat werkt volgens de gradient-based optimization method. Aan deze methode kleven enkele nadelen. Ten eerste is de gradient-based optimization method alleen geschikt voor continue optimalisatie problemen. Het optimalisatievraagstuk van de vakwerkligger is echter een gemengd continu-discreet probleem en bestaat dus uit zowel continue als discrete ontwerpvariabelen. De discrete ontwerpvariabelen dienen volgens een andere aanpak verwerkt te worden in het computermodel. Binnen dit onderzoek is voor deze variabelen gebruik gemaakt van de brute force aanpak. De splitsing van het optimalisatievraagstuk heeft als gevolg dat er twee aanpakken geïmplementeerd moeten worden in het model. Dit resulteert in meer werk en mogelijk een langere rekentijd dan wanneer voor het gehele probleem een enkele aanpak toegepast kan worden.

Het tweede nadeel van de gradient-based optimization method is het risico dat er een lokaal optimum gevonden wordt in plaats van het globale optimum. Van de uitkomst die met deze methode gevonden wordt, kan dan ook niet met zekerheid gezegd worden dat het daadwerkelijk het optimale ontwerp is.

Ondanks de nadelen van de gradient-based optimization method is voor dit onderzoek toch gekozen deze methode toe te passen, omdat de methode snel het optimum kan bepalen en goed te sturen is. Er wordt echter aanbevolen om bij eventueel vervolg onderzoek naar de optimalisatie van vakwerkconstructies gebruik te maken van een genetic search algorithm. Deze optimalisatie aanpak is goed in het bepalen van het globale optimum en kan daarnaast omgaan met zowel continue als discrete ontwerpvariabelen.

De tweede aanbeveling komt voort uit de beperking van het computermodel met betrekking tot de kostenbepaling. Er wordt aangeraden om te onderzoeken hoe met deze beperking omgegaan kan worden. Zijn er bijvoorbeeld een aantal veel gebruikte manieren waarop de kosten van de onderdelen berekend worden? Zo ja, volstaat het dan om deze manieren in te programmeren en de gebruiker vooraf te laten kiezen welke manier toegepast wordt?

Ten derde wordt aanbevolen te onderzoeken of er een computermodel ontwikkeld kan worden dat volledig zelfstandig kan bepalen waar de knooppunten en de staven geplaatst moeten worden. In tegenstelling tot het huidige model, dat bij de optimalisatie moet kiezen uit een beperkt aantal voorgeprogrammeerde configuraties, zou zo'n model zelfstandig de configuratie kunnen *ontwerpen*. Hierdoor zou voor iedere ontwerpvraag een configuratie ontwikkeld kunnen worden die volledig aansluit op de situatie. Op die manier zouden meer efficiënte vakwerkconstructies ontworpen kunnen worden. Het is hierbij de vraag of zo'n zelfstandig werkend computermodel ontwikkeld kan worden en of het betrouwbare resultaten kan leveren.

Ten vierde wordt aanbevolen een sciptje te schrijven waarmee de keuze van het startpunt voor het optimalisatie algoritme geautomatiseerd kan worden. Bij de beschrijving van de beperkingen van het huidige model is toegelicht dat het noodzakelijk is dat het startpunt binnen het ontwerpgebied ligt en dus voldoet aan de constraints. Het ontwerpgebied verschilt per ontwerpvraag, waardoor bij het huidige model handmatig het startpunt bepaald moet worden. Door deze handeling te automatiseren, hoeft de gebruiker van het model hier geen aandacht aan te besteden en wordt de gebruiksvriendelijkheid van het computermodel verhoogd.

De vijfde aanbeveling is het organiseren van een wedstrijd tussen een ervaren constructeur en het ontwikkelde computermodel. Door zowel de benodigde tijd als de manier van werken als de

'optimale' ontwerpen met elkaar te vergelijken, kan ingeschat worden waar het computermodel sterk in is en waarin het tekort schiet. Aan de hand van de uitkomst van de wedstrijd kan het computermodel verder aangepast en verbeterd worden.

8.2 Uitbreidingsmogelijkheden

De toepassingsmogelijkheden en de kwaliteit van het computermodel kunnen vergroot worden door de aanbevelingen uit te voeren. Daarnaast zijn er ook vele uitbreidingsmogelijkheden waarmee dit doel bereikt kan worden. Er worden enkele suggesties opgesomd:

- Beschouw de totale constructie/samenwerking van de vakwerkliggers i.p.v. een enkele ligger
  Stel hierbij (onder andere) de volgende vragen:
  - Hoe worden de vakwerkligers met elkaar verbonden?
  - Hoe wordt de zijdelingse ondersteuning van de onderrandstaven geregeld?
  - Mag de bouwwerkfactor ($c_s c_d$) gereduceerd worden?
- Zoom in op de verbindingen van de staven van het vakwerk
  Beschouw hierbij bijvoorbeeld eisen met betrekking tot de gap en overlap van de verbindingen
- Onderzoek wat er verandert indien het vakwerk zich niet als *zuiver* vakwerk gedraagd
- Configuraties toevoegen (niet alleen voor uitkragende liggers, maar bijvoorbeeld ook voor liggers op twee steunpunten)
  Denk hierbij aan de toepassing van:
  - Configuraties voor verschillende mechanicaschema's
  - Dubbele diagonalen
  - Liggers onder een helling met parallele randstaven
  - Liggers met randstaven die niet parallel lopen
  - Boogliggers met parallele randstaven
  - ...
- Dakhelling als ontwerpvariabele toevoegen (mits de dakhelling gelijk wordt gesteld aan de helling van die ligger; zie voorgaande suggesties)
- Uitbreiding van de module waarmee de windbelasting wordt berekend:
  Uitbreiding door het toevoegen van:
  - Verschillende dakvormen (bijvoorbeeld met een knik)
  - Hoek alfa als ontwerpvariabelen van het optimalisatievraagstuk
  - Grotere range van de gebouwhoogte
  - Windbelasting op de zijgevels (indien de hele constructie wordt beschouwd)
  - Grotere range voor het aantal velden van de vakwerkligger
- Profielen toevoegen
- Meer staafgroepen definiëren:
  De staafafmetingen worden per staafgroep geoptimaliseerd. Voor dit onderzoek zijn de staafgroepen randstaven en wandstaven gehanteerd. Het toepassen van meerdere staafgroepen kan op verschillende manieren:
  - Ligger opsplitsen in delen en per deel rand- en wandstaven beschouwen
  - Onderscheid maken tussen boven- en onderrandstaven
  - Onderscheid maken tussen staanders en diagonalen
  - ...
- ...

9 Bronvermelding

Abspoel, R., De Vries, P.A. & Bijlaard, F.S.K. (2013). CTB2220 Beton- en Staalconstructies: *Staalconstructies*. [internet versie] TU Delft Faculteit CiTG. Beschikbaar op: https://blackboard.tudelft.nl/webapps/blackboard/content/listContent.jsp?course_id=_53368_1&content_id=_2519766_1

Animaatjes (n.d.). *Stationshal* [digitale foto] Beschikbaar op: http://www.animaatjes.nl/trein-wallpapers-545767

Architizer (2011). Project: *Al Shaqab Equestrian Performance Arena.* [digitale foto] Beschikbaar op: http://architizer.com/projects/al-shaqab-equestrian-performance-arena/

Bartels (2015). *Go Ahead Eagles Stadion.* [online] Beschikbaar op: http://www.bartels-global.com/nl/projecten/go-ahead-eagles-stadion

Bloemers, A.J.W. (2014). Bachelor Eindproject: *Model om de staafkrachten te benaderen van een 3D vakwerkboog.* [internet versie] TU Delft Faculteit CiTG. Beschikbaar op: http://homepage.tudelft.nl/p3r3s/BSc_projects/BSc_projects.html

Chamoret, D., Qiu, K. & Domaszewski, M. (2008). Optimization of truss structures by a stochastic method. *International Journal for Simulation and Multidisciplinary Design Optimization,* 7(3), 321-325. Beschikbaar op: http://www.ijsmdo.org/

De Architect (2012). Galerij: *EK 2012 – Arena Lviv, Oekraïne.* [digitale foto] Beschikbaar op: http://www.dearchitect.nl/projecten/2012/22/ek-2012---lviv-oekraine/galerijen/impressies-tekeningen-albert-wimmer.html?picIndex=6&picName=def-doorsnede-tribune-bezoekers-wimmer.jpg#foto

De Stentor (2015). *GA Eagles dubt over naam Leo Halle voor nieuwe tribune (poll).* [digitale foto] Beschikbaar op: http://www.destentor.nl/sport/voetbal/ga-eagles/ga-eagles-dubt-over-naam-leo-halle-voor-nieuwe-tribune-poll-1.5150771

De Vries, P.A., Fennis, S.A.A.M. & Pasterkamp, S. & (2013). CTB2220/CTB2320: *Veiligheid van constructies en Belastingen op constructies.*TU Delft Faculteit CiTG. Zoetermeer, Nederland: Bouwen met Staal.

Dolsma, A. (2015). Staal gaat hét maken: *Elfstedenhal - Leeuwarden.* Bouwen met Staal [digitale foto] Beschikbaar op: http://www.staalmakers.nl/alle-projecten/53/Elfstedenhal,+Leeuwarden/

Elferink, B. (2010). Bachelor Eindproject: *De optimale koepel.* [internet versie] TU Delft Faculteit CiTG. Beschikbaar op: http://homepage.tudelft.nl/p3r3s/BSc_projects/BSc_projects.html

Halkes, J. (2015). Metro: *NS sluit ook poortjes op Station Blaak Rotterdam.* [digitale foto] Beschikbaar op: http://www.metronieuws.nl/rotterdam/2015/05/ns-sluit-ook-poortjes-op-station-blaak-rotterdam

Hoogendoorn, B. (2015). Bachelor Eindproject: *Optimalisatie van vakwerkbogen met 3Dtruss.* [internet versie] TU Delft Faculteit CiTG. Beschikbaar op: http://homepage.tudelft.nl/p3r3s/BSc_projects/BSc_projects.html

Hu, X. (2006). Particle Swarm Optimization: *Introduction*. [online] Beschikbaar op: http://www.swarmintelligence.org/

Hunting, H. (2003). *Shelter.* [digitale foto] beschikbaar op: http://radio-weblogs.com/0119080/stories/2003/03/11/galleryUrbanNomadics.html

Jacobson, L. (2012). The project spot: *Creating a genetic algorithm for beginners.* [online] Beschikbaar op: http://www.theprojectspot.com/tutorial-post/creating-a-genetic-algorithm-for-beginners/3

Koning, L. (2015). Bachelor Eindproject: *Vakwerkkoepels, Optimalisatie kosten en ECF*. [internet versie] TU Delft Faculteit CiTG. Verkregen via begeleider R. Abspoel.

Lambert Engineering (2015). *Het skelet van een parel – De structuur van de Ghelamco arena.* [digitale foto] Beschikbaar op: https://medium.com/@lambengineering/het-skelet-van-een-parel-de-structuur-van-de-ghelamco-arena-3d8a9f13e04d#.njt5s485k

McCullock, J. (2012). Home: Al main: PSO: *Particla Swarm Optimization.* [online] Beschikbaar op: http://mnemstudio.org/particle-swarm-introduction.htm

MSP Dak en Wand (2015). *Leo Halle Tribune – Hoofdtribune GAE "De Adelaarshorst".* [online] Beschikbaar op: http://msp-dakenwand.nl/product/hoofdtribune-de-adelaarshorst/

Nijsse, R. (2012). BK4043 Draagconstructies 2: *Dictaat Draagconstructies II*. [internet versie] TU Delft Faculteit Bouwkunde. Beschikbaar op: http://wiki.bk.tudelft.nl/mw_bk-wiki/images/c/c2/Draagconstructies_II.pdf

Octatube, (n.d.). Systemen & Oplossingen: *Staalconstructies.* [digitale foto] Beschikbaar op: http://www.octatube.nl/systemen-oplossingen/3/staalconstructies/

OpenBuilding (n.d.). *Al Shaqab Equestrian Arena.* [digitale foto] Beschikbaar op: http://openbuildings.com/buildings/al-shaqab-equestrian-arena-profile-40484

Parkinson, A.R., Balling, R.J. & Hedengren, J.D. (2013). *Optimization Methods for Engineering Design: Applications and Theory.* [internet versie] Brigham Young University. Beschikbaar op: http://apmonitor.com/me575/uploads/Main/optimization_book.pdf

Prandi, J. (n.d.). *Maior ponte em arco do mundo (De grootste boogbrug ter wereld).* [digitale foto] Beschikbaar op: http://gigantesdomundo.blogspot.nl/2014/04/maior-ponte-em-arco-do-mundo.html

Raaij, B.P.M., Soons, F.A.M. & Wagemans, L.A.G. (2014). *Quick Reference*. TU Delft Faculteit CiTG. Delft, Nederland: Section Structural and Building Engineering.

Rajeev, S. & Krishnamoorthy, C.S. (1997). *Genetic algorithms-based methodologies for design optimization of trusses.* [internet versie] American Society of Civil Engineers. Beschikbaar op: http://ascelibrary.org/doi/abs/10.1061/(ASCE)0733-9445(1997)123:3(350)

Riemens, K. (2011). Bachelor Eindproject: *De Optimale Koepel.* [internet versie] TU Delft Faculteit CiTG. Beschikbaar op: http://homepage.tudelft.nl/p3r3s/BSc_projects/BSc_projects.html

RTV Oost (2015). *Nieuwe overdekte tribunes in Adelaarshorst hielden fans GA Eagles niet droog.* [online] Beschikbaar op: http://www.rtvoost.nl/nieuws/default.aspx?nid=224226&_ga=1.70420944.1783824223.1473688106

SciPy.org (2016a). Optimization and root finding (scipy.optimize): *Optimization.* [online] The SciPy community. Beschikbaar op: https://docs.scipy.org/doc/scipy-0.18.1/reference/optimize.html

SciPy.org (2016b). *Scipy.optimize.differential_evolution* [online] The SciPy community. Beschikbaar op: https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.optimize.differential_evolution.html#scipy.optimize.differential_evolution

SciPy.org (2016c). *Scipy.optimize.minimize* [online] The SciPy community. Beschikbaar op: https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.optimize.minimize.html#scipy.optimize.minimize

Staalprijzen (2015). *Buizen staalprijzen.* [internet versie] Breedveld Staal. Beschikbaar op: http://staalprijzen.nl/buizen

Stolpe, M. (2015). Truss optimization with discrete design variables: a critical review. *Structural and Multidisciplinary Optimization,* 53(2), 349-374. Beschikbaar op: http://link.springer.com/article/10.1007/s00158-015-1333-x

Straathof, M.H., Van Tooren, M.J.L., Voskuijl, M. & Koren, B. (2008). *Aerodynamic shape parameterisation and optimisation of novel configurations.* [internet versie] Tu Delft Faculteit Aerospace Engineering. Beschikbaar op: http://repository.tudelft.nl/islandora/object/uuid:107c7106-4fc7-4427-8d6d-7e56a9ec3d79?collection=research

Van der Stap, E. (2014). Bachelor Eindproject: *Ontwerpformules voor ruimtelijke vakwerken.* [internet versie] TU Delft Faculteit CiTG. Beschikbaar op: http://homepage.tudelft.nl/p3r3s/BSc_projects/BSc_projects.html

Varoquaux, G. (n.d.). Scipy lectures: *2.7. Mathematical optimization: finding minima of functions.* [online] Beschikbaar op: http://www.scipy-lectures.org/advanced/mathematical_optimization/

Wardenier, G.J., Packer, J.A., Zhao, X.-L. & Van der Vegte, J. (2010). Bouwen met Staal: *Hollow sections in structural applications*. Zwitserland: CIDECT.

Welleman, H. (2007). CTB2210 Constructiemechanica 3: *Introductie verplaatsingenmethode*. [internet versie] TU Delft Faculteit CiTG. Beschikbaar op: http://icozct.tudelft.nl/TUD_CT/CT2031/collegestof/verplaatsingenmethode/

Wu, S. & Chow, P. (1993). *Integrated discrete and configuration optimization of trusses using genetic algorithms.* [internet versie] Department of Mechanical Engineering, National Sun Yat-Sen Unitversity, Republic of China. Beschikbaar op: http://www.sciencedirect.com/science/article/pii/0045794994004264

10 Bijlagen

A. Berekening belastingen per FC

In deze bijlage wordt uiteengezet hoe de grootte van de belastingen is berekend. Deze informatie is nodig om te kunnen bepalen welke FC naar verwachting maatgevend zal zijn voor het optimalisatievraagstuk van dit onderzoek (zie paragraaf 3.3.4).

Bij deze berekening van de grootte van debelastingen is uitgegaan van een hart-op-hart-afstand (a) van 8,0 m. De belastingen zijn uitgerekend per strekkende meter van de ligger. De gebruikte formules en waarden van de parameters zijn besproken in paragraaf 2.4 respectievelijk 3.3.

<u>Permanente belasting:</u>

Belasting ten gevolge van de dakbedekking

F_dak = gewicht per m^2 * a [kN/m]

F_dak = 0,15 * 8,0 = **1,2 kN/m**

<u>Variabele belastinig:</u>

Windbelasting:

$F\_wind = c_s c_d * c_f * q_p(z_e) * a$ [kN/m]

Maximale windbelasting wordt gevonden bij een zo groot mogelijke waarde van de bouwwerkhoogte z_e. De dakhelling α staat bij dit onderzoek vast en heeft een waarde van 0°.

