TI3800 Bachelorproject

Twin-Hub Network Project

Eindverslag

4008847 Marlou Pors
1170902 Jonathan Tetteroo
1308378 Carlo van der Valk

Technische Universiteit Delft
Faculteit Elektrotechniek, Wiskunde en Informatica

26 juni 2013

Voorwoord

Dit verslag is het resultaat van het ons Bachelor Eindproject voor de studie Technische Informatica aan de Technische Universiteit Delft. Het doel van dit project is een eigen applicatie te bouwen waarbij geleerde concepten en ontwikkelmethoden uit de vakken van de bachelor worden toepast. Tijdens het project is een volledig softwareontwikkelproces doorlopen.

De afgelopen 10 weken hebben we ons bezig gehouden met de ontwikkeling van een applicatie voor het Twin Hub Netwerk-Project en de TU Delft. De applicatie is gebouwd als 'schil' rondom een bestaand algoritme die het Twin Hub Bundling Probleem oplost en een Twin Hub Netwerk opzet. Met behulp van de applicatie kan de gebruiker een probleeminstantie invoeren en op een gebruiksvriendelijke manier alle details van de gevonden oplossing opvragen, terwijl het Twin Hub Netwerk in graafvorm is getekend.

We willen graag de volgende mensen bedanken voor hun begeleiding tijdens het project. Sebastiaan Meijer, die voor zijn afstudeeropdracht het algoritme heeft geschreven die het Twin Hub Bundling Probleem oplost en ons te hulp schoot als we problemen hadden. Michel Wilson, onze begeleider met wie we wekelijkse besprekingen hadden en ons feedback gaf op en meedacht over de applicatie. Tenslotte onze opdrachtgevers Ekki Kreutzberger en Cees Witteveen, die ons suggesties en ideeën gaven vanuit het Twin Hub Netwerk Project zelf.

Delft, juni 2013
Carlo van der Valk
Jonathan Tetteroo
Marlou Pors

Samenvatting

Dit Bacheloreindproject is uitgevoerd voor het Twin Hub Netwerk-Project, een samenwerkingsproject tussen zeehavens, intermodale spoorwegondernemingen en universiteiten, zoals de TU Delft. Dit project is opgezet om het goederentransport per spoor te stimuleren in Noordwest Europa. Zij doen dit door het goederentransport per trein te laten gaan over een netwerk dat gebruik maakt van een Hub (spoorknooppunt). Voor zijn Masterthesis heeft Sebastiaan Meijer onderzoek gedaan naar het Twin Hub Bundling Probleem, het berekenen van de totale kosten voor een opgegeven set locaties en containerstromen. Hij heeft een algoritme geschreven dat een Twin Hub Netwerk creeërt dat de minste kosten met zich meebrengt.

De applicatie die wij hebben gemaakt voor het eindproject is zo opgezet dat de gebruiker gemakkelijk een probleeminstantie kan creëren voor het algoritme van Sebastiaan en vervolgens alle details kan bekijken van het Twin Hub Netwerk dat het algoritme als oplossing geeft. Hiervoor hebben wij de applicatie opgedeeld in drie onderdelen. In het invoerscherm kan de gebruiker alle parameters voor de probleeminstantie instellen en binnen de applicatie een matrix opzetten die de containerstromen tussen de verschillende bron- en eindbestemmingen weergeeft (de zogenaamde Flow Matrix). Terwijl het algoritme een oplossing berekent krijgt de gebruiker in het processcherm meer informatie over de tot nu toe gevonden oplossingen. In het uitvoerscherm kan de gebruiker alle details opvragen van het gevonden Twin Hub Netwerk. Het netwerk wordt in de vorm van een graaf weergegeven, waar de gebruiker op de verschillende delen van de oplossing (zogenaamde batches) kan inzoomen. De gebruiker kan zien tussen welke locaties treinen of vrachtwagens rijden en hoeveel containers deze meebrengen voor welke eindbestemming.