De waarden van de variabelen zijn hiermee als volgt:

$z_{e,max}$ = 20,0 m → $q_p(20)$ = 0,74

α = 0° → c_f verschilt per zone van het dak

$F\_wind = 1,0 * c_f * 0,74 * 8,0 = 5,92 * c_f$ kN/m

- Neerwaartse windbelasting:
  - Zone A: F_wind = 5,92 * 0,5 = **2,96 kN/m**
  - Zone C: F_wind = 5,92 * 1,1 = **6,51 kN/m**
- Opwaartse windbelasting:
  - Zone A: F_wind = 5,92 * -1,5 = **-8,88 kN/m**
  - Zone C: F_wind = 5,92 * -2,2 = **-13,02 kN/m**

Sneeuwbelasting:

$F\_sneew = \mu_i * C_e * C_t * s_k * a$ [kN/m]

F_sneew = 0,8 * 1,0 * 1,0 * 0,7 * 8,0 = **4,48 kN/m**

Belasting door goederen & personen:

- Gelijkmatige verdeeld:
  - q_k = 1,0 kN/m^2 op een oppervlak van 10,0 m^2
- Puntlast:
  - Q_k = 1,5 kN op een oppervlak van 0,1 x 0,1 m^2

B. Aanpassing stijfheidseis: bepaling gewogen veiligheidsfactor

Bij dit onderzoek is gebruik gemaakt van een aangepaste stijfheidseis. Deze aanpassing is noodzakelijk vanwege de toepassing van de veiligheidsfactoren bij de berekening van de verplaatsingen. Doordat de verplaatsingen berekend zijn mét toepassing van de veiligheidsfactoren is de 'normale' stijfheidseis te streng. Er wordt gebruik gemaakt van een gewogen veiligheidsfactor, die de maximaal toegestane verplaatsing verhoogd. In deze bijlage wordt beschreven hoe de grootte van deze gewogen veiligheidsfactor wordt bepaald.

De grootte van de gewogen veiligheidsfactor is afhankelijk van de verhouding tussen de permanente en de variabele belasting. De verhouding tussen de permanente en de variabele belasting verschilt per fundamentele belastingcombinatie. Voor de twee FCs die in het model opgenomen zijn, zijn deze verhoudingen bepaald. Hierbij is gebruik gemaakt van de grootte van de belastingen die in bijlage A zijn berekend. De windbelasting is omgerekend naar een gemiddelde belasting per strekkende meter ligger, zie Figuur 30. Voor de belasting door het eigen gewicht van de vakwerkligger is, met behulp van het computermodel, een schatting gemaakt van het gewicht per strekkende meter. Aangezien het eigen gewicht slechts een klein deel uitmaakt van de totale permanente belasting, is een schatting van deze belasting in dit geval nauwkeurig genoeg.

<u>Permanente belasting</u>

Veiligheidsfactor: 0,9 / 1,3

- Dakbedekking: 0,15 * 8,0 = 1,2 kN/m
- Eigen gewicht: circa. 0,07 á 0,09 kN/m
- Totale permanente belasting: **1,29 kN/m**

<u>Variabele belasting</u>

Veiligheidsfactor: 1,65

- Gemiddelde opwaartse windbelasting (max):

$$\frac{2 * 13{,}0 + 8 * 8{,}9}{10} = \mathbf{9{,}72\ kN/m}$$

- Sneeuwbelasting: **4,48 kN/m**

<u>Verhouding per FC:</u>

- FCII.a Opwaartse windbelasting (max)

  1,29 : 9,7 → 1 : 7,53

  Gewogen veiligheidsfactor:

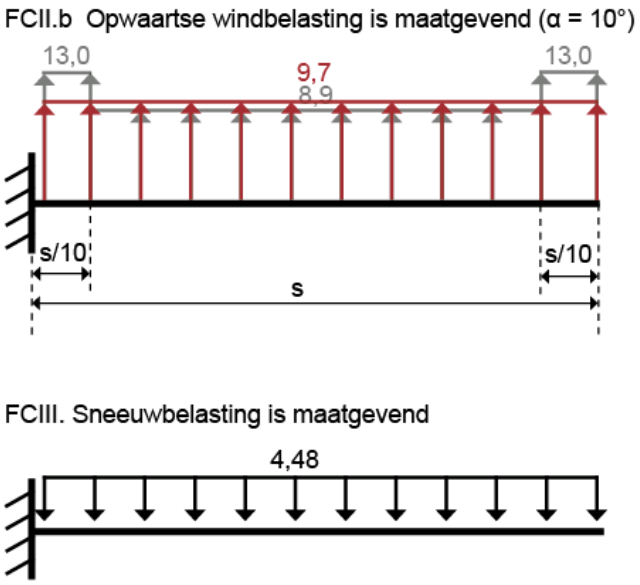
$$\frac{0{,}9 + 7{,}53 * 1{,}65}{8{,}53} = \mathbf{1{,}56}$$

- FCIII Sneeuwbelasting

  1,29 : 4,48 → 1 : 3,47

  Gewogen veiligheidsfactor:

$$\frac{1{,}3 + 3{,}47 * 1{,}65}{4{,}47} = \mathbf{1{,}57}$$



Figuur 30. (Gemiddelde) Variable belasting per FC

De gewogen veiligheidsfactor is minimaal 1,56. Er wordt gekozen om de minimale waarde toe te passen om te voorkomen dat de stijfheidseis in bepaalde gevallen te soepel wordt. De stijfheidseis wordt dus aangepast met de **gewogen veiligheidsfactor 1,56.**

De aangepaste eis voor de verticale verplaatsing wordt daarmee:

$$\mathbf{w_{aangepast} \leq 1{,}56 * 0{,}004 * L_{rep}}$$

C. Validatie van het eerste model (2D)

In deze bijlage wordt de eerste versie van het computermodel beschreven. Het eerste versimpelde model dat gemaakt is, beschouwt een vlakke vakwerkligger met een enkel veld. De input die voor dit model nodig is en de beperkingen die opgesteld zijn, zijn als volgt:

Beperkingen:

- 2D-vakwerk bestaande uit onder- en bovenrrandstaven, staanders en diagonalen.
- Maximaal 1 type diagonaal; keuze uit stijgend of dalend
- Willekeurige afmetingen voor de staafgroepen (niet variabel)
- Alleen belasting in y-richting (willekeurige grootte, neerwaarts)
- Permanente belasting grijpt aan in alle knooppunten van de bovenrandstaaf, waarbij de twee buitenste knopen belast worden door ½ F en de overige knopen door F

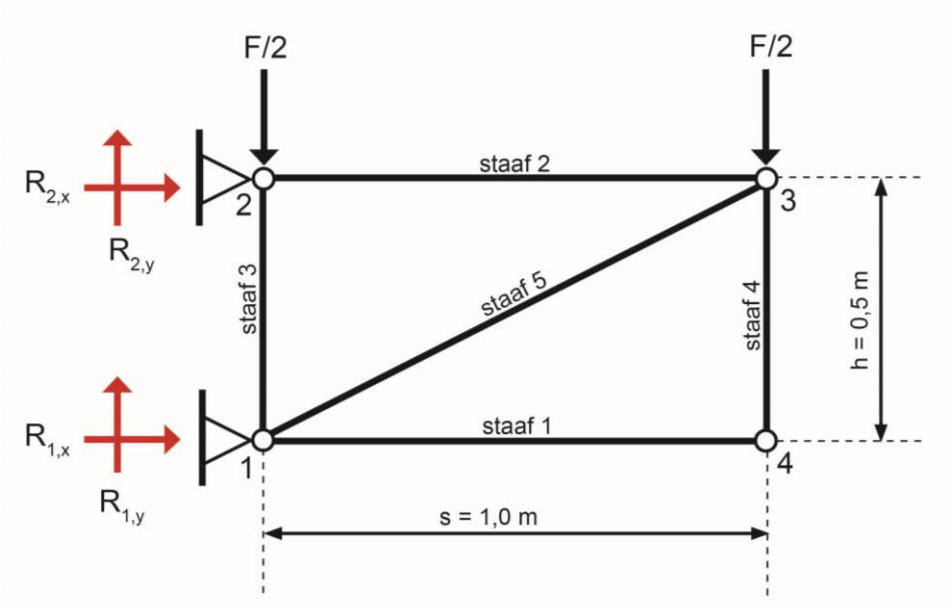
Input:

- Overspanning (span) — s in meters
- Aantal velden — nfields
- Hoogte van de ligger (height) — h in meters
- Type diagonaal — ds (stijgend) of dd (dalend)
- Totale belasting — Ftotal in kN (naar beneden is negatief)
- Opleggingen; knoop en type — supp_node; supp_type(inklemming, oplegging, rol)
- Afmetingen van de staven per groep — d en t in meters

Materiaalgegevens

- Elasticiteitsmodulus — E

Voor de eerste test, waarbij onderzocht wordt hoe de code vanuit Python weggeschreven dient te worden naar een .TRS-file, wordt uitgegaan van het simpele vakwerkje afgebeeld in Figuur 31. De waarden van de parameters die bij die vakwerkje horen, zijn te vinden in Tabel 14. Om te controleren of de koppeling tussen Python en 3D Truss goed gaat, wordt de output van 3D Truss vergeleken met de output van MatrixFrame. Voor het model dat in MatrixFrame is gezet, zijn, als vanzelfsprekend, dezelfde waarden van de parameters gebruikt als in het Pythonscript.



Figuur 31. Eerste model: vlakke vakwerkligger bestaand uit een veld

Tabel 14. Inputgegevens voor het eerste model

Parameters	Waarde
Overspanning (span) s	1.0 m
Nfields	1
Hoogte van de vakwerkligger h	0.5 m
Type diagonaal	ds (stijgend)
Totale rustende belasting Ftotal	100.0 kN
Opleggingen knoopnummer supp_node	1 en 2
Opleggingen type supp_type	Inklemming; inklemming
Afmetingen onderrandstaven	d_ors = 0.1 m; t_ors = 0.01 m
Afmetingen bovenrandstaven	d_brs = 0.1 m; t_brs = 0.01 m
Afmetingen staanders	d_st = 0.06 m; t_st = 0.006 m
Afmetingen diagonalen	d_dia = 0.06 m; t_dia = 0.006 m
Elasticiteitsmodulus staal	E = 210.000.000 kN/m^2

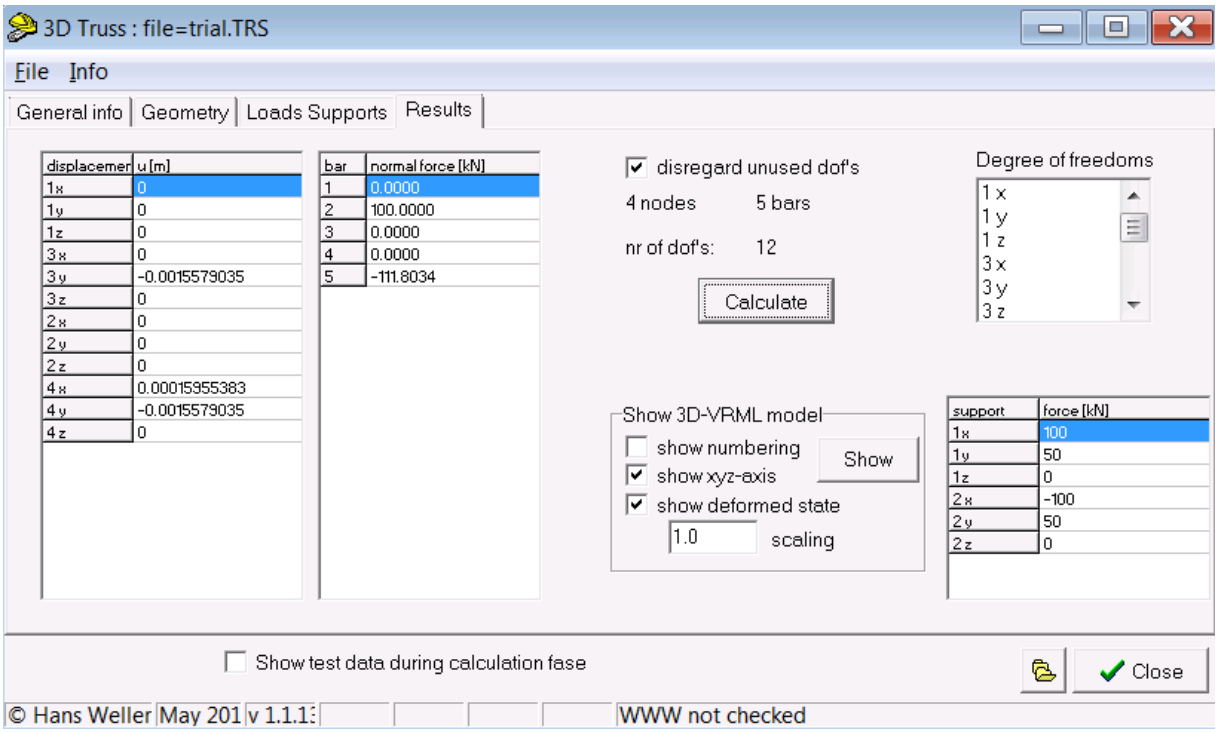
De output van het programma 3D Truss, die hoort bij de situatie als geschetst in Figuur 31, komt overeen met de resultaten van het MatrixFrame model. Er kan zodoende geconcludeerd worden dat de wijze waarop de inputfile voor 3D Truss gegenereerd wordt kloppend is.

De output van zowel 3D Truss als MatrixFrame is hieronder weergegeven.

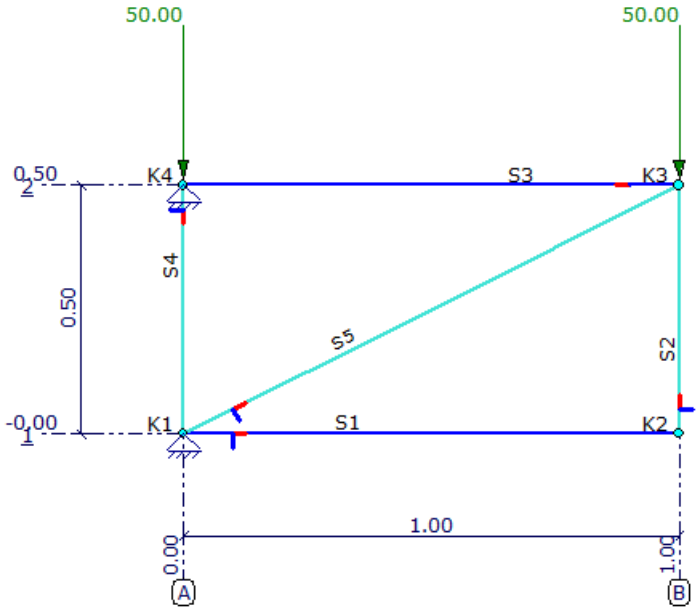
Pythonscript (inputform geeft weer hoe de inputfile eruit ziet)

```
The coordinates of the nodes are:
   node number  x    y  z     input-form
0            1  0  0.0  0  1=0.0@0.0@0.0
1            2  0  0.5  0  2=0.0@0.5@0.0
2            3  1  0.0  0  3=1.0@0.0@0.0
3            4  1  0.5  0  4=1.0@0.5@0.0
The elements are:
   element number  start point  end point  type input-form
0               1            1          3     1    1=1@3@1
1               2            2          4     2    2=2@4@2
2               3            1          2     3    3=1@2@3
3               4            3          4     3    4=3@4@3
4               5            1          4     4    5=1@4@4
The properties of elements are:
   type number          E      Area                          input-form
0            1  210000000  0.002985  1=210000000.0@0.00298451302091
1            2  210000000  0.002985  2=210000000.0@0.00298451302091
2            3  210000000  0.001074  3=210000000.0@0.00107442468753
3            4  210000000  0.001074  4=210000000.0@0.00107442468753
The loads on the truss are:
   load number  node number direction  magnitude   input-form
0            1            2         y        -50  1=2@y@-50.0
1            2            4         y        -50  2=4@y@-50.0
The supports of the truss are:
   support number  node number prevented direction input-form
0               1            1                   x      1=1@x
1               2            1                   y      2=1@y
2               3            1                   z      3=1@z
3               4            2                   x      4=2@x
4               5            2                   y      5=2@y
5               6            2                   z      6=2@z
```

Output 3D Truss



MatrixFrame model



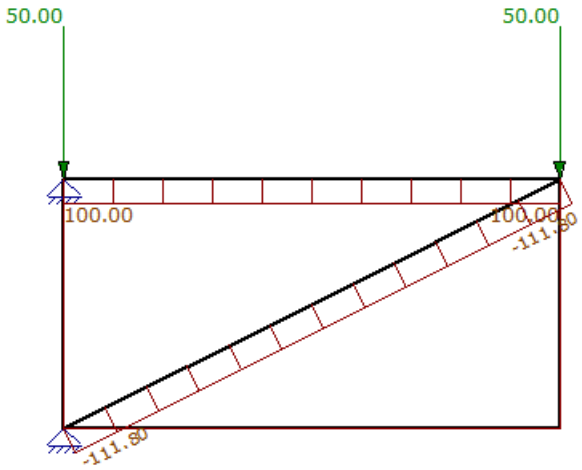
Profielen

	Label	Oppervl.	E-mod.	Mat. type	Mat. code	Fabr. type
▶	P1	2.9850e-0	2.1000e+	Handma	Niet gedef.	
	P2	1.0740e-0	2.1000e+	Handma	Niet gedef.	

Staven

	Staaf	Profiel B
▶	S1	P1
	S2	P2
	S3	P1
	S4	P2
	S5	P2

Resultaten MatrixFrame



Oplegreacties

B.G.	Opleggin	Knoop	X	Z
B.G.1	O1	K4	-100.000	-50.000
B.G.1	O2	K1	100.000	-50.000

Staafkrachten

Staaf	B.G.	T/D	Nxmax
S1	B.G.1	-	0.00
S2		-	0.00
S3		T	100.00
S4		-	0.00
S5		D	-111.8

Knoopverplaatsingen

Knoop	B.G.	X	Z
K1	B.G.1	-0.0000	0.0000
K2		0.0000	0.0016
K3		0.0002	0.0016
K4		0.0000	0.0000

D. Validatie van het computermodel met behulp van handberekeningen

Deze bijlage bevat de handberekeningen die gebruikt zijn om de werking van het computermodel te controleren. Per component zijn de resultaten van de handberekeningen vergeleken met de output van het computermodel. Deze kwamen allen overeen. Dit betekent dat geconcludeerd kan worden dat de code van het model klopt en dat ook bij andere inputgegevens de juiste resultaten verwacht mogen worden.

De handberekeningen die in deze bijlage te zien zijn, horen bij het ontwerp van een vakwerkligger met de Howe truss configuratie, met dezelfde staafafmetingen voor alle staven en belast door de belastingcombinatie waarbij de opwaartse windbelasting maatgevend is gekozen. De code die hoort bij de overige configuraties en fundamentele combinaties is op dezelfde wijze met behulp van handberekeningen gecontroleerd op juistheid.

Naast de handberekeningen is ook de inputfile voor 3D Truss in .TRS-format opgenomen in deze bijlage.

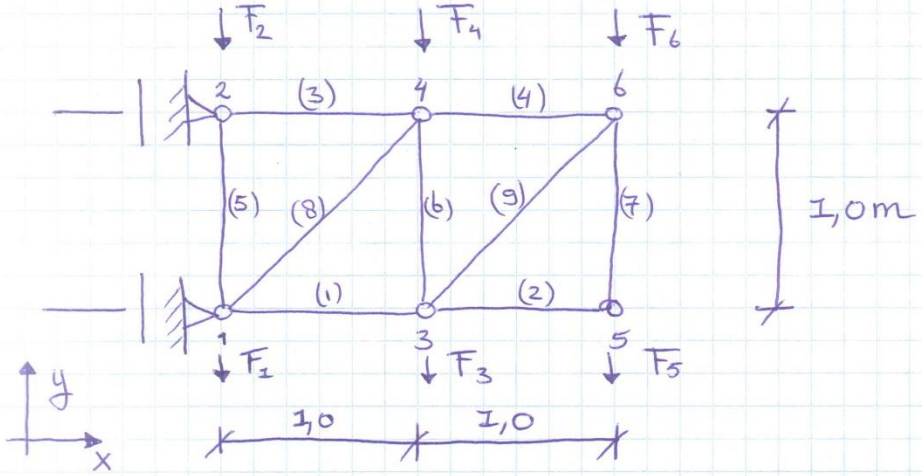
Validatie van 2D-vakwerk computermodel

Input-ontwerpgegevens:

s = 2,0 m
h = 1,0 m
nblocks = 2
config = 'howe'
roofangle = 0°

Belastingcombinatie: FC2b
(windbelasting opwaarts is maatgevend)

(overige ontwerpgegevens zijn gelijk aan de waarden omschreven in hoofdstuk 3)



Ontwerpmodule

* Coördinaten van de knooppunten:

nummer	x	y	z
1	0	0	0
2	0	1	0
3	1	0	0
4	1	1	0
5	2	0	0
6	2	1	0

* Locatie en type van de staven

nummer	startpunt	eindpunt	type
1	1	3	1
2	3	5	1
3	2	4	2
4	4	6	2
5	1	2	3
6	3	4	3
7	5	6	3
8	1	4	4
9	3	6	4

De typen zijn:
1. onderrandstaaf (ors)
2. bovenrandstaaf (brs)
3. staander (st)
4. diagonaal (dia)

* Staafgegevens

type	diameter	wanddikte	doorsnede oppervlak*
1=ors	0,04 m	0,004 m	$2,3876 \cdot 10^{-4}$ m²
2=brs	"	"	"
3=st	"	"	"
4=dia	"	"	"

LET OP! Hier zijn de constraints voor de staafafmetingen nog niet toegepast!

* Doorsnede opp. is als volgt berekend:

$$A = \frac{1}{4} \cdot \pi \cdot \left(d^2 - (d-t)^2\right)$$

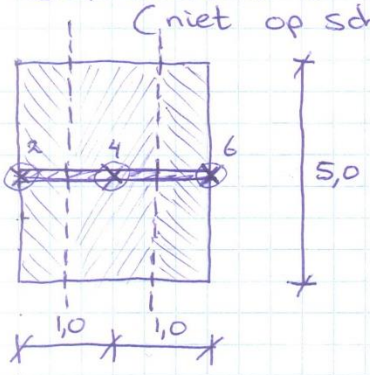
Module met belastingen & FC's

* Permanente belastingen

- Dakbedekking → 0,15 kN/m²

Dakvlak dat afdraagt aan de ligger:

(niet op schaal)



$F_2 = F_6 = 0,5 \cdot 5,0 \cdot 0,15 = 0,375$ kN

$F_4 = 1,0 \cdot 5,0 \cdot 0,15 = 0,75$ kN

Eigen gewicht → $\rho_{staal} = 80\ kN/m^3$

Gewicht van een staaf wordt afgedragen aan de 2 knooppunten die door de staaf met elkaar verbonden worden.

$F_{E.G.} = A \cdot \ell \cdot \rho_{staal}$

Staaftype	A	ℓ	$F_{E.G.}$ [kN]
1, 2 en 3	$2{,}3876 \cdot 10^{-4}$	1,0	0,0191
4	"	1,414	0,0270

Knoopnummer	belasting t.g.v. eigen gewicht
1 en 6	$\frac{1}{2} \cdot F_{1,2} + \frac{1}{2} \cdot F_3 + \frac{1}{2} \cdot F_4 = 0{,}0326$ kN
2 en 5	$\frac{1}{2} \cdot F_{1,2} + \frac{1}{2} F_3 = 0{,}0191$ kN
3 en 4	$1 \cdot F_{1,2} + \frac{1}{2} \cdot F_3 + \frac{1}{2} \cdot F_4 = 0{,}0422$ kN

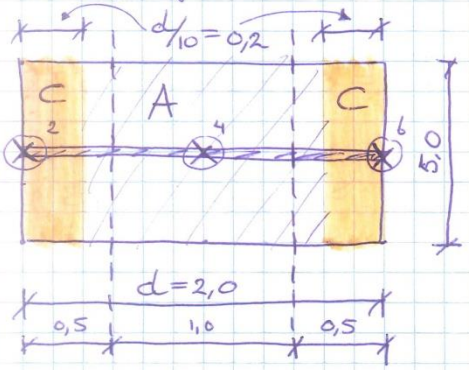
* Veranderlijke belastingen

Windbelasting

Windgebied 3; urban; $z_e = 13{,}5 + 1{,}0 = 14{,}5$ m.

→ $q_p(z_e = 14{,}5) = 0{,}66\ kN/m^2$

Zones op het dakvlak: (niet op schaal)



Opwaartse windbelasting:

$c_{s-A} = -1{,}5$

$c_{s-C} = -2{,}2$

$F_{wind} = C_s C_d \cdot C_s \cdot q_p \cdot A_{ref}$

$C_s C_d = 1{,}0$

$F_{wind,A} = 1{,}0 \cdot -1{,}5 \cdot 0{,}66 = -0{,}99 \ kN/m^2$

$F_{wind,C} = 1{,}0 \cdot -2{,}2 \cdot 0{,}66 = -1{,}452 \ kN/m^2$

$F_2 = F_6 = 0{,}2 \cdot 5{,}0 \cdot F_{wind,C} + (0{,}5 - 0{,}2) \cdot 5{,}0 \cdot F_{wind,A}$

$= 0{,}2 \cdot 5{,}0 \cdot -1{,}452 + 0{,}3 \cdot 5{,}0 \cdot -0{,}99 = -2{,}937 \ kN(\uparrow)$

$F_4 = 1{,}0 \cdot 5{,}0 \cdot F_{wind,A} = 5 \cdot -0{,}99 = -4{,}95 \ kN(\uparrow)$

Fundamentele belastingcombinatie

Totale belasting = $0{,}9 \cdot G + 1{,}65 \cdot Q_{wind,qp}$

G = permanente belasting:

Dak: [0 0,375 0 0,75 0 0,375]

E.G.: [0,0326 0,991 0,0422 0,0422 0,0191 0,0326] +

G: [0,0326 0,3941 0,0422 0,7922 0,0191 0,4076]

$Q_{wind,qp}$ = variabele windbelasting; qp waarts

$Q_{wind,qp}$: [0 -2,937 0 -4,95 0 -2,937]

$F_{totaal} = 0,9 \cdot G + 1,65 \cdot Q_{wind,op}$:

F_{totaal}: [0,02934 ↓; -4,4914 ↑; 0,03798 ↓; -7,4545 ↑; 0,01719 ↓; -4,4792 ↑]

Richting van de belasting: y (+ of -)

De gegevens die nu berekend zijn worden ingevoerd in de rekenkern 3D Truss. Dit programma geeft als output de staafkrachten, de knoopverplaatsingen en de reactiekrachten.
De staafkrachten zijn:

N_Ed = [① -4,463; ② 0; ③ 16,345; ④ 4,463; ⑤ 0; ⑥ 4,4262; ⑦ -0,0168; ⑧ -16,8032; ⑨ -6,312]

Toetsingsmodule

* Normaalkrachtcapaciteit

$N\_Rd = A \cdot f_y$; aangezien de vier staaftypen allemaal dezelfde afmetingen hebben, is ook N_Rd voor alle staven hetzelfde.

$N\_Rd = 2,3876 \cdot 10^{-4} \cdot 10^{6} \cdot 235 \cdot 10^{-3} \approx 56,1086$ kN
$\approx 56,1$ kN

* Knikcapaciteit

Buisprofiel, warmvervaardigd, S235:

Knikkromme a; imperfectiefactor 0,21

Grensslankheid: $\lambda_e = \pi \cdot \sqrt{\frac{E}{S_y}} = \pi \cdot \sqrt{\frac{210.000}{235}} \approx 93,91$

$$I = \frac{1}{64} \cdot \pi \cdot (d^4 - (d-t)^4)$$

$$= \frac{1}{64} \cdot \pi \cdot (0,04^4 - (0,04-0,004)^4) \approx 4,32157 \cdot 10^{-8} \; m^4$$

$$A \approx 2,3876 \cdot 10^{-4} \; m^2$$

$$i = \sqrt{I/A} \approx 0,0135 \; m$$

Staaftype	Knik lengte	$\lambda = \frac{\ell}{i}$	$\lambda_{rel} = \frac{\lambda}{\lambda_e}$
1,2 en 3	1,0	74,329	0,791
4	1,414	105,10	1,119

$$\phi = 0,5 \left[1 + \alpha (\lambda_{rel} - 0,2) + \lambda_{rel}^2 \right]$$

$$\phi_{1,2,3} = 0,5 \left[1 + 0,21 (0,791 - 0,2) + 0,791^2 \right] \approx 0,8749$$

$$\phi_4 = 0,5 \left[1 + 0,21 (1,119 - 0,2) + 1,119^2 \right] \approx 1,2226$$

$$\chi = \frac{1}{\phi + \sqrt{\phi^2 - \lambda_{rel}^2}}$$

$$\chi_{1,2,3} = \frac{1}{9,8749 + \sqrt{0,8749^2 - 0,791^2}} \approx 0,8008$$

$$\chi_4 = \frac{1}{1,2226 + \sqrt{1,2226^2 - 1,119^2}} \approx 0,5831$$

$$N_{b,Rd} = \chi \cdot N_{Rd}$$

$$N_{b,Rd_{1,2,3}} \cong 0,8008 \cdot 56,1 \approx 44,9 \text{ KN}$$

$$N_{b,Rd_4} \cong 0,5831 \cdot 56,1 \approx 32,7 \text{ KN}$$

* Unity checks voor sterkte en stabiliteit:

$$UC\_N = \frac{N_{Ed}}{N_{Rd}} \overset{EIS}{\leq} 1,0$$

$$UC\_Nb^* = \frac{N_{Ed}}{N_{b,Rd}} \overset{EIS}{\leq} 1,0$$

* Alleen voor op druk belaste staven.

NB. Bovenstaande handberekening hoort bij de validatie van een oudere versie van het computermodel. De formules die bij deze oudere versie zijn gebruikt, zijn later aangepast en verbeterd. Doordat bij de handberekening echter dezelfde formules zijn gebruikt, blijft de validatie geldig.

Na iedere uitbreiding/aanpassing van het model zijn de nieuwe/aangepaste onderdelen gevalideerd. Deze validatie is iedere keer volgens hetzelfde principe uitgevoerd als beschreven in deze bijlage.

Inputfile voor de rekenkern 3D Truss. Format .TRS.

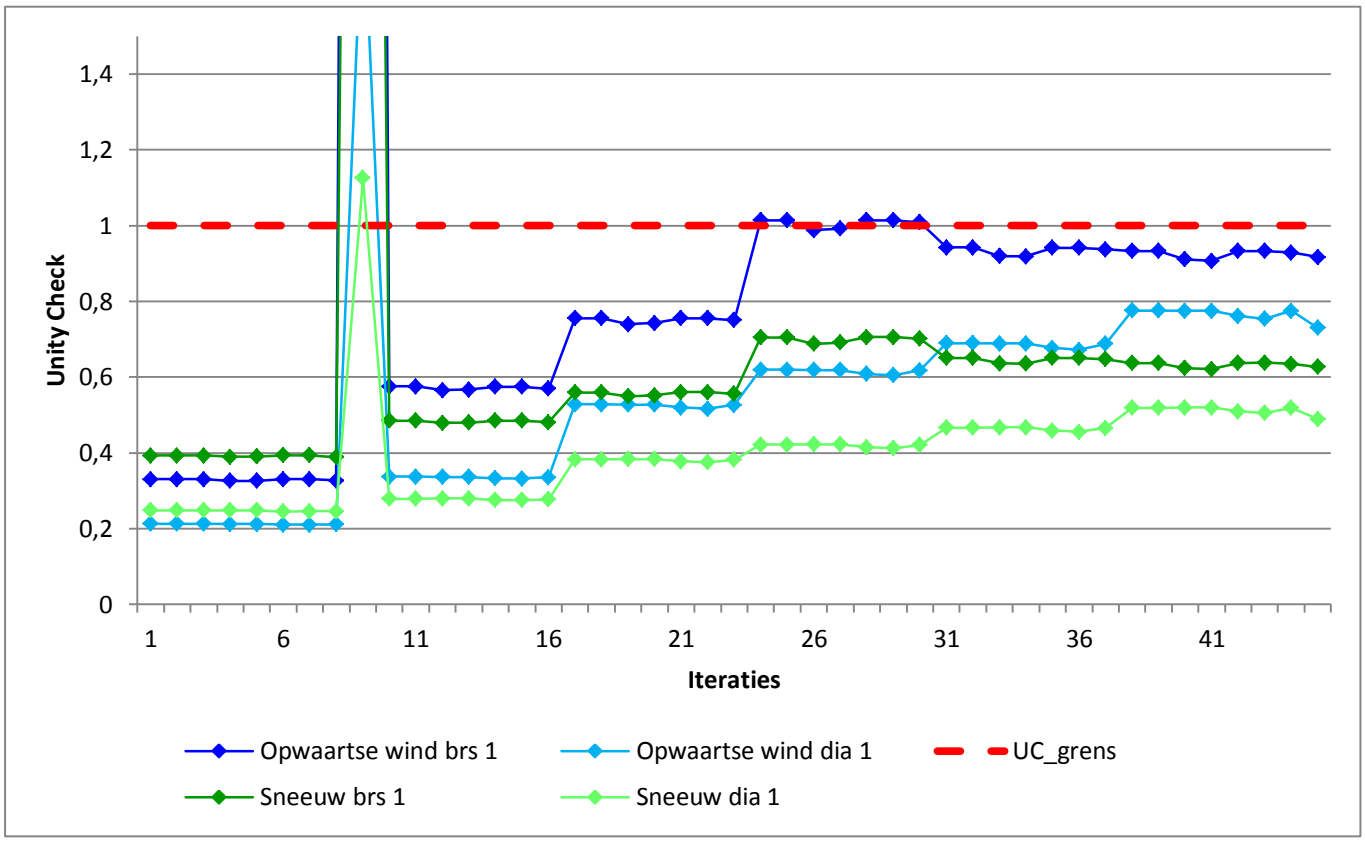
```
[settings]
maxlength=100
[properties]
E=210000000
A=1
[coordinates]
1=0.000@0.000@0.000
2=0.000@1.000@0.000
3=1.000@0.000@0.000
4=1.000@1.000@0.000
5=2.000@0.000@0.000
6=2.000@1.000@0.000
[elements]
1=1@3@1
2=3@5@1
3=2@4@2
4=4@6@2
5=1@2@3
6=3@4@3
7=5@6@3
8=1@4@4
9=3@6@4
[barproperties]
1=210000000@0.000238761041673
2=210000000@0.000238761041673
3=210000000@0.000238761041673
4=210000000@0.000238761041673
[loads]
1=1@x@0
2=1@y@0.0293465227192
3=1@z@0
4=2@x@0
5=2@y@-4.491359205
6=2@z@0
7=3@x@0
8=3@y@0.0379419202195
9=3@z@0
10=4@x@0
11=4@y@-7.45455807978
12=4@z@0
13=5@x@0
14=5@y@0.0171907950004
15=5@z@0
16=6@x@0
17=6@y@-4.47920347728
18=6@z@0
```

```
[supports]
1=1@x
2=1@y
3=1@z
4=2@x
5=2@y
6=2@z
[displacements]
1=1@x@0
2=1@y@0
3=1@z@0
4=3@x@-8.89914E-5
5=3@y@-0.0010842191
6=3@z@0
7=5@x@-8.89914E-5
8=5@y@-0.0018394662
9=5@z@0
10=2@x@0
11=2@y@0
12=2@z@0
13=4@x@0.00032590149
14=4@y@-0.0009959844
15=4@z@0
16=6@x@0.00041489289
17=6@y@-0.0018398091
18=6@z@0
[elementforces]
1=-4.4620
2=0.0000
3=16.3406
4=4.4620
5=0.0000
6=4.4241
7=-0.0172
8=-16.7989
9=-6.3102
[reactions]
1=1@x@16.340642
2=1@y@11.849282
3=1@z@0
4=2@x@-16.340642
5=2@y@4.4913592
6=2@z@0
```