Om de applicatie te bouwen hebben we een koppeling moeten bedenken om vanuit de code van Sebastiaan een eigen graafobject te kunnen creëren om deze te kunnen weergeven in het uitvoerscherm. Dit hebben we gedaan door de knopen van de graaf te koppelen aan de Terminals en elke kant van de graaf een bijbehorend Transport uit de bestaande code te geven. Daarnaast geven we elke knoop een set coördinaten mee om zo een realistische weergave te kunnen creëren. Door gebruik te maken van de JUNG bibliotheek waren we in staat de graaf te tekenen met de knopen op de meegegeven coördinaten. Ook om de invoer toegankelijk te maken voor de gebruiker hebben we eigen I/O klassen geschreven die om kunnen gaan met de Distance Matrix en Flow Matrix van Sebastiaan. De gehele applicatie is gebouw met behulp van het RCP (Rich Client Platform).

We hebben tijdens dit project een aantal onderdelen van de 'scrum' ontwikkelmethode gebruikt. De totale periode werd opgedeeld in vijf perioden waarin we drie ontwikkelmilestones hebben afgesproken. Op deze manier hebben we de totale periode van tien weken gestructureerd kunnen aanpakken. Wekelijks hadden we een vergadering met Michel Wilson om onze voortgang te bespreken. De laatste twee perioden waren we bezig met het finetunen van de applicatie. We hebben ook veel tijd gestoken in het schrijven van tests, zodat de applicatie later gemakkelijk kan worden uitgebreid zonder regressie. We hebben onze code opgestuurd naar de Software Improvement Group. Zij wezen ons erop dat onze code veel duplicatie kende. Hierop hebben we onze code flink kunnen verbeteren.

Ondanks dat we als eindproduct een complete applicatie hebben afgeleverd, zijn we van mening dat deze nog wel kan worden uitgebreid. We eindigen dit verslag dan ook met een aantal aanbevelingen voor de toekomstige ontwikkeling.

Inhoudsopgave

1 Inleiding **1**

2 Probleemstelling **2**
2.1 Het Twin Hub Netwerk-Project . . . 2
2.2 Het Twin Hub Bundling Probleem . . . 2
2.3 Applicatie . . . 3

3 Applicatie **4**
3.1 Overzicht van de applicatie . . . 4
3.2 Functionaliteiten van het invoerscherm . . . 4
3.3 Functionaliteiten van het processcherm . . . 5
3.4 Functionaliteiten van het uitvoerscherm . . . 6

4 Ontwerp **9**
4.1 Globaal Ontwerp . . . 9
4.1.1 Implementatie . . . 9
4.2 Ontwerp van de Graaf . . . 10
4.2.1 Nodes en Terminals . . . 10
4.2.2 Edges & Transports . . . 10
4.2.3 Graaf . . . 11
4.3 Weergave van de Graaf . . . 12
4.3.1 JUNG Bibliotheek . . . 12
4.3.2 Implementatie van graafweergave . . . 12
4.4 Input/Output . . . 15
4.4.1 Kosten- en Capaciteitenmodel . . . 15
4.4.2 Advanced Settings . . . 15
4.4.3 Flow Matrix . . . 15
4.4.4 Distance Matrix . . . 15
4.4.5 Koppeling Flow Matrix - Distance Matrix . . . 16
4.4.6 CoordinateList . . . 16
4.4.7 Validatie van de Inputs . . . 16
4.5 Tijdens de werking van de Gurobi Solver . . . 17
4.6 Opzet van de Applicatie / Interface . . . 18
4.7 Sequencediagrammen . . . 18
4.7.1 Graph Weergave . . . 19
4.7.2 Node Details Weergave . . . 20
4.7.3 Matrix Invoer . . . 20

5 Ontwikkelproces **22**
5.1 Projectproces . . . 22
5.1.1 Oriëntatie- en ontwerpfase . . . 22
5.1.2 Milestonemoment 1 . . . 22
5.1.3 Milestonemoment 2 . . . 23
5.1.4 Milestonemoment 3 . . . 23
5.1.5 Afronding . . . 23
5.2 Hulpmiddelen . . . 23
5.2.1 Java / RCP . . . 24
5.2.2 Checkstyle . . . 24

5.2.3 Gurobi . . . 24
5.2.4 Mantis . . . 24
5.2.5 Headless builds met Tycho/Maven . . . 25
5.2.6 Automatische builds met Jenkins Continuous Integration . . . 25
5.3 Testen . . . 25
5.3.1 JUnit . . . 25
5.3.2 Mockito . . . 25
5.3.3 Jenkins test automation en RCP . . . 25
5.3.4 Code coverage . . . 26
5.4 Versiebeheer . . . 26
5.5 Software Improvement Group . . . 26
5.5.1 Algemene Analyse . . . 26
5.5.2 Duplicatie . . . 27
5.5.3 Unit Size . . . 27
5.5.4 Unit Interfacing . . . 27
5.6 Integratie en acceptance . . . 28