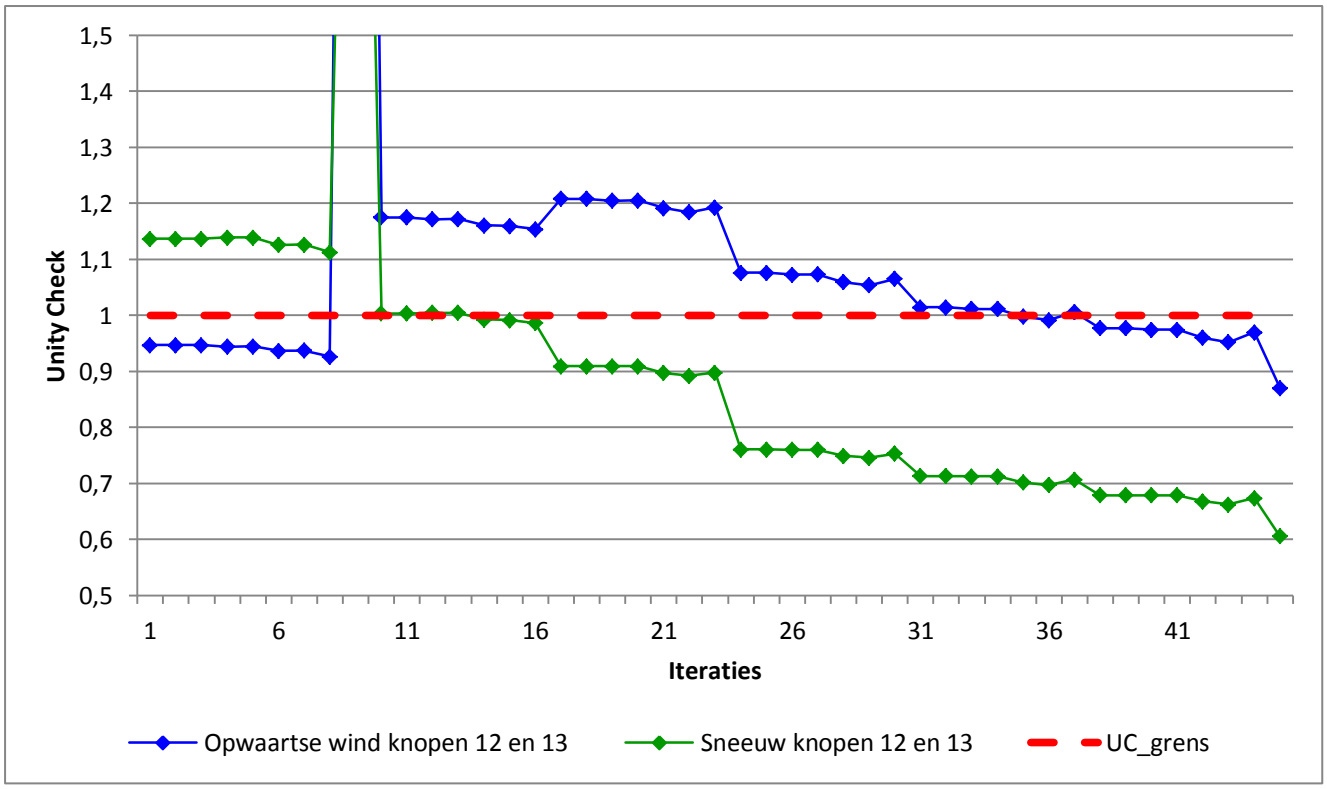
E. Resultaten: Constraints voor het optimale ontwerp

In paragraaf 6.4 is gecontroleerd of het optimale ontwerp, dat gevonden door het computermodel is bepaald, voldoet aan de constraints. Daarbij is het verloop van de drie constraints die maatgevend bleken te zijn voor de dimensionering van het vakwerk weergegeven en toegelicht. Het verloop van de overige constraints is opgenomen in deze bijlage. Het gaat hierbij om de constraints met betrekking tot de normaalkracht, de verticale verplaatsing, de diameterverhouding en de hoek tussen de rand- en wandstaven.

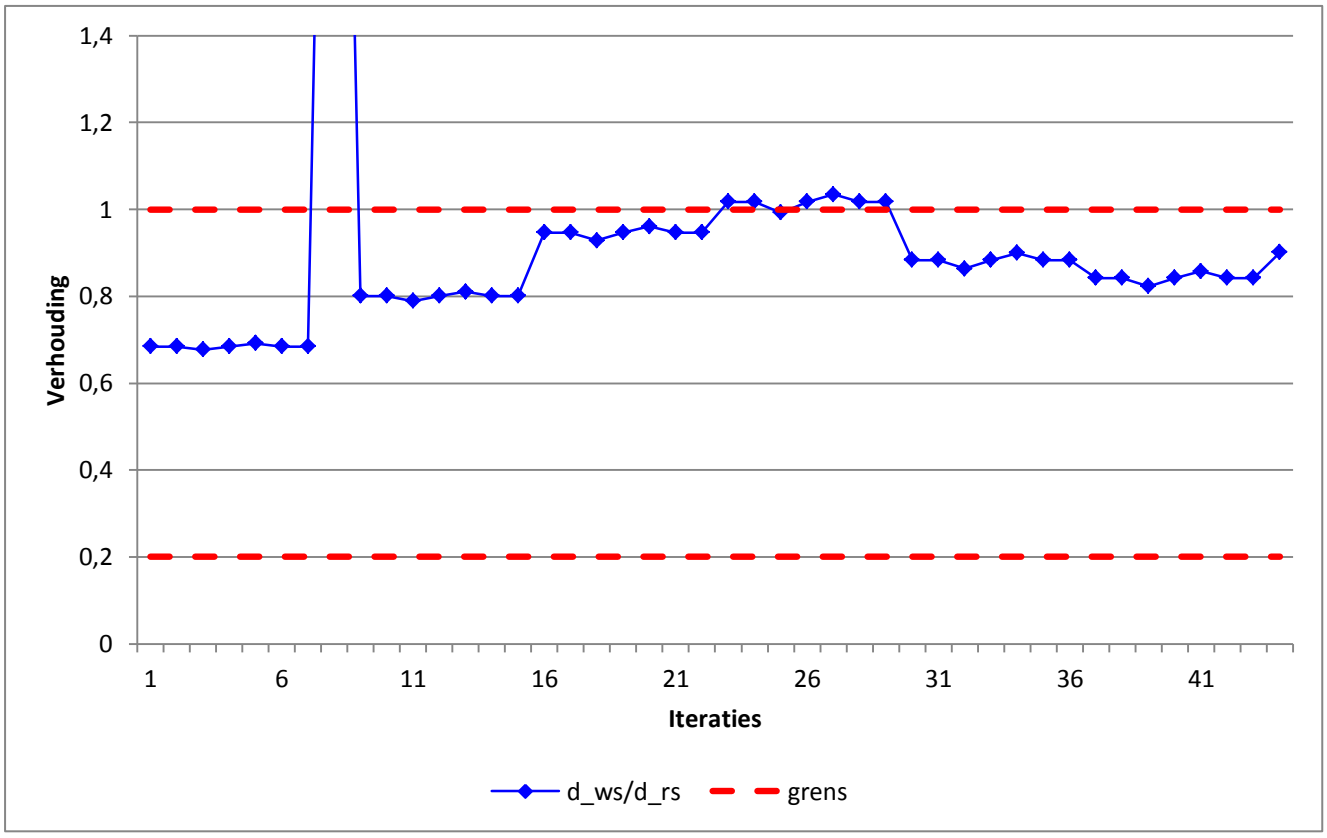
Grafiek 12. Unity Check normaalkracht van de zwaarst belaste randstaaf en diagonaal per FC (10 velden, Warren truss, systeemverbindingen)



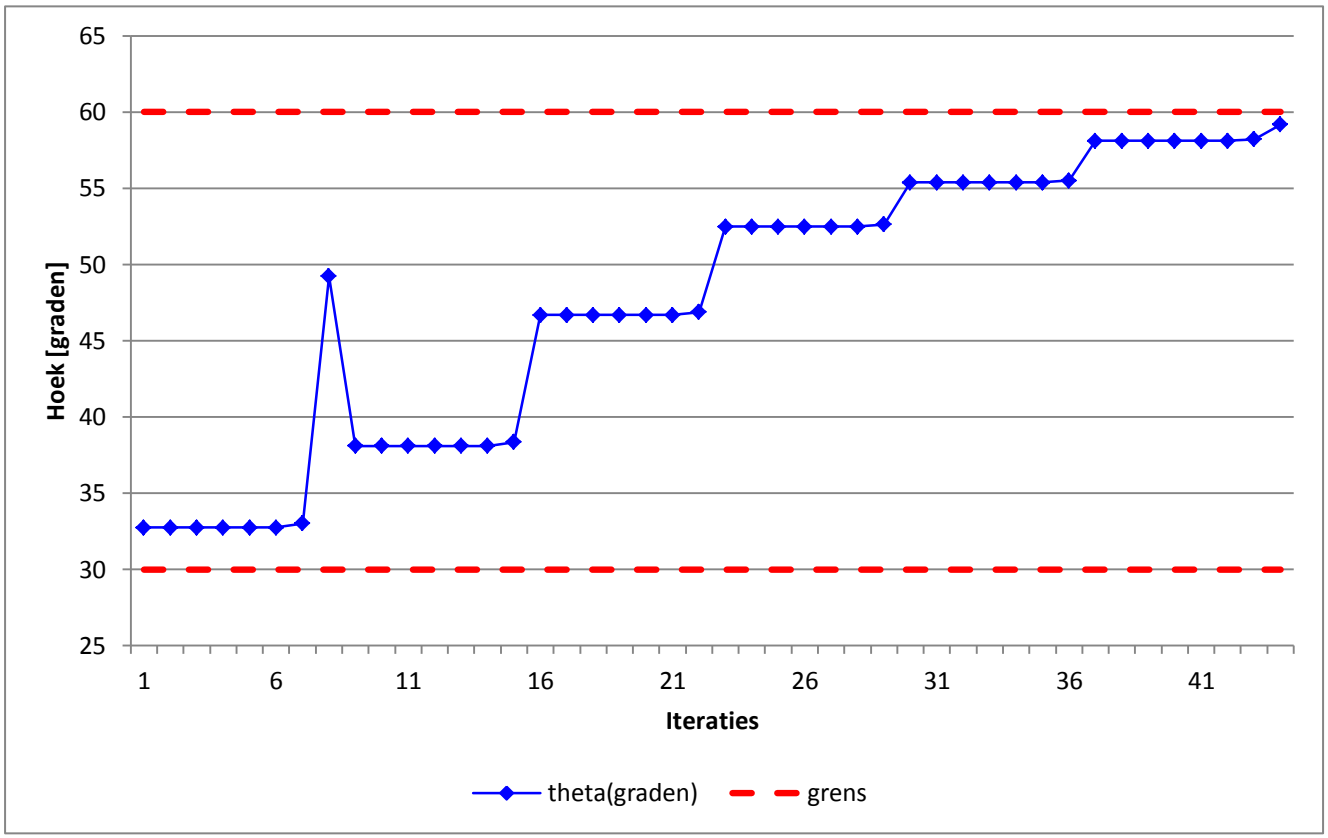
Grafiek 13. Unity Check verticale verplaatsing van het uiteinde van de vakwerkligger per FC (10 velden, Warren truss, systeemverbindingen)



Grafiek 14. Diameterverhouding rand- & wandstaven (10 velden, Warren truss, systeemverbindingen)



Grafiek 15. Hoek tussen de rand- en wandstaven (10 velden, Warren truss, systeemverbindingen)



F. Code van het Computermodel

In deze bijlage is de code van het computermodel opgenomen. Zoals beschreven is in het rapport, is een modulaire opbouw gehanteerd voor het model. De modules waaruit het model is opgebouwd, zijn opgeslagen als losse files. Dit maakt het mogelijk om verschillende onderdelen gemakkelijk uit te breiden, aan te passen of te vervangen.

Deze bijlage is onderverdeeld in 8 deelparagrafen. In iedere deelparagraaf wordt de code van een losstaand gedeelte van het computermodel behandeld.

1. Code: Database & Constantsfile

Deze deelparagraaf bevat de code van de database en de constantsfile.

Database: Circle class

```
import numpy as np

class Circle():
  def __init__(self, name, d, t, vector):
    with open('constants.txt') as f:
      for line in f:
        lines = line.split()
        if lines[0] == 'rho_steel':
          rho_steel = float(lines[2])
        elif lines[0] == 's':
          s = float(lines[2])
        elif lines[0] == 'nfields':
          nfields = int(lines[2])
        elif lines[0] == 'config':
          config = (lines[2])
    h = vector[8]

    self.name = name
    self.d = d
    self.t = t
    self.rho_steel = rho_steel
    self.s = s
    self.nfields = nfields
    self.h = h
    self.config = config

  def display(self):
    if self.name == 'rs1':
      return 'Deze staaf is een randstaaf van liggerdeel 1'
    if self.name == 'rs2':
      return 'Deze staaf is een randstaaf van liggerdeel 2'
    if self.name == 'rs_mid':
      return 'Deze staaf is een randstaaf in het middelste deel van ligger met Warren config.'
    if self.name == 'ws1_st':
      return 'Deze staaf is een staander van liggerdeel 1'
    if self.name == 'ws2_st':
      return 'Deze staaf is een staander van liggerdeel 2'
    if self.name == 'ws1_dia':
      return 'Deze staaf is een diagonaal van liggerdeel 1'
```

```
    if self.name == 'ws2_dia':
        return 'Deze staaf is een diagonaal van liggerdeel 2'

def area(self):
    return 0.25 * np.pi * (self.d ** 2 - (self.d - 2 * self.t) ** 2)

def periphery(self):
    return np.pi * self.d

def length(self):
    if self.name == 'rs1' or self.name == 'rs2':
        return self.s / self.nfields
    elif self.name == 'rs_mid':
        return 2*(self.s / self.nfields)
    elif self.name == 'ws1_st' or self.name == 'ws2_st':
        return self.h
    elif self.name == 'ws1_dia' or self.name == 'ws2_dia':
        return np.sqrt((self.s / self.nfields) ** 2 + self.h ** 2)
    else:
        print 'Name of element is not determined properly for length calculation.'

def volume(self):
    if self.name == 'rs1' or self.name == 'rs2':
        return (self.s / self.nfields) * (0.25 * np.pi * (self.d ** 2 - (self.d - 2 * self.t) ** 2))
    elif self.name == 'rs_mid':
        return 2 * (self.s / self.nfields) * (
                    0.25 * np.pi * (self.d ** 2 - (self.d - 2 * self.t) ** 2))
    elif self.name == 'ws1_st' or self.name == 'ws2_st':
        return self.h * (0.25 * np.pi * (self.d ** 2 - (self.d - 2 * self.t) ** 2))
    elif self.name == 'ws1_dia' or self.name == 'ws2_dia':
        return (np.sqrt((self.s / self.nfields) ** 2 + self.h ** 2)) * (
            0.25 * np.pi * (self.d ** 2 - (self.d - 2 * self.t) ** 2))
    else:
        print 'Name of element is not determined properly for volume calculation.'

def F_self(self):
    if self.name == 'rs1' or self.name == 'rs2':
        return (self.s / self.nfields) * (0.25 * np.pi * (self.d ** 2 - (self.d - 2 * self.t) ** 2)) \
                * self.rho_steel / 100
    elif self.name == 'rs_mid':
        return 2 * (self.s / self.nfields) * (0.25 * np.pi * (self.d ** 2 - (self.d - 2 * self.t) ** 2)) \
                * self.rho_steel / 100
    elif self.name == 'ws1_st' or self.name == 'ws2_st':
        F = self.h * (0.25 * np.pi * (
            self.d ** 2 - (self.d - 2 * self.t) ** 2)) * self.rho_steel / 100
        return F
    elif self.name == 'ws1_dia' or self.name == 'ws2_dia':
        F = (np.sqrt((self.s / self.nfields) ** 2 + self.h ** 2)) * (0.25 * np.pi * (
            self.d ** 2 - (self.d - 2 * self.t) ** 2)) * self.rho_steel / 100
        return F
    else:
```

```
            print 'Name of element is not determined properly for selfweight calculation.'

    def inertia(self):
        return (1 / 64.) * np.pi * (self.d ** 4 - (self.d - 2 * self.t) ** 4)
```

Constantsfile

```
def write_constants_file(n, config, jointtype):
    filename = 'constants.txt'
    with open(filename, "w") as f:
        f.write('E = 210000000' + '\n')
        f.write('fy = 355' + '\n')
        f.write('Fm2_roof = 0.15' + '\n')
        f.write('rho_steel = 7850' + '\n')
        f.write('cs_cd = 1.0' + '\n')
        f.write('s = 21.' + '\n')
        f.write('pform = warmvervaardigd' + '\n')
        f.write('a = 8.0' + '\n')
        f.write('h_vrij = 13.5' + '\n')
        f.write('windregion = 3' + '\n')
        f.write('regiontype = urban' + '\n')
        f.write('nfields = ' + str(n) + '\n')
        f.write('config = ' + config + '\n')
        f.write('mu = 0.8' + '\n')
        f.write('ce = 1.0' + '\n')
        f.write('ct = 1.0' + '\n')
        f.write('sk = 0.7' + '\n')
        f.write('correction = 1.56' + '\n')
        f.write('roofangle = 0' + '\n')
        f.write('supp_node = [1,2]' + '\n')
        f.write("supp_type = ['inklemming','inklemming']" + '\n')
        f.write('price_weldvolume = 0.15' + '\n')
        f.write('price_prefabjoint2 = 50.' + '\n')
        f.write('price_prefabjoint3 = 100.' + '\n')
        f.write('price_prefabjoint4 = 150.' + '\n')
        f.write('jointtype = ' + jointtype + '\n')
    pass
```

2. Code: Ontwerpmodule

In deze deelparagraaf is de code opgenomen van de functies, die horen bij de ontwerpmodule. Deze functies verwerken de volgende onderdelen in het computermodel: de coördinaten van de knooppunten, de staven, de staaftypen en de steunpunten.

Coordinates

```
import numpy as np

def coordinates(vector):
    with open('constants.txt') as f:
        for line in f:
            lines = line.split()
            if lines[0] == 'config':
                config = lines[2]
            elif lines[0] == 's':
                s = float(lines[2])
            elif lines[0] == 'nfields':
                nfields = int(lines[2])
    h = vector[8]

    l = s / nfields
    xlist = []
    zlist = []
    coorlist = []
    if config == 'howe' or config == 'pratt':
        for i in range(nfields + 1):
            x = i * l
            xlist.append(x)
            xlist.append(x)
            z_odd = 0.
            zlist.append(z_odd)
            z_even = h
            zlist.append(z_even)
    elif config == 'warren':
        xlist.append(0)
        zlist.append(0)
        if (nfields % 2. == 0):  # if true: number of fields is even
            xlist.append(0)
            zlist.append(h)
            for i in range(nfields):
                x = i * l + l
                xlist.append(x)
                if (i + 1) % 2. == 0:
                    z = 0
                else:
                    z = h
                zlist.append(z)
        else:
            for i in range(nfields + 1):
                x = i * l
                xlist.append(x)
```

```
            if (i % 2. == 0):
                z = h
            else:
                z = 0
            zlist.append(z)
        xfinal = nfields * l
        xlist.append(xfinal)
        zfinal = h
        zlist.append(zfinal)
    ylist = np.zeros(len(zlist))

    for i in range(len(xlist)):
        c = str(i + 1) + '=' + str(xlist[i]) + '@' + str(ylist[i]) + '@' + str(zlist[i])
        coorlist.append(c)
    return coorlist, xlist, ylist, zlist
```

Elements

```
def elements():
    with open('constants.txt') as f:
        for line in f:
            lines = line.split()
            if lines[0] == 'nfields':
                nfields = int(lines[2])
            elif lines[0] == 'config':
                config = (lines[2])

    start = []
    end = []
    typ = []

    if config == 'howe' or config == 'pratt':
        # onderrandstaven toevoegen
        for i in range(nfields):
            ors_st = 1 + i * 2
            start.append(ors_st)
            ors_end = ors_st + 2
            end.append(ors_end)
            typ.append(1)
        # bovenrandstaven toevoegen
        for i in range(nfields):
            brs_st = 2 + i * 2
            start.append(brs_st)
            brs_end = brs_st + 2
            end.append(brs_end)
            typ.append(1)
        # staanders toevoegen
        for i in range(nfields + 1):
            st_st = 1 + i * 2
            start.append(st_st)
            st_end = st_st + 1
```

```
            end.append(st_end)
            typ.append(2)
        # diagonalen toevoegen
        if config == 'howe':
            for i in range(nfields):
                dia_st = 1 + i * 2
                start.append(dia_st)
                dia_end = dia_st + 3
                end.append(dia_end)
                typ.append(3)
        elif config == 'pratt':
            for i in range(nfields):
                dia_st = 2 + i * 2
                start.append(dia_st)
                dia_end = dia_st + 1
                end.append(dia_end)
                typ.append(3)
    elif config == 'warren':
        if (nfields % 2. == 0):  # if true: number of fields is even
            # onderrandstaven toevoegen
            ors_st = 1
            start.append(ors_st)
            ors_end = ors_st + 3
            end.append(ors_end)
            typ.append(4)
            for i in range((nfields - 2) / 2):
                ors_st = 4 + i * 2
                start.append(ors_st)
                ors_end = ors_st + 2
                end.append(ors_end)
                typ.append(4)
            # bovenrandstaven toevoegen
            brs_st = 2
            start.append(brs_st)
            brs_end = brs_st + 1
            end.append(brs_end)
            typ.append(1)
            for i in range((nfields / 2) -1):
                brs_st = 3 + i * 2
                start.append(brs_st)
                brs_end = brs_st + 2
                end.append(brs_end)
                typ.append(4)
            start.append(nfields+1)
            end.append(nfields+3)
            typ.append(1)
            # staanders toevoegen
            st_st1 = 1
            start.append(st_st1)
            st_end1 = st_st1 + 1
            end.append(st_end1)
```

```
    typ.append(2)
    st_st2 = nfields + 2
    start.append(st_st2)
    st_end2 = st_st2 + 1
    end.append(st_end2)
    typ.append(2)
    # diagonalen toevoegen
    dia_st = 1
    start.append(dia_st)
    dia_end = 3
    end.append(dia_end)
    typ.append(3)
    for i in range(nfields - 1):
        dia_st = 3 + i
        start.append(dia_st)
        dia_end = dia_st + 1
        end.append(dia_end)
        typ.append(3)
else:
    # onderrandstaven toevoegen
    ors_st = 1
    start.append(ors_st)
    ors_end = ors_st + 2
    end.append(ors_end)
    typ.append(1)
    for i in range((nfields - 1) / 2):
        ors_st = 3 + i * 2
        start.append(ors_st)
        ors_end = ors_st + 2
        end.append(ors_end)
        typ.append(4)
    # bovenrandstaven toevoegen
    for i in range(((nfields - 1) / 2)):
        brs_st = 2 + i * 2
        start.append(brs_st)
        brs_end = brs_st + 2
        end.append(brs_end)
        typ.append(4)
    start.append(nfields+1)
    end.append(nfields+3)
    typ.append(1)
    # staanders toevoegen
    st_st1 = 1
    start.append(st_st1)
    st_end1 = st_st1 + 1
    end.append(st_end1)
    typ.append(3)
    st_st2 = nfields + 2
    start.append(st_st2)
    st_end2 = st_st2 + 1
    end.append(st_end2)
```

```
            typ.append(2)
            # diagonalen toevoegen
            for i in range(nfields):
                dia_st = 2 + i
                start.append(dia_st)
                dia_end = dia_st + 1
                end.append(dia_end)
                typ.append(3)
    else:
        print 'Configuration is not defined properly.'

    elemlist = []
    for i in range(len(start)):
        element = str(i + 1) + '=' + str(start[i]) + '@' + str(end[i]) + '@' + str(typ[i])
        elemlist.append(element)
    return elemlist, start, end, typ
```

Bartypes

```
import numpy as np

def bartypes(rs1, rs_mid, rs2, ws1_st, ws2_st, ws1_dia, ws2_dia):
    with open('constants.txt') as f:
        for line in f:
            lines = line.split()
            if lines[0] == 'E':
                E = float(lines[2])

    type_A = []
    type_A.append(rs1.area())
    type_A.append(rs_mid.area())
    type_A.append(ws1_st.area())
    type_A.append(ws1_dia.area())
    type_A.append(rs2.area())
    type_A.append(ws2_st.area())
    type_A.append(ws2_dia.area())

    typelist = []
    for i in range(len(type_A)):
        props = str(i + 1) + '=' + str(E) + '@' + str(type_A[i])
        typelist.append(props)
    return typelist, type_A
```

Supports

```
from optitruss.coordinates import coordinates

def supports(supp_node, supp_type, vector):
    coorlist = coordinates(vector)[0]
    supp_node_list = []
    supp_type_list = []
```

```
for node in supp_node:
    supp_node_list.append(node)
for type in supp_type:
    supp_type_list.append(type)
for i in range(len(coorlist)):
    supp_node_list.append(i+1)
    supp_type_list.append('rol_xz')
supp_list = []
supp_direct = []
for i in range(len(supp_node_list)):
    if supp_type_list[i] == 'inklemming':
        for j in range(2):
            supp_list.append(supp_node_list[i])
        supp_direct.append('x')
        supp_direct.append('z')
    if supp_type_list[i] == 'rol_x':
        supp_list.append(supp_node_list[i])
        supp_direct.append('z')
    if supp_type_list[i] == 'rol_y':
        print 'LET OP! Geen zijwaartse ondersteuning in dit knooppunt!'
        for j in range(2):
            supp_list.append(supp_node_list[i])
        supp_direct.append('x')
        supp_direct.append('z')
    if supp_type_list[i] == 'rol_z':
        supp_list.append(supp_node_list[i])
        supp_direct.append('x')
    if supp_type_list[i] == 'rol_xy':
        print 'LET OP! Geen zijwaartse ondersteuning in dit knooppunt!'
        supp_list.append(supp_node_list[i])
        supp_direct.append('z')
    if supp_type_list[i] == 'rol_xz':
        supp_list.append(supp_node_list[i])
        supp_direct.append('y')
    if supp_type_list[i] == 'rol_yz':
        print 'LET OP! Geen zijwaartse ondersteuning in dit knooppunt!'
        supp_list.append(supp_node_list[i])
        supp_direct.append('x')

supportlist = []
for i in range(len(supp_list)):
    support = str(i + 1) + '=' + str(supp_list[i]) + '@' + str(supp_direct[i])
    supportlist.append(support)
return supportlist, supp_list, supp_direct
```

3. Code: Module ter bepaling van de belastingen en de fundamentele combinaties

Deze deelparagraaf bevat de functies die de grootte van de verschillende belastingen berekenen. Ook zijn de functies die de FCs verwerken in het computermodel in deze deelparagraaf terug te vinden.