6 Conclusie **29**

7 Aanbevelingen **30**
7.1 Kleine Uitbreidingen . . . 30
7.2 Fictief Netwerk . . . 31
7.3 Hub Detail Weergave . . . 31
7.4 Licenties . . . 32

Bibliografie **32**

Appendices **34**

A Opbouw van de Distance Matrix **35**

B Orientatie Document **36**

C Requirements Document **62**

1 Inleiding

Het is onze opdracht een applicatie af te leveren die de uitwerking van een bestaande Solver voor Twin Hub Bundling Probleem aantrekkelijk weergeeft en daarnaast gebruiksvriendelijk is voor personen die geen achtergrond hebben in deze technieken. Dit verslag dient als resultaat van het gehele eindproject.

Het Twin Hub Netwerk-Project heeft het doel inter-modaal goederentransport per spoor te stimuleren in Noordwest Europa. Zij proberen dit door een netwerk te creëren dat winstgevend is voor alle betrokken partijen. Hoe in dit netwerk de containerstromen ingedeeld moeten worden is het zogenaamde Twin Hub Bundling Probleem. Een Solver voor dit algoritme is al gemaakt, maar tot op heden is er geen interface waarin oplossingen duidelijk getoond kunnen worden aan investeerders om zo duidelijk te maken wat het project inhoudt en wat de voordelen hiervan zijn. Dit is onze opdracht.

In dit verslag zullen we na het uitleggen van de probleemstelling de applicatie presenteren. Hierbij geven we zowel een totaal overzicht als een uitleg van de functionaliteiten van de losse onderdelen, invoer, proces en uitvoer. Vervolgens zullen we uitleggen hoe we de applicatie hebben gebouwd. We presenteren het globale ontwerp ontworpen op basis van het Requirementsdocument en leggen uit hoe we de verschillende onderdelen hebben geïmplementeerd. Naast de applicatie zullen we ook ingaan op het ontwikkelproces en hoe we de afgelopen tien weken hebben ingedeeld. We hebben onze code naar de Software Improvement Group gestuurd en zullen de reactie die zij hebben gegeven ook verwerken in dit verslag. We zullen het verslag eindigen met een aantal aanbevelingen voor onze applicatie voor wanneer deze wordt uitgebreid of aangepast.

2 Probleemstelling

2.1 Het Twin Hub Netwerk-Project

Dit Bachelorproject wordt uitgevoerd voor het Twin Hub Netwerk-Project, een samenwerkingsproject tussen zeehavens, inter-modale spoorwegondernemingen en universiteiten (zoals de TU Delft) [11]. Het Twin Hub Netwerk-Project is opgezet om het goederentransport per spoor te stimuleren in Noordwest Europa. Dit omdat transport per trein duurzamer is dan per vrachtwagen en het totale transportnetwerk met gebruik van treinen ook robuuster wordt.

Het Twin Hub Netwerk-Project zet het goederentransport per trein op via zowel directe verbindingen als verbindingen die gebruiken van een spoorknooppunt, een zogenaamde 'Hub' (dit is Rotterdam of Antwerpen) om op deze manier per trein verschillende ladingen te combineren en de efficiëntie van het treintransport te verhogen. Echter, in sommige gevallen kan het nog wel zijn dat de inzet van vrachtwagens minder kosten oplevert.

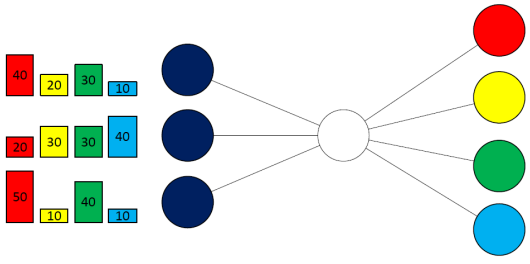
De ideale oplossing voor het indelen van de totale containerstromen (per dag) te berekenen is een probleem op zich, het zogenaamde Twin Hub Bundling Probleem. Voor zijn Masterthesis heeft Sebastiaan Meijer een algoritme geschreven dat, gegeven een kosten- en capaciteitenmodel, bekende afstanden en containerstromen tussen locaties, berekent hoe het transport ingedeeld dient te worden [13]. Hieronder zullen we kort bespreken wat het probleem precies inhoudt.