Loads

```
import numpy as np

from optitruss.coordinates import coordinates
from optitruss.elements import elements

def permloads(rs1, rs_mid, rs2, ws1_st, ws2_st, ws1_dia, ws2_dia, vector):
    with open('constants.txt') as f:
        for line in f:
            lines = line.split()
            if lines[0] == 'Fm2_roof':
                Fm2_roof = float(lines[2])
            elif lines[0] == 's':
                s = float(lines[2])
            elif lines[0] == 'a':
                a = float(lines[2])
    h = vector[8]

    coorlist = coordinates(vector)[0]
    zlist = coordinates(vector)[3]
    startlist = elements()[1]
    endlist = elements()[2]
    typelist = elements()[3]

# Load per node due to selfweight of the elements
    force_truss = np.zeros(len(coorlist))
    for i in range(len(startlist)):
        if typelist[i] == 1:
            F_bar = rs1.F_self()
        elif typelist[i] == 2:
            F_bar = ws1_st.F_self()
        elif typelist[i] == 3:
            F_bar = ws1_dia.F_self()
        elif typelist[i] == 4:
            F_bar = rs_mid.F_self()
        elif typelist[i] == 5:
            F_bar = rs2.F_self()
        elif typelist[i] == 6:
            F_bar = ws2_st.F_self()
        elif typelist[i] == 7:
            F_bar = ws2_dia.F_self()
        else:
            print 'Element type is not defined properly.'
        start = startlist[i]
        end = endlist[i]
        force_truss[start - 1] += 0.5 * F_bar
```

```
        force_truss[end - 1] += 0.5 * F_bar

# Load per node due to the weight of the roof
    F_roof = s * a * Fm2_roof
    force_roof = np.zeros(len(coorlist))
    numb_node = 0.
    for z in zlist:
        if z == h:
            numb_node += 1
    Fnode_roof = F_roof / (numb_node - 1)
    for i in range(len(zlist)):
        if zlist[i] == h:
            if i == 1 or i == len(zlist) - 1:
                force_roof[i] += Fnode_roof / 2.
            else:
                force_roof[i] += Fnode_roof

# Total permanent load per node
    force_perm = force_truss + force_roof
    return force_perm, force_truss, force_roof


def windload(cf_type, vector):
    with open('constants.txt') as f:
        for line in f:
            lines = line.split()
            if lines[0] == 'cs_cd':
                cs_cd = float(lines[2])
            elif lines[0] == 'config':
                config = lines[2]
            elif lines[0] == 'nfields':
                nfields = int(lines[2])
            elif lines[0] == 'h_vrij':
                h_vrij = float(lines[2])
            elif lines[0] == 'a':
                a = float(lines[2])
            elif lines[0] == 's':
                s = float(lines[2])
            elif lines[0] == 'windregion':
                windregion = int(lines[2])
            elif lines[0] == 'regiontype':
                regiontype = (lines[2])
            elif lines[0] == 'roofangle':
                roofangle = float(lines[2])

    h = vector[8]

# Determine the 'extreme stuwdruk qp':
    z_e = h_vrij + h
    if z_e <= 10. or z_e > 20.:
        print 'Height is out of range'
```

```
        qp = 1.16
    else:
        if windregion == 1:
            if regiontype == 'coastal':
                if 10. < z_e <= 15.:
                    qp = 1.71
                elif 15 < z_e <= 20.:
                    qp = 1.80
                else:
                    print 'qp for 1_coastal is not defined.'
            elif regiontype == 'rural':
                if 10. < z_e <= 15.:
                    qp = 1.16
                elif 15 < z_e <= 20.:
                    qp = 1.27
                else:
                    print 'qp for 1_rural is not defined.'
            elif regiontype == 'urban':
                if 10. < z_e <= 15.:
                    qp = 0.96
                elif 15. < z_e <= 20.:
                    qp = 1.07
                else:
                    print 'qp for 1_urban is not defined.'
            else:
                print 'regiontype for windregion 1 is not defined properly.'
        elif windregion == 2:
            if regiontype == 'coastal':
                if 10. < z_e <= 15.:
                    qp = 1.43
                elif 15 < z_e <= 20.:
                    qp = 1.51
                else:
                    print 'qp for 2_coastal is not defined.'
            elif regiontype == 'rural':
                if 10. < z_e <= 15.:
                    qp = 0.98
                elif 15 < z_e <= 20.:
                    qp = 1.07
                else:
                    print 'qp for 2_rural is not defined.'
            elif regiontype == 'urban':
                if 10. < z_e <= 15.:
                    qp = 0.80
                elif 15. < z_e <= 20.:
                    qp = 0.90
                else:
                    print 'qp for 2_urban is not defined.'
            else:
                print 'regiontype for windregion 2 is not properly.'
        elif windregion == 3:
```

```
            if regiontype == 'rural':
                if 10. < z_e <= 15.:
                    qp = 0.80
                elif 15 < z_e <= 20.:
                    qp = 0.88
                else:
                    print 'qp for 3_rural is not defined.'
            elif regiontype == 'urban':
                if 10. < z_e <= 15.:
                    qp = 0.66
                elif 15. < z_e <= 20.:
                    qp = 0.74
                else:
                    print 'qp for 3_urban is not defined.'
            else:
                print 'regiontype for windregion 3 is not defined properly.'
        else:
            print 'windregion is not defined properly.'

# Determine the 'krachtcoefficient cf':
    if cf_type == 'down':
        if roofangle == 0.:
            cf_A = 0.5
            cf_C = 1.1
        elif roofangle == 5.:
            cf_A = 0.8
            cf_C = 1.3
        elif roofangle == 10.:
            cf_A = 1.2
            cf_C = 1.6
        else:
            print 'roofangle is out of range.'
    elif cf_type == 'up':
        if roofangle == 0.:
            cf_A = -1.5
            cf_C = -2.2
        elif 0. < roofangle <= 5.:
            cf_A = -1.6
            cf_C = -2.5
        elif 5. < roofangle <= 10.:
            cf_A = -2.1
            cf_C = -2.7
        else:
            print 'roofangle is out of range.'
    else:
        print 'cf_type is not defined properly'

    F_wa = cs_cd * cf_A * qp  # in kN/m2
    F_wc = cs_cd * cf_C * qp

# Load per node due to the wind:
```

```
    l = s / nfields
    coorlist = coordinates(vector)[0]
    zlist = coordinates(vector)[3]
    F_wind = np.zeros(len(coorlist))
    if config == 'howe' or config == 'pratt':
        for i in range(len(coorlist)):
            if zlist[i] == h:
                if nfields <= 10:
                    if nfields <= 5:
                        if i == 1 or i == len(coorlist) - 1:
                            F_wind[i] = a * ((s / 10. * F_wc) + (((l / 2.) - (s / 10.)) * F_wa))
                        else:
                            F_wind[i] = a * l * F_wa
                    else:
                        if i == 1 or i == len(coorlist) - 1:
                            F_wind[i] = a * (l / 2.) * F_wc
                        elif i == 3 or i == len(coorlist) - 3:
                            F_wind[i] = a * ((((s / 10.) - (l / 2.)) * F_wc) + ((l - (s / 10.)) * F_wa) + ((l / 2.) * F_wa))
                        elif i > 3 and i < len(coorlist) - 3:
                            F_wind[i] = a * l * F_wa
                        else:
                            print '1. Number of nodes is out of range.'
                elif 10 < nfields <= 20:
                    if nfields <= 15:
                        if i == 1 or i == len(coorlist) - 1:
                            F_wind[i] = a * (l / 2.) * F_wc
                        elif i == 3 or i == len(coorlist) - 3:
                            F_wind[i] = a * ((((l / 2.) * F_wc) + (((s / 10.) - l) * F_wc) + (((1.5 * l) - (s / 10.)) * F_wa))
                        elif i > 3 and i < len(coorlist) - 3:
                            F_wind[i] = a * l * F_wa
                        else:
                            print '2. Number of nodes is out of range.'
                    else:
                        if i == 1 or i == len(coorlist) - 1:
                            F_wind[i] = a * (l / 2.) * F_wc
                        elif i == 3 or i == len(coorlist) - 3:
                            F_wind[i] = a * l * F_wc
                        elif i > 3 and i < len(coorlist) - 3:
                            F_wind[i] = a * ((((s / 10.) - (1.5 * l)) * F_wc) + (((2. * l) - (s / 10.)) * F_wa) +
((l / 2.) * F_wa))
                        else:
                            print '3. Number of nodes is out of range.'
                else:
                    print 'Number of fields is out of range. s/10 is greater than 2*l.'
                    break
    elif config == 'warren':
        if (nfields % 2. == 0):  # if true: number of fields is even
            for i in range(len(coorlist)):
                if zlist[i] == h:
                    if nfields <= 10:
                        if nfields == 2:
```

```
                if i == 1 or i == len(coorlist) - 1:
                    F_wind[i] = a * (((s / 10.) * F_wc) + (((l / 2.) - (s / 10.)) * F_wa))
                else:
                    F_wind[i] = a * l * F_wa
            elif nfields == 4:
                if i == 1 or i == len(coorlist) - 1:
                    F_wind[i] = a * (((s / 10.) * F_wc) + (((l / 2.) - (s / 10.)) * F_wa))
                else:
                    F_wind[i] = a * (1.5 * l) * F_wa
            else:
                if i == 1 or i == len(coorlist) - 1:
                    F_wind[i] = a * (l / 2.) * F_wc
                elif i == 2 or i == len(coorlist) - 3:
                    F_wind[i] = a * ((((s / 10.) - (l / 2.)) * F_wc) + ((l - (s / 10.)) * F_wa) + (l * F_wa))
                elif i > 2 and i < len(coorlist) - 3:
                    F_wind[i] = a * (2. * l) * F_wa
                else:
                    print '1. Number of nodes is out of range for config Warren.'
        elif 10 < nfields <= 20:
            if i == 1 or i == len(coorlist) - 1:
                F_wind[i] = a * (l / 2.) * F_wc
            elif i == 2 or i == len(coorlist) - 3:
                F_wind[i] = a * ((((s / 10.) - (l / 2.)) * F_wc) + (((2. * l) - (s / 10.)) * F_wa))
            elif i > 2 and i < len(coorlist) - 3:
                F_wind[i] = a * (2. * l) * F_wa
            else:
                print '2. Number of nodes is out of range for config Warren.'
        else:
            print 'Number of fields is out of range for config Warren(even). s/10 is greater than 2*l.'
            break
else: # number of fields is odd
    for i in range(len(coorlist)):
        if zlist[i] == h:
            if nfields <= 10:
                if nfields == 1:
                    F_wind[i] = a * (((s / 10.) * F_wc) + (((l / 2.) - (s / 10.)) * F_wa))
                elif nfields == 3:
                    if i == 1:
                        F_wind[i] = a * (((s / 10.) * F_wc) + ((l - (s / 10.)) * F_wa))
                    elif i == len(coorlist) - 1:
                        F_wind[i] = a * (((s / 10.) * F_wc) + (((l / 2.) - (s / 10.)) * F_wa))
                    else:
                        F_wind[i] = a * (1.5 * l) * F_wa
                else:
                    if i == 1:
                        F_wind[i] = a * (((s / 10.) * F_wc) + ((l - (s / 10.)) * F_wa))
                    elif i == len(coorlist) - 1:
                        F_wind[i] = a * (((l / 2.) * F_wc))
                    elif i == len(coorlist) - 3:
                        F_wind[i] = a * ((((s / 10.) - (l / 2.)) * F_wc) + (((2. * l) - (s / 10.)) * F_wa))
                    elif i > 1 and i < len(coorlist) - 3:
```

```
                F_wind[i] = a * (2. * l) * F_wa
            else:
                print '1. Number of nodes is out of range for config Warren.'
        elif 10 < nfields <= 20:
            if i == 1:
                F_wind[i] = a * l * F_wc
            elif i == len(coorlist) - 1:
                F_wind[i] = a * (((l / 2.) * F_wc))
            elif i == 3:
                F_wind[i] = a * ((((s / 10.) - l) * F_wc) + (((3. * l) - (s / 10.)) * F_wa))
            elif i == len(coorlist) - 3:
                F_wind[i] = a * ((((s / 10.) - (l / 2.)) * F_wc) + (((2. * l) - (s / 10.)) * F_wa))
            elif i > 3 and i < len(coorlist) - 3:
                F_wind[i] = a * (2. * l) * F_wa
            else:
                print '2. Number of nodes is out of range for config Warren.'
        else:
            print 'Number of fields is out of range for config Warren(odd). s/10 is greater than 2*l.'
            break
    else:
        print 'Configuration is not determined properly for windload calculation.'
    return F_wind


def snowload(vector):
    coorlist = coordinates(vector)[0]
    zlist = coordinates(vector)[3]

    with open('constants.txt') as f:
        for line in f:
            lines = line.split()
            if lines[0] == 's':
                s = float(lines[2])
            elif lines[0] == 'a':
                a = float(lines[2])
            elif lines[0] == 'mu':
                mu = float(lines[2])
            elif lines[0] == 'ce':
                ce = float(lines[2])
            elif lines[0] == 'ct':
                ct = float(lines[2])
            elif lines[0] == 'sk':
                sk = float(lines[2])
    h = vector[8]

    F_snow = mu * ce * ct * sk * s * a
    force_snow = np.zeros(len(coorlist))
    nnodes = 0.
    for z in zlist:
        if z == h:
            nnodes += 1.
```

```
    Fnode_snow  = F_snow / (nnodes - 1.)
    for i in range(len(zlist)):
        if zlist[i] == h:
            if i == 1 or i == len(zlist) - 1:
                force_snow[i] += Fnode_snow / 2.
            else:
                force_snow[i] += Fnode_snow
    return force_snow
```

Total load

```
import numpy as np
from optitruss.loads import permloads, windload, snowload

def totalload(rs1, rs_mid, rs2, ws1_st, ws2_st, ws1_dia, ws2_dia, vector,FC_type):
    if FC_type == 'FC2a':
        cf = 'down'
        Fwind = windload(cf, vector)
        Fsnow = np.zeros(len(Fwind))
        Fperm = permloads(rs1, rs_mid, rs2, ws1_st, ws2_st, ws1_dia, ws2_dia, vector)[0]
        Ftotal = 1.3 * Fperm + 1.65 * Fwind
    elif FC_type == 'FC2b':
        cf = 'up'
        Fwind = windload(cf, vector)
        Fsnow = np.zeros(len(Fwind))
        Fperm = permloads(rs1, rs_mid, rs2, ws1_st, ws2_st, ws1_dia, ws2_dia, vector)[0]
        Ftotal = 0.9 * Fperm + 1.65 * Fwind
    elif FC_type == 'FC3':
        Fsnow = snowload(vector)
        Fwind = np.zeros(len(Fsnow))
        Fperm = permloads(rs1, rs_mid, rs2, ws1_st, ws2_st, ws1_dia, ws2_dia, vector)[0]
        Ftotal = 1.3 * Fperm + 1.65 * Fsnow
    else:
        print 'The loadcase(FC) is not defined properly.'

    force_nodes = np.arange(1, len(Fperm) + 1)
    direction = []
    for i in range(len(Fperm)):
        direction.append('z')
    loadlist = []
    for i in range(len(Fperm)):
        loads = str(i + 1) + '=' + str(force_nodes[i]) + '@' + str(direction[i]) + '@' + str(-Ftotal[i])
        loadlist.append(loads)
    return loadlist, Ftotal, Fperm, Fwind, Fsnow
```

4. Code: Rekenmodule

In deze deelparagraaf is terug te vinden hoe de koppeling tussen Python en 3D Truss is gemaakt. Ten eerste is te zien hoe waarop de informatie vanuit Python weggeschreven wordt naar de .TRS-inputfile voor 3D Truss. Ten tweede is de code waarmee 3D Truss in batch mode gerund wordt in de deelparagraaf opgenomen. Ten slotte is ook terug te vinden hoe de output van 3D Truss wordt ingelezen in Python. Hierbij worden de staafkrachten, de knoopverplaatsing en de reactiekrachten met behulp van 3 verschillende functies ingelezen.

Write inputfile & Run 3D Truss

```
import subprocess
from optitruss.bartypes_totalload import bartypes, totalload
from optitruss.circle import Circle
from optitruss.convert_design_vector import convert_vector
from optitruss.coordinates import coordinates
from optitruss.elements import elements
from optitruss.supports import supports

def run_3DTruss(vector):
    with open('constants.txt') as f:
        for line in f:
            lines = line.split()
            if lines[0] == 'E':
                E = float(lines[2])
            elif lines[0] == 'supp_node':
                supp_node = eval(lines[2])
            elif lines[0] == 'supp_type':
                supp_type = eval(lines[2])

    conv_des_vec = convert_vector(vector)

    rs1 = Circle('rs1', conv_des_vec[0], conv_des_vec[1], conv_des_vec)
    rs_mid = Circle('rs_mid', conv_des_vec[0], conv_des_vec[1], conv_des_vec)
    rs2 = Circle('rs2', conv_des_vec[2], conv_des_vec[3], conv_des_vec)
    ws1_st = Circle('ws1_st', conv_des_vec[4], conv_des_vec[5], conv_des_vec)
    ws2_st = Circle('ws2_st', conv_des_vec[6], conv_des_vec[7], conv_des_vec)
    ws1_dia = Circle('ws1_dia', conv_des_vec[4], conv_des_vec[5], conv_des_vec)
    ws2_dia = Circle('ws2_dia', conv_des_vec[6], conv_des_vec[7], conv_des_vec)

    # WRITE INPUTFILE (.TRS)
    coorlist = coordinates(conv_des_vec)[0]
    elemlist = elements()[0]
    typelist = bartypes(rs1, rs_mid, rs2, ws1_st, ws2_st, ws1_dia, ws2_dia)[0]
    supportlist = supports(supp_node, supp_type, conv_des_vec)[0]

    for fc_type in ['FC2b', 'FC3']:
        loadlist = totalload(rs1, rs_mid, rs2, ws1_st, ws2_st, ws1_dia, ws2_dia,
                             conv_des_vec, fc_type)[0]
        filename = 'loadcase_' + fc_type + '.TRS'
        with open(filename, "w") as f:
            f.write('[settings]' + '\n')
            f.write('maxlength=' + str(100) + '\n')
```

```
        f.write('[properties]' + '\n')
        f.write('E=' + str(E) + '\n')
        f.write('A=' + str(1) + '\n')
        f.write('[coordinates]' + '\n')
        for i in range(len(coorlist)):
            f.write(str(coorlist[i]) + '\n')
        f.write('[elements]' + '\n')
        for i in range(len(elemlist)):
            f.write(str(elemlist[i]) + '\n')
        f.write('[barproperties]' + '\n')
        for i in range(len(typelist)):
            f.write(str(typelist[i]) + '\n')
        f.write('[loads]' + '\n')
        for i in range(len(loadlist)):
            f.write(str(loadlist[i]) + '\n')
        f.write('[supports]' + '\n')
        for i in range(len(supportlist)):
            f.write(str(supportlist[i]) + '\n')

    Program = 'C:/Users/Tirsa/Downloads/setup_truss3D/programfiles/w-it/truss3D/truss3D.exe'
    Inputfile = 'C:/Users/Tirsa/Desktop/tirsa bep/optitruss/' + filename
    p = subprocess.call([Program, '-i', Inputfile])
```

Read 3D Truss outputfile

```
def read_element_forces(barsnumb, config_parser):
    elemforcelist = []
    for i in range(barsnumb):
        data_elemforcelist = config_parser.get('elementforces', '%s' % (i+1))
        elemforce = float(data_elemforcelist)
        elemforcelist.append(elemforce)
    return elemforcelist

def read_reactions(suppnumb, config_parser):
    xreactlist = []
    yreactlist = []
    zreactlist = []
    for i in range(suppnumb):
        data_reactlist = config_parser.get('reactions', '%s' % (i+1))
        direction = data_reactlist.split('@')[1]
        if direction == 'x':
            xreaction = data_reactlist.split('@')[2]
            xreaction = float(xreaction)
            xreactlist.append(xreaction)
        if direction == 'y':
            yreaction = data_reactlist.split('@')[2]
            yreaction = float(yreaction)
            yreactlist.append(yreaction)
        if direction == 'z':
            zreaction = data_reactlist.split('@')[2]
            zreaction = float(zreaction)
```

```
            zreactlist.append(zreaction)
    return xreactlist, yreactlist, zreactlist

def read_displacements(nodesnumb, config_parser):
    xdisplist = []
    ydisplist = []
    zdisplist = []
    for i in range(nodesnumb * 3):
        data_displist = config_parser.get('displacements', '%s' % (i+1))
        direction = data_displist.split('@')[1]
        if direction == 'x':
            xdisp = data_displist.split('@')[2]
            xdisp = float(xdisp)
            xdisplist.append(xdisp)
        if direction == 'y':
            ydisp = data_displist.split('@')[2]
            ydisp = float(ydisp)
            ydisplist.append(ydisp)
        if direction == 'z':
            zdisp = data_displist.split('@')[2]
            zdisp = float(zdisp)
            zdisplist.append(zdisp)
    return xdisplist, ydisplist, zdisplist
```

5. Code: Toetsingsmodule

In deze deelparagraaf is te vinden hoe de toetsingsmodule verwerkt is in het computermodel. Hier hoort ook de code bij waarmee de capaciteit van de staven wordt berekend. In deze deelparagraaf is te zien hoe de inequality constraints zijn opgesteld en in welke vorm zij meegegeven worden aan het optimalisatie algoritme.