2.2 Het Twin Hub Bundling Probleem

Het Twin Hub Bundling Probleem (THB) is het berekenen van de totale kosten voor opgegeven begin- en eindbestemmingen met daarbij behorende containerstromen. Uit deze oplossing kan het bijbehorende Bundling Netwerk worden opgezet.

Om een geschikte oplossing voor het totale netwerk te vinden, kan het THB probleem worden opgedeeld in verschillende subproblemen. Allereerst kan worden aangenomen dat het transport over de hub Rotterdam en de hub Antwerpen niet van elkaar afhankelijk zijn, zodat probleeminstanties van deze hubs los van elkaar berekend kunnen worden. Daarnaast kan het probleem worden opgedeeld in verschillende groepen wat betreft afstand; Het is zo dat transport dat over een afstand van twee dagen gaat niet gecombineerd kan worden met transport dat over een afstand van drie dagen gaat. Een derde opdeling is dat er vervoer in één richting kan worden bekeken, omdat deze verschillende stromen ook onafhankelijk van elkaar zijn.

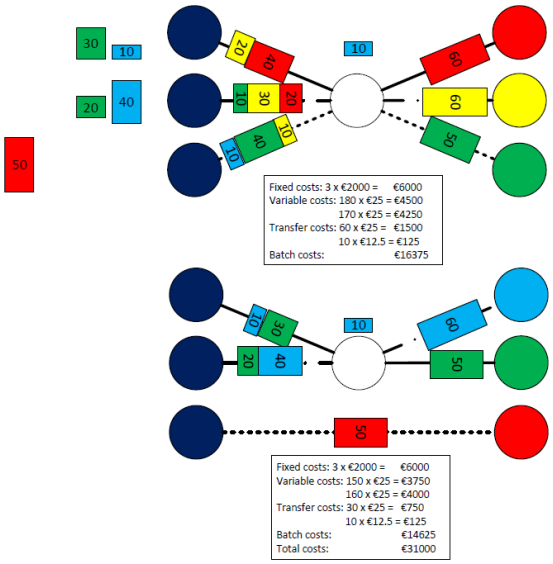
Een voorbeeldinstantie van het probleem is hieronder weergegeven in een graafvorm. Dit is in feite een subprobleem van het totale THB probleem, de hierboven genoemde afwegingen meegenomen:



Figuur 2.1: Een instantie van het THB Problem; Thesis S. Meijer [13]

Het THB Probleem is een NP-hard probleem (het bewijs hiervoor staat in de masterthesis van Sebastiaan [13]). Zoals eerder gezegd moet het algoritme naast de afstanden en containerstromen ook rekening houden met het bestaande kosten- en capaciteitenmodel van de treinen en vrachtwagen.

De oplossing die door het algoritme van Sebastiaan wordt berekend maakt gebruik van vijf verschillende vervoersopties: 'ToHubTrain', 'ToHubTruck', 'FromHubTrain', 'FromHubTruck' en een 'IndirectTrain' (de namen spreken voor zich). Daarnaast kan het transport van de containers opgedeeld worden in verschillende batches. Een batch is een opdeling van het transport dat over de Hub gaat en tegelijkertijd kan worden uitgevoerd zonder dat hier de capaciteit van de Hub wordt overschreden. Dit zijn bij verschillende vervoerstypen die samen een deel van de oplossing doorvoeren. De optimale oplossing is deze die de minste kosten met zich meebrengt.



Figuur 2.2: De optimale oplossing, verdeeld in 2 Batches; Thesis S. Meijer [13]

2.3 Applicatie

In de vorige paragraaf konden we zowel de probleeminstantie als de oplossing hiervan d.m.v. een graaf weergeven. Hoewel het algoritme van Sebastiaan slaagt in het berekenen van de optimale oplossing, creëert de applicatie die hier omheen gebouwd nog geen weergave van deze oplossing die voor de gebruiker eenvoudig te begrijpen is - deze weergave was d.m.v. tekst. Het is onze opdracht voor dit Bachelorproject een applicatie te bouwen die de uitkomsten van het algoritme duidelijker kan weergeven. Onze applicatie zal dan ook als 'schil' rondom het algoritme functioneren.

Er zijn verschillende eisen aan de applicatie. Omdat het een schil is rondom het bestaande algoritme moet er een goede koppeling tussen de applicatie en het algoritme zelf. Op deze manier kan er een applicatie gemaakt worden die het voor de gebruiker vergemakkelijkt om een probleeminstantie in te voeren, informatie te verkrijgen tijdens het oplossen van de probleeminstantie en het bekijken van de uitkomst van het algoritme.

Wij hebben een applicatie gemaakt die de oplossing net als in de vorige paragraaf weergeeft d.m.v. een graaf, waar alle gedetailleerde informatie in is op te vragen. Op deze manier kan de gebruiker veel inzicht in de oplossing verkrijgen en deze overzichtelijk aan geïnteresseerden laten zien. Hierdoor kan promotie van het Twin Hub Netwerk een boost krijgen. Daarnaast kan de applicatie in een later stadium gebruikt worden als planningstool wanneer het Twin Hub Netwerk gebruikt wordt om het transport te plannen.

3 Applicatie

In dit hoofdstuk bespreken we de uiteindelijke werking van de applicatie. Allereerst geven we een beschrijving van de totale applicatie, daarna zullen we dieper ingaan op de verschillende schermen: Het invoer-, proces- en uitvoerscherm.

3.1 Overzicht van de applicatie

We hebben een applicatie gemaakt die de gebruiker de mogelijkheid geeft de Solver van Sebastiaan Meijer te gebruiken. Dit betekent dat de gebruiker een probleeminstantie moet kunnen creëren, meer informatie krijgt wanneer de Solver aan het werk is en de uiteindelijke uitkomst ook gemakkelijk kan ontcijferen. Om deze drie verschillende onderdelen duidelijk te scheiden hebben we de applicatie drie losstaande onderdelen gegeven.

De applicatie opent in het invoerscherm, waar de gebruiker een probleeminstantie kan opzetten. Dit houdt een paar verschillende onderdelen in. De afstandrelaties tussen alle mogelijke locaties en hun coördinaten liggen al vast, maar de gebruiker moet wel een 'Flow Matrix' opzetten die weergeeft hoeveel containers van de bron- naar de eindbestemmingen vervoerd moeten worden. Daarnaast kan het kosten- en capaciteitenmodel worden aangepast. Voordat de gebruiker door kan naar het procesgedeelte moet de invoer gevalideerd worden; dit gebeurt automatisch.

In het processcherm kan de gebruiker meer informatie verkrijgen over wat de Solver nu werkelijk berekent. Door middel van tekst en een grafiek wordt onder andere duidelijk gemaakt hoe goed de tot nu toe gevonden oplossingen zijn. De gebruiker kan op dit moment er ook voor kiezen de Solver stop te zetten.