Constraints

```
import numpy as np
from optitruss.circle import Circle
from optitruss.convert_design_vector import convert_vector
from optitruss.elements import elements
from optitruss.coordinates import coordinates
from optitruss.read_3DTruss_output import read_element_forces, \
  read_displacements
from optitruss.run_3DTruss import run_3DTruss
from optitruss.write_history import write_history_file

def inequality_constraints(vector):
  run_3DTruss(vector)
  converted_design_vector = convert_vector(vector)
  normal_checks, buckling_checks, displ_checks = unitychecks(converted_design_vector)
  gamma_checks, beta_checks, tau_checks, theta_check =jointconstraints(converted_design_vector)

  normal_force_ineq_constraints = [0.99 - normal_check for normal_check in normal_checks]
  buckling_ineq_constraints = [0.99 - buckling_check for buckling_check in buckling_checks]
  displ_ineq_constraints = [0.99 - displ_check for displ_check in displ_checks]
  gamma_ineq_constraints = [50. - gamma_check for gamma_check in gamma_checks]
  beta_ineq_constraints1 = [1. - beta_check for beta_check in beta_checks]
  beta_ineq_constraints2 = [-0.2 + beta_check for beta_check in beta_checks]
  tau_ineq_constraints = [-2. + tau_check for tau_check in tau_checks]
  theta_ineq_constraints1 = [np.pi/3. - theta_check]
  theta_ineq_constraints2 = [-np.pi/6. + theta_check]

  write_history_file(str(normal_checks), 'normal_checks')
  write_history_file(str(buckling_checks), 'buckling_checks')
  write_history_file(str(displ_checks), 'displ_checks')
  write_history_file(str(gamma_checks), 'gamma_checks')
  write_history_file(str(beta_checks), 'beta_checks')
  write_history_file(str(tau_checks), 'tau_checks')
  write_history_file(str(theta_check), 'theta_check')

  return normal_force_ineq_constraints + buckling_ineq_constraints + displ_ineq_constraints \
      + gamma_ineq_constraints + beta_ineq_constraints1 + beta_ineq_constraints2 \
      + tau_ineq_constraints + theta_ineq_constraints1 + theta_ineq_constraints2


def capacity(rs1, rs_mid, rs2, ws1_st, ws2_st, ws1_dia, ws2_dia, barsnumb, config_parser):
  with open('constants.txt') as f:
    for line in f:
      lines = line.split()
      if lines[0] == 'E':
```

```
            E = float(lines[2])
        elif lines[0] == 'fy':
            fy = float(lines[2])
        elif lines[0] == 'pform':
            pform = lines[2]

    N_Ed = read_element_forces(barsnumb, config_parser)
    elemtypes = elements()[3]
    if pform == 'koudgevormd':
        alfa = 0.49
    if pform == 'warmvervaardigd':
        alfa = 0.21

# Determine capacity for:  Normalforce and Bucklingforce
    sl_e = np.pi * np.sqrt((E / 1000) / fy)
    N_Rdlist = []
    N_b_Rdlist = []
    chi_list = []
    for i in range(len(elemtypes)):
        if elemtypes[i] == 1:
            N_Rd = fy * rs1.area() * 10 ** 3
            N_Rdlist.append(N_Rd)
            if N_Ed[i] < 0:
                sl = rs1.length() / np.sqrt(rs1.inertia() / rs1.area())
                sl_rel = sl / sl_e
                phi = 0.5 * (1 + alfa * (sl_rel - 0.2) + sl_rel ** 2)
                chi = 1 / (phi + np.sqrt(phi ** 2 - sl_rel ** 2))
                N_b_Rd = chi * N_Rd
                N_b_Rdlist.append(N_b_Rd)
                chi_list.append(chi)
            else:
                N_b_Rdlist.append('NaN')
                chi_list.append('NaN')
        elif elemtypes[i] == 2:
            N_Rd = fy * ws1_st.area() * 10 ** 3
            N_Rdlist.append(N_Rd)
            if N_Ed[i] < 0:
                sl = ws1_st.length() / np.sqrt(ws1_st.inertia() / ws1_st.area())
                sl_rel = sl / sl_e
                phi = 0.5 * (1 + alfa * (sl_rel - 0.2) + sl_rel ** 2)
                chi = 1 / (phi + np.sqrt(phi ** 2 - sl_rel ** 2))
                N_b_Rd = chi * N_Rd
                N_b_Rdlist.append(N_b_Rd)
                chi_list.append(chi)
            else:
                N_b_Rdlist.append('NaN')
                chi_list.append('NaN')
        elif elemtypes[i] == 3:
            N_Rd = fy * ws1_dia.area() * 10 ** 3
            N_Rdlist.append(N_Rd)
            if N_Ed[i] < 0:
```

```
            sl = ws1_dia.length() / np.sqrt(ws1_dia.inertia() / ws1_dia.area())
            sl_rel = sl / sl_e
            phi = 0.5 * (1 + alfa * (sl_rel - 0.2) + sl_rel ** 2)
            chi = 1 / (phi + np.sqrt(phi ** 2 - sl_rel ** 2))
            N_b_Rd = chi * N_Rd
            N_b_Rdlist.append(N_b_Rd)
            chi_list.append(chi)
        else:
            N_b_Rdlist.append('NaN')
            chi_list.append('NaN')
    elif elemtypes[i] == 4:
        N_Rd = fy * rs_mid.area() * 10 ** 3
        N_Rdlist.append(N_Rd)
        if N_Ed[i] < 0:
            sl = rs_mid.length() / np.sqrt(rs_mid.inertia() / rs_mid.area())
            sl_rel = sl / sl_e
            phi = 0.5 * (1 + alfa * (sl_rel - 0.2) + sl_rel ** 2)
            chi = 1 / (phi + np.sqrt(phi ** 2 - sl_rel ** 2))
            N_b_Rd = chi * N_Rd
            N_b_Rdlist.append(N_b_Rd)
            chi_list.append(chi)
        else:
            N_b_Rdlist.append('NaN')
            chi_list.append('NaN')
    elif elemtypes[i] == 5:
        N_Rd = fy * rs2.area() * 10 ** 3
        N_Rdlist.append(N_Rd)
        if N_Ed[i] < 0:
            sl = rs2.length() / np.sqrt(rs2.inertia() / rs2.area())
            sl_rel = sl / sl_e
            phi = 0.5 * (1 + alfa * (sl_rel - 0.2) + sl_rel ** 2)
            chi = 1 / (phi + np.sqrt(phi ** 2 - sl_rel ** 2))
            N_b_Rd = chi * N_Rd
            N_b_Rdlist.append(N_b_Rd)
            chi_list.append(chi)
        else:
            N_b_Rdlist.append('NaN')
            chi_list.append('NaN')
    elif elemtypes[i] == 6:
        N_Rd = fy * ws2_st.area() * 10 ** 3
        N_Rdlist.append(N_Rd)
        if N_Ed[i] < 0:
            sl = ws2_st.length() / np.sqrt(ws2_st.inertia() / ws2_st.area())
            sl_rel = sl / sl_e
            phi = 0.5 * (1 + alfa * (sl_rel - 0.2) + sl_rel ** 2)
            chi = 1 / (phi + np.sqrt(phi ** 2 - sl_rel ** 2))
            N_b_Rd = chi * N_Rd
            N_b_Rdlist.append(N_b_Rd)
            chi_list.append(chi)
        else:
            N_b_Rdlist.append('NaN')
```

```
            chi_list.append('NaN')
        elif elemtypes[i] == 7:
            N_Rd = fy * ws2_dia.area() * 10 ** 3
            N_Rdlist.append(N_Rd)
            if N_Ed[i] < 0:
                sl = ws2_dia.length() / np.sqrt(ws2_dia.inertia() / ws2_dia.area())
                sl_rel = sl / sl_e
                phi = 0.5 * (1 + alfa * (sl_rel - 0.2) + sl_rel ** 2)
                chi = 1 / (phi + np.sqrt(phi ** 2 - sl_rel ** 2))
                N_b_Rd = chi * N_Rd
                N_b_Rdlist.append(N_b_Rd)
                chi_list.append(chi)
            else:
                N_b_Rdlist.append('NaN')
                chi_list.append('NaN')
        else:
            print 'Element type is not defined properly.'
    return N_Rdlist, N_b_Rdlist, chi_list


def unitychecks(vector):
    import ConfigParser
    with open('constants.txt') as f:
        for line in f:
            lines = line.split()
            if lines[0] == 's':
                s = float(lines[2])
            elif lines[0] == 'correction':
                correction = float(lines[2])

    d_rs1 = vector[0]
    t_rs1 = vector[1]
    d_rs2 = vector[2]
    t_rs2 = vector[3]
    d_ws1 = vector[4]
    t_ws1 = vector[5]
    d_ws2 = vector[6]
    t_ws2 = vector[7]

    rs1 = Circle('rs1', d_rs1, t_rs1, vector)
    rs_mid = Circle('rs_mid', d_rs1, t_rs1, vector)
    rs2 = Circle('rs2', d_rs2, t_rs2, vector)
    ws1_st = Circle('ws1_st', d_ws1, t_ws1, vector)
    ws2_st = Circle('ws2_st', d_ws2, t_ws2, vector)
    ws1_dia = Circle('ws1_dia', d_ws1, t_ws1, vector)
    ws2_dia = Circle('ws2_dia', d_ws2, t_ws2, vector)

    coorlist = coordinates(vector)[0]
    elemlist = elements()[0]
    nodesnumb = len(coorlist)
    barsnumb = len(elemlist)
```

```
    Lrep = 2 * s

    UC_N = []
    UC_Nb = []
    UC_z = []
    test = 0
    for fc_type in ['FC2b', 'FC3']:
        filename = 'loadcase_' + fc_type + '.TRS'
        config_parser = ConfigParser.ConfigParser()
        config_parser.read(filename)

        N_Rd, N_b_Rd, chi_list = capacity(rs1, rs_mid, rs2, ws1_st, ws2_st, ws1_dia, ws2_dia,
                                          barsnumb, config_parser)
        if fc_type == 'FC3':
            write_history_file(str(chi_list), 'chi_FC3')
        else:
            write_history_file(str(chi_list), 'chi_FC2b')

        N_Ed = read_element_forces(barsnumb, config_parser)
        zdisp = read_displacements(nodesnumb, config_parser)[2]

# Write lists of Unity Checks for both Normalforce and Bucklingforce:
        for i in range(len(N_Ed)):
            UC_N.append(abs(N_Ed[i]) / N_Rd[i])
        for i in range(len(N_Ed)):
            if N_b_Rd[i] == 'NaN' or N_b_Rd[i] == 0:
                pass
            else:
                UC_Nb.append(abs(N_Ed[i]) / N_b_Rd[i])
        for i in range(len(zdisp)):
            UC_z.append(abs(zdisp[i]) / (correction * 0.004 * Lrep))

# Warning in case the design fails for one of the constraints:
        for i in range(len(N_Ed)):
            if abs(N_Ed[i]) > N_Rd[i]:
                test += 1
                print 'De constructie faalt, doordat de normaalkrachtcapaciteit', N_Rd[i],\
                    'in staaf', i + 1, 'lager is dan de belasting', abs(N_Ed[i])
            if N_Ed[i] < 0:
                if abs(N_Ed[i]) > N_b_Rd[i]:
                    test += 1
                    print 'De constructie faalt, doordat de knikcapaciteit', N_b_Rd[i],\
                        'in staaf', i + 1, 'lager is dan de belasting', abs(N_Ed[i])
        for i in range(nodesnumb):
            if abs(zdisp[i]) > correction * 0.004 * Lrep:
                test += 1
                print 'De verticale verplaatsing van knooppunt ', i + 1, 'is te groot.'
        if test > 0:
            print 'De constructie voldoet NIET voor loadcase: ' + fc_type + '.'
        else:
            print 'De constructie voldoet voor loadcase: ' + fc_type + '.'
```

```
    return UC_N, UC_Nb, UC_z


def jointconstraints(vector):
    with open('constants.txt') as f:
        for line in f:
            lines = line.split()
            if lines[0] == 's':
                s = float(lines[2])
            elif lines[0] == 'nfields':
                nfields = int(lines[2])

    d_rs1 = vector[0]
    t_rs1 = vector[1]
    d_ws1 = vector[4]
    t_ws1 = vector[5]
    h = vector[8]

    gamma = []
    beta = []
    tau = []
    gamma.append(d_rs1/t_rs1)
    gamma.append(d_ws1/t_ws1)
    beta.append(d_ws1/d_rs1)
    tau.append(t_rs1/t_ws1)
    theta = np.arctan(h / (s / nfields))

# Warning in case the design fails for one of the constraints:
    for gam in gamma:
        if gam > 50:
            print 'De constructie voldoet niet aan de d/t-verhouding.'
    for bet in beta:
        if bet < 0.2:
            print 'De constructie voldoet niet aan de diameterverhouding tussen rand- en wandstaven.'
        elif bet > 1.:
            print 'De constructie voldoet niet aan de diameterverhouding tussen rand- en wandstaven.'
    if tau < 2.:
        print 'De constructie voldoet niet aan de wanddikteverhouding tussen rand- en wandstaven.'
    if theta < (np.pi / 6) or theta > (np.pi / 3):
        print 'De constructie voldoet niet aan de eis m.b.t. theta'
    return gamma, beta, tau, theta
```

6. Code: Objective function

De objective function is de functie die de totale kosten van de vakwerkligger als output geeft. In deze deelparagraaf is deze functie terug te vinden. Daarnaast zijn ook de functies die de totale kosten en de deelkosten berekenen, opgenomen in deze deelparagraaf.

Objective function:

```
from optitruss.convert_design_vector import convert_vector
from optitruss.draw_geometry import draw_truss_elements
from optitruss.cost_per_part import welded_joints, cost_bars, prefab_joints, total_costs
from optitruss.write_history import write_history_file


def objective_function(vector, display_geometry=False):
    converted_design_vector = convert_vector(vector)
    write_history_file(str(converted_design_vector), 'converted_vector')

    if display_geometry:
        draw_truss_elements(converted_design_vector)
    cost_total = total_costs(converted_design_vector)
    print 'Totale kosten zijn: ' + str(cost_total)
    with open('init_value.txt') as f:
        for line in f:
            lines = line.split()
    value = float(cost_total) / float(lines[0])
    print 'obj func value: ' + str(value)
    write_history_file(str(value), 'obj_value')
    write_history_file(str(vector), 'obj_vector')
    return value
```

Cost per part:

```
from optitruss.circle import Circle
from optitruss.coordinates import coordinates
from optitruss.elements import elements
from optitruss.write_history import write_history_file

def total_costs(converted_design_vector):
    with open('constants.txt') as f:
        for line in f:
            lines = line.split()
            if lines[0] == 'jointtype':
                jointtype = lines[2]

    if jointtype == 'welded':
        cost_joints = welded_joints(converted_design_vector)
    elif jointtype == 'prefab':
        cost_joints = prefab_joints(converted_design_vector)
    else:
        print 'Jointtype is not determined properly.'
    cost_elements = cost_bars(converted_design_vector)
    write_history_file(str(cost_elements), 'cost_elements_total')
```

```
    write_history_file(str(cost_joints), 'cost_joints_total')
    return cost_joints + cost_elements

def welded_joints(vector):
    with open('constants.txt') as f:
        for line in f:
            lines = line.split()
            if lines[0] == 'config':
                config = lines[2]
            elif lines[0] == 'price_weldvolume':
                price_weldvolume = float(lines[2])

    d_ws1 = vector[4]
    t_ws1 = vector[5]

    coorlist = coordinates(vector)[0]
    startlist = elements()[1]
    endlist = elements()[2]

    ws1_st = Circle('ws1_st', d_ws1, t_ws1, vector)
    ws1_dia = Circle('ws1_dia', d_ws1, t_ws1, vector)

    joints = []
    weldvolume = []
    for i in range(len(coorlist)):
        joints.append(startlist.count(i+1) + endlist.count(i+1))
    for i in range(len(joints)):
        if joints[i] == 2:
            # NODE_T (staander & randstaaf)
            V_weld = ws1_st.periphery() * (0.35 * t_ws1)**2
            weldvolume.append(V_weld)
        elif joints[i] == 3:
            # NODE_N (staander, diagonal & randstaaf)
            V_weld = ws1_st.periphery() * (0.35 * t_ws1) ** 2 + ws1_dia.periphery() * (0.35 * t_ws1) ** 2
            weldvolume.append(V_weld)
        elif joints[i] == 4:
            if config == 'howe' or config == 'pratt':
                # NODE_N (staander, diagonal & randstaaf)
                V_weld = ws1_st.periphery() * (0.35 * t_ws1)**2 + ws1_dia.periphery() * (0.35 * t_ws1)**2
                weldvolume.append(V_weld)
            elif config == 'warren':
                # NODE_K (2 diagonalen & randstaaf)
                V_weld = 2 * (ws1_dia.periphery() * (0.35 * t_ws1) ** 2)
                weldvolume.append(V_weld)
            else:
                print 'Configuration for jointtype is unkown.'
        else:
            print 'Jointtype is unknown for welded joints.'
        cost_weldedjoints = (sum(weldvolume) * 1e9) * price_weldvolume
    return cost_weldedjoints
```

```
def prefab_joints(vector):
    with open('constants.txt') as f:
        for line in f:
            lines = line.split()
            if lines[0] == 'price_prefabjoint2':
                price_prefabjoint2 = float(lines[2])
            elif lines[0] == 'price_prefabjoint3':
                price_prefabjoint3 = float(lines[2])
            elif lines[0] == 'price_prefabjoint4':
                price_prefabjoint4 = float(lines[2])

    coorlist = coordinates(vector)[0]
    startlist = elements()[1]
    endlist = elements()[2]

    joints = []
    for i in range(len(coorlist)):
        joints.append(startlist.count(i + 1) + endlist.count(i + 1))
    price_prefab_joint = []
    for i in range(len(joints)):
        if joints[i] == 2:
            price_joint = price_prefabjoint2
            price_prefab_joint.append(price_joint)
        elif joints[i] == 3:
            price_joint = price_prefabjoint3
            price_prefab_joint.append(price_joint)
        elif joints[i] == 4:
            price_joint = price_prefabjoint4
            price_prefab_joint.append(price_joint)
        else:
            print 'Jointtype is unknown for prefab joints.'
    cost_prefabjoints = sum(price_prefab_joint)
    return cost_prefabjoints


def cost_bars(vector):
    d_rs1 = vector[0]
    t_rs1 = vector[1]
    d_rs2 = vector[2]
    t_rs2 = vector[3]
    d_ws1 = vector[4]
    t_ws1 = vector[5]
    d_ws2 = vector[6]
    t_ws2 = vector[7]

    elemtypes = elements()[3]

    rs1 = Circle('rs1', d_rs1, t_rs1, vector)
    rs_mid = Circle('rs_mid', d_rs1, t_rs1, vector)
    ws1_st = Circle('ws1_st', d_ws1, t_ws1, vector)
```

```
ws1_dia = Circle('ws1_dia', d_ws1, t_ws1, vector)
rs2 = Circle('rs2', d_rs2, t_rs2, vector)
ws2_st = Circle('ws2_st', d_ws2, t_ws2, vector)
ws2_dia = Circle('ws2_dia', d_ws2, t_ws2, vector)

price_bars = []
for i in range(len(elemtypes)):
    if elemtypes[i] == 1:
        A = rs1.area()
        L = rs1.length()
    elif elemtypes[i] == 2:
        A = ws1_st.area()
        L = ws1_st.length()
    elif elemtypes[i] == 3:
        A = ws1_dia.area()
        L = ws1_dia.length()
    elif elemtypes[i] == 4:
        A = rs_mid.area()
        L = rs_mid.length()
    elif elemtypes[i] == 5:
        A = rs2.area()
        L = rs2.length()
    elif elemtypes[i] == 6:
        A = ws2_st.area()
        L = ws2_st.length()
    elif elemtypes[i] == 7:
        A = ws2_dia.area()
        L = ws2_dia.length()
    else:
        print 'Bartype is unknown for calculation of cost per element.'
    price_bars.append(((10 ** 7 * A ** 2) + (27130 * A) + 3.641) * L)
cost_elements = sum(price_bars)
return cost_elements
```

7. Code: Resultaten wegschrijven

De (tussen-)resultaten van het optimalisatieproces worden weggeschreven naar een textfile. In deze deelparagraaf wordt aangegeven hoe dit gedaan wordt.

Write history

```
import errno
import os

def write_history_file(variables, data_type):
    if data_type not in ['obj_value', 'obj_vector', 'optim_summary',
                         'normal_checks', 'buckling_checks', 'displ_checks',
                         'gamma_checks', 'beta_checks', 'tau_checks',
                         'theta_check', 'summary', 'converted_vector',
                         'chi_FC2b', 'chi_FC3',
                         'cost_elements_total', 'cost_joints_total']:
        print 'choose applicable data type, no file is saved now'
        pass
    else:
        with open('constants.txt') as f:
            for line in f:
                lines = line.split()
                if lines[0] == 'nfields':
                    nfields = lines[2]
                elif lines[0] == 'config':
                    config = lines[2]
                elif lines[0] == 'jointtype':
                    jointtype = lines[2]
        try:
            folder = 'output/results' + '_' + str(nfields) + '_' + config + \
                     '_' + jointtype
            os.makedirs(folder)
        except OSError as exception:
            if exception.errno != errno.EEXIST:
                raise
        if data_type == 'summary':
            filename = 'output/' + data_type + '.txt'
        else:
            filename = folder + '/' + data_type + '.txt'
        with open(filename, "a") as f:
            f.write(variables + '\n')
```

Initial objective function

```
from optitruss.convert_design_vector import convert_vector
from optitruss.cost_per_part import total_costs
def initial_obj_value(vector):
    converted_design_vector = convert_vector(vector)
    cost_total = total_costs(converted_design_vector)
    filename = 'init_value.txt'
    with open(filename, "w") as f:
        f.write(str(float(cost_total)))
    return cost_total
```

8. Code: Optimalisatie algoritme

Deze deelparagraaf bevat de code waarmee het optimalisatie algoritme in het computermodel is geïmplementeerd. Naast de code die nodig is om dit optimalisatie algoritme te runnen, is ook de code opgenomen waarmee een specifiek ontwerp doorgerekend kan worden.

Run optimization

```
import os
import time
from scipy.optimize import minimize, leastsq
from optitruss.constraints import inequality_constraints
from optitruss.convert_design_vector import convert_vector
from optitruss.cost_per_part import total_costs
from optitruss.initial_objective_func import initial_obj_value
from optitruss.objective_func import objective_function
from optitruss.write_config_file import write_constants_file
from optitruss.write_history import write_history_file

if os.path.exists('output'):
   import shutil
   shutil.rmtree('output')

results = []
for n_fields in xrange(4, 21):
   for config in ['howe', 'pratt', 'warren']:
      for jointtype in ['welded', 'prefab']:
         write_constants_file(n_fields, config, jointtype)
         start_vec = [0.5] * 5
         bounds = []
         t0 = time.time()

         initial_obj_value(start_vec)
         initial_design_vector = convert_vector(start_vec)
         write_history_file(str(start_vec), 'obj_vector')
         initial_costs = total_costs(initial_design_vector)
         write_history_file(str(1), 'obj_value')

         bounds_d = (0.1, 1.5)  # initial value is 380. resp. 260. mm
         bounds_t = (0.1, 1.5)  # initial value is 16. resp. 8. mm
         bounds_h = (0.35, 1.25)  # initial value is 3. m

for i in range(2):
         bounds.append(bounds_d)
         bounds.append(bounds_t)
         bounds.append(bounds_h)
         cons = {'type': 'ineq', 'fun': inequality_constraints}
         optim2 = minimize(fun=objective_function, x0=start_vec,
                           bounds=bounds,
                           method='SLSQP', constraints=cons,
                           options={'disp': True,
                                    'eps': 0.01,
                                    'ftol': 1e-02})
```

```
        print optim2.x
        print optim2.success
        print optim2.message
        converted_design_vector = convert_vector(optim2.x)
        optimized_costs = total_costs(converted_design_vector)
        print optimized_costs
        t1 = time.time()

        write_history_file(('optimized design vector: ' + str(optim2.x)), 'optim_summary')
        write_history_file(('initial value: ' + str(initial_costs)), 'optim_summary')
        write_history_file(('optimized value: ' + str(optimized_costs)), 'optim_summary')
        write_history_file(('optimizstion time: ' + str(t1 - t0)),'optim_summary')

        with open('constants.txt') as f:
            for line in f:
                lines = line.split()
                if lines[0] == 'nfields':
                    nfields = lines[2]
                elif lines[0] == 'config':
                    config = lines[2]
                elif lines[0] == 'jointtype':
                    jointtype = lines[2]

        variable = str([(str(nfields) + '_' + config + '_' + jointtype), optimized_costs])
        write_history_file(variable, 'summary')

import datetime
ts = time.time()
st = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H_%M_%S')
new_folder_name = st
os.rename('output', 'output ' + new_folder_name)
```

Convert design vector

```
def convert_vector(vector):
    d_rs1 = vector[0] * 0.38
    t_rs1 = vector[1] * 0.016
    d_rs2 = 0.3
    t_rs2 = 0.016
    d_ws1 = vector[2] * 0.26
    t_ws1 = vector[3] * 0.008
    d_ws2 = 0.3
    t_ws2 = 0.008
    h = vector[4] * 3.0
    return [d_rs1, t_rs1, d_rs2, t_rs2, d_ws1, t_ws1, d_ws2, t_ws2, h]
```

Run separate design

```
from optitruss.constraints import inequality_constraints
from optitruss.initial_objective_func import initial_obj_value
from optitruss.objective_func import objective_function
```

```
from optitruss.write_config_file import write_constants_file

start_vec = […,  …,  …,  …,  …]
write_constants_file(n=10, config='warren', jointtype='welded')
initial_obj_value(start_vec)
print objective_function(start_vec)
print inequality_constraints(start_vec)
```

G. Zelfevaluatie & Besprekingsverslagen

Het onderzoek dat in dit rapport besproken is, heb ik met veel plezier uitgevoerd. Hoewel de weg er naartoe soms wat hobbelig was, ben ik erg tevreden met het resultaat.

De start van dit onderzoek verliep vrij moeizaam. Het onderwerp dat ik voor de zomervakantie had gekozen, bleek niet meer aangeboden te worden. Ik wilde graag een onderzoek doen waarbij programmeren een grote rol speelde. De paar vakken uit de bachelor opleiding waarbij geprogrammeerd moest worden, vond ik namelijk erg interessant. Ik wilde graag testen of ik programmeren ook leuk zou vinden als dagelijkse bezigheid. Aangezien er geen andere onderwerpen over programmeren werden aangeboden, besloot ik zelf een onderwerp te bedenken. Mijn gebrek aan ervaring met programmeren hielp hierbij niet mee. Het lukte niet om mijn ideeën duidelijk te verwoorden, doordat ik de termen van het programmeren niet kende. Hierdoor kon ik niet goed tot een onderwerp komen waar zowel mijn twee eerste begeleiders, Roland Abspoel en Peter de Vries, als ik enthousiast over waren. Gelukkig heeft Pierre Hoogenboom mij hiermee kunnen helpen. Met zijn hulp en met de inspiratie die ik uit de gesprekken haalde, is het mij gelukt om mijn ideeën op papier te zetten. In het begin van de tweede week heb ik mijn onderzoeksvraag en werkplan gepresenteerd aan Peter en Pierre. Met hun goedkeuring kon ik toen eindelijk echt beginnen met mijn onderzoek.

Programmeren was voor mij een heel nieuw onderwerp. De enige ervaring die ik had, had ik opgedaan door het schrijven van kleine stukjes code voor zeer basale programmeeropdrachten. Ik had echter, voordat ik aan dit onderzoek begon, nog nooit zonder een concrete opdracht iets geprogrammeerd. Ik liep er tegenaan dat ik hierdoor niet goed een planning kon maken voor mijn onderzoek. Het lukte me niet om in te schatten hoeveel tijd ik nodig zou hebben om bepaalde onderdelen van het computermodel te ontwikkelen. Dit riep veel stress op. Ik werk altijd gestructureerd en het geeft mij rust om te weten hoever ik ben en wat ik nog moet doen. Ik heb dit probleem opgelost door de onderzoeksperiode op te splitsen in kleine stukken. Per week heb ik een planning gemaakt en deze heb ik per dag bijgesteld afhankelijk van wat ik die dag had afgerond. Op die manier kreeg ik steeds meer inzicht in de benodigde tijd voor de verschillende taken. Naarmate het onderzoek vorderde, hoefde ik mijn weekplanning steeds minder bij te stellen. Ook de stress werd hierdoor minder.

Ondanks de moeite die ik had met het maken van de planning, ben ik erg tevreden over het verloop van mijn onderzoek. Inhoudelijk gezien ben ik geen enkele keer echt vastgelopen. Daarbij bleef ik het programmeren leuk vinden. Ik heb ontzettend veel geleerd van dit onderzoek. Ik heb veel inzicht verkregen in de manier waarop ontwerptaken geautomatiseerd kunnen worden. Gedurende het onderzoek heb ik regelmatig mijn code aangepast en verbeterd, maar het hele model aanpassen iedere keer dat ik iets nieuws leerde, was niet te doen in de beperkte duur van dit onderzoek.

Ik kan me er mateloos aan ergeren dat mijn model niet "*helemaal klopt"*. Als ik dit onderzoek opnieuw zou uitvoeren met de kennis die ik nu heb, zou ik het computermodel heel anders opbouwen. Ik heb nu meer inzicht in de (onderlinge) afhankelijkheden van de vele variabelen die een rol spelen bij de optimalisatie van een vakwerkligger. Ook ken en beheers ik nu meer functies dan toen ik begon aan dit onderzoek. Hierdoor zou ik de code een stuk compacter kunnen maken en overzichtelijker kunnen ordenen.

Door dit onderzoek heb ik ontdekt dat ik programmeren erg interessant vind. Ik zou gerust nog een kwartaal (of meer) kunnen besteden aan het uitbreiden en verbeteren van mijn computermodel. Ik heb nooit goed geweten wat ik na mijn studie zou willen doen. Het lijkt erop dat ik nu eindelijk een vakgebied gevonden heb waarin ik mezelf wel zie werken.

Werkplanbespreking 13 september 2016

Dit was de eerste meeting met zowel Peter de Vries, vervangend eerste begeleider, als Pierre Hoogenboom, tweede begeleider. Roland Abspoel, eerste begeleider, is drie weken afwezig en wordt gedurende die periode op de hoogte gehouden via de mail.

De dag voor de bespreking is de startnotitie gemaild naar de begeleiders. Gezien de moeizame start in de eerste week, waarin nog veel onduidelijk bleef, was het nodig om allereerst toe te lichten welke ontwikkelingen hadden plaatsgevonden en hoe uiteindelijk het voorgelegde projectvoorstel tot stand was gekomen. In de eerste week moest het onderwerp gedefinieerd worden, wat erg lastig bleek te zijn en de doelstelling veranderde gedurende die week vaak. Beide begeleiders gaven aan dat de projectomschrijving de duidelijkheid gaf, die de voorgaande week nog ontbrak. De startnotitie is door beide begeleiders goedgekeurd, maar er werd wel aangeraden om de vraagstelling scherper te formuleren en ook de deelvragen nog eens goed te bekijken.

Pierre Hoogenboom raadde aan om vooraf vast na te denken over de vorm van de conclusie en de mogelijke antwoorden op de deelvragen. Ook zei hij dat het handig is om tijdens het programmeren stil te staan bij de knelpunten en de mogelijkheden, die ontdekt worden. Wellicht kunnen daar mogelijke conclusies uit getrokken worden of komen er vervolgvragen naar voren.

Het maken van een duidelijke planning was nog niet gelukt. De reden hiervoor was dat ik door gebrek aan ervaring met programmeren niet goed in kon schatten hoeveel tijd er nodig zou zijn voor de verschillende taken/onderdelen. Dit is besproken met de begeleiders en er werd aangeraden om een planning te maken die 1 á 2 weken vooruit kijkt. Gedurende het verloop van het project kan de planning dan aangescherpt en aangevuld worden. Er is afgesproken dat de planning voor de komende week de dag na de bespreking wordt gemaild.

Tot slot is afgesproken hoe gestart gaat worden met het ontwikkelen van het model. In het plan van aanpak was beschreven dat eerst voor een 2D-vakwerk een model met beperkte variabelen geschreven zou worden. Vervolgens kunnen daar dan steeds meer variabelen aan toegevoegd worden en kan uiteindelijk ook de stap van 2D naar 3D gemaakt worden. Voor de volgende afspraak dient een doel gesteld te worden, zodat er dan concreet een up-date gegeven kan worden over dit onderdeel van het onderzoek.

Tussenpeiling 6 oktober 2016

Bij de officiële tussenpeiling waren alledrie de begeleiders, Peter de Vries, Roland Abspoel en Pierre Hoogenboom aanwezig.

Het tussenrapport is twee dagen voor de tussenpeiling gemaild naar de begeleiders om hen voldoende gelegenheid te bieden deze door te nemen. Aan de hand van een PowerPoint presentatie zijn de voortgang van het project, de knelpunten en het plan van aanpak voor de resterende tijd gepresenteerd. Gedurende de presentatie was er gelegenheid voor vragen en opmerkingen.

De drie begeleiders hebben allen een positieve beoordeling gegeven. De feedback die gegeven is, is genoteerd en hieronder in de vorm van een opsomming weergegeven.

- *Vlakke* i.p.v. *platte* vakwerkliggers
- *Velden* i.p.v. *blokken*
- *Belasting door goederen & personen* i.p.v. *Opgelegde belasting*
- Windbelasting:
    - Toelichting bij bepaling van de bouwwerkfactor;
    - Waarom kan bij dit onderzoek een waarde van 1,0 gebruikt worden?
- SLS vs. ULS:
    - Controle van de SLS én de ULS wordt gedaan aan de hand van een enkele berekening waarbij veiligheidsfactoren worden toegepast
      → Hoe wordt hiermee omgegaan bij de controle van de vervormingen?
- Stijfheidseis:

    - Maximaal toegestane vervorming verhogen i.v.m. toepassing van de veiligheidsfactoren
      → Welke factor wordt hiervoor gebruikt?
    - Aanname = vervormingen zijn níet maargevend
      → Extra waarschuwing in het model opnemen indien deze aanname níet klopt! (als de slankheid van de vakwerkligger te groot wordt, kan de stijfheidseis wel maatgevend worden!)
- Discrete variabelen:
    - Hoe worden deze verwerkt in het optimalisatievraagstuk?
- Optimalisatie naar kosten:
    - Waar hangen de kosten van de verbindingen vanaf?
        - Type: lassen vs. systeemknopen
        - Kosten van lasverbindingen zijn afhankelijk van:
            - Volume van de las: staafdiameter en –wanddikte
        - Kosten van systeemverbindingen zijn afhankelijk van:
            - Aantal te verbinden staven
- Opleggingen in 3D Truss:
    - Geef aan dat de ligger in de knooppunten zijwaartse ondersteund wordt. Dit verhoogt de rekensnelheid, omdat het programme dan niet zelf de kinematisch onbepaalde matrix om hoeft te schrijven naar een kinematisch bepaald probleem.
- Bronvermelding:
    - Nummering van de bronnen hoort niet bij de toegepaste referentiestijl.

Verder is besproken hoe de laatste weken besteed zullen worden. Het model wordt in principe niet meer uitgebreid naar 3D, omdat dit niet veel toevoegt aan de diepgang van het onderzoek. In plaats daarvan zal extra tijd besteed worden aan het analyseren en controleren van de werking van het computermodel. Enkele manieren waarop deze controle uitgevoerd kan worden zijn:

- Controleren of de uitkomst van de optimizer voldoet aan alle constraints.
- De afmeting van een enkele staaf van het gevonden optimum veranderen. Vervolgens de kosten voor opnieuw bepalen en controleren of deze omhoog zijn gegaan.
- Onderzoeken of de toepassing van meerdere staafgroepen (waarbij de ligger bijvoorbeeld wordt opgesplitst in delen) leidt tot hogere of lagere kosten.

Er is afgesproken om een week na de tussenpeiling met Roland Abspoel de vorderingen te bespreken.