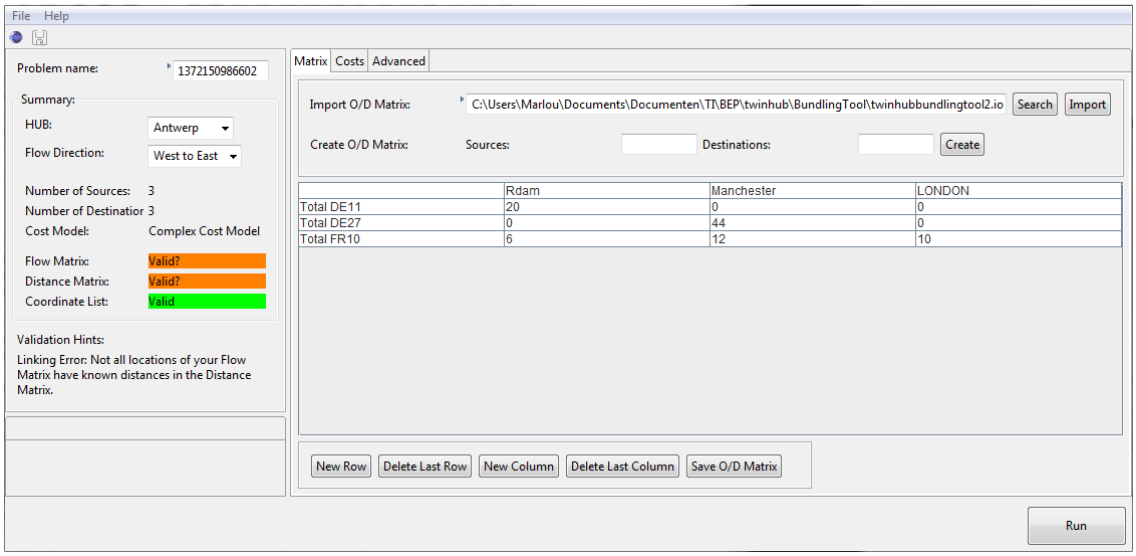
In het derde scherm kan de gebruiker alle details van de laatst gevonden / uiteindelijke oplossing bekijken. Veel onderdelen zijn meegenomen uit de applicatie van Sebastiaan, nieuw is dat het netwerk wordt weergegeven als een graaf. In deze weergave is het ook mogelijk om in te zoomen op verschillende batches en hier details van op te vragen, zoals het aantal containers dat vertrekt van een bepaalde locatie en met welke eindbestemmingen deze containers hebben.

3.2 Functionaliteiten van het invoerscherm

Voordat het algoritme een oplossing kan berekenen moet er door de gebruiker een correcte probleeminstantie worden opgezet. Deze probleeminstantie bevat een distance matrix, Flow Matrix, capaciteitenmodel en kostenmodel. Het opzetten van de laatste twee onderdelen gaat in onze applicatie zoals dat in de applicatie van Sebastiaan ook ging. Onze voornaamste uitbreiding is op het gebied van het opzetten van de Flow Matrix - de Distance Matrix staat vast.
Het invoerscherm is te zien op de volgende pagina. Door middel van tabbladen kunnen de verschillende onderdelen van de probleeminstantie aangepast kunnen worden. Aan de linkerkant van het scherm kan de probleemnaam worden ingevoerd en is er een samenvatting gegeven van ingevoerde onderdelen.

In het rechtergedeelte van het scherm kan de Flow Matrix worden opgemaakt. Dit kan door een geheel nieuwe matrix te creëren, waarbij het aantal begin- en eindbestemmingen moet worden opgegeven, of een bestaande Flow Matrix te importeren. Het is nodig deze matrix in te laden in het scherm (d.m.v. de Importknop). Vervolgens is het mogelijk de matrix nog aan te passen, waarbij er gebruik gemaakt kan worden van de knoppen onder de weergegeven matrix. Er is ook een mogelijkheid de aangepaste matrix weer op te slaan d.m.v. de 'Save O/D Matrix' knop.

De door de gebruiker gecreëerde / geïmporteerde Flow Matrix wordt door de applicatie automatisch gevalideerd. Hierbij wordt in het rechtergedeelte van het scherm met behulp van kleur



Figuur 3.1: Het invoeren van de Flow Matrix van de probleeminstantie

aangegeven of de verschillende matrices valide zijn en of de koppeling tussen deze matrices klopt. Algemene informatie over de I/O klassen is in hoofdstuk 4 te vinden, waarbij er in paragraaf 4.4.7 wordt uitgelegd hoe de validatie in zijn werk gaat. Naast de gekleurde blokken wordt daaronder breder uitgelegd wat het probleem kan zijn. De tekst in de afbeelding geeft aan dat een aantal locaties in de Flow Matrix niet gedeclareerd zijn in de Distance Matrix. De gebruiker kan dit probleem verhelpen door of de Flow Matrix aan te passen (binnen de applicatie) of de Distance Matrix aan te passen / controleren (buiten de applicatie).

Naast dat het linkergedeelte een samenvatting en informatie over de validatie geeft, kan hier ingesteld worden welke stad de hub is en of het goederentransport van west naar oost loopt of vice versa. In dit stadium van het Twin Hub Netwerk-Project wordt alleen nog maar gebruik gemaakt van de hub Antwerpen. Wij hebben deze knop toegevoegd om de applicatie ook bruikbaar te maken in de toekomst (meer informatie over deze uitbreiding is te vinden bij het hoofdstuk Aanbevelingen).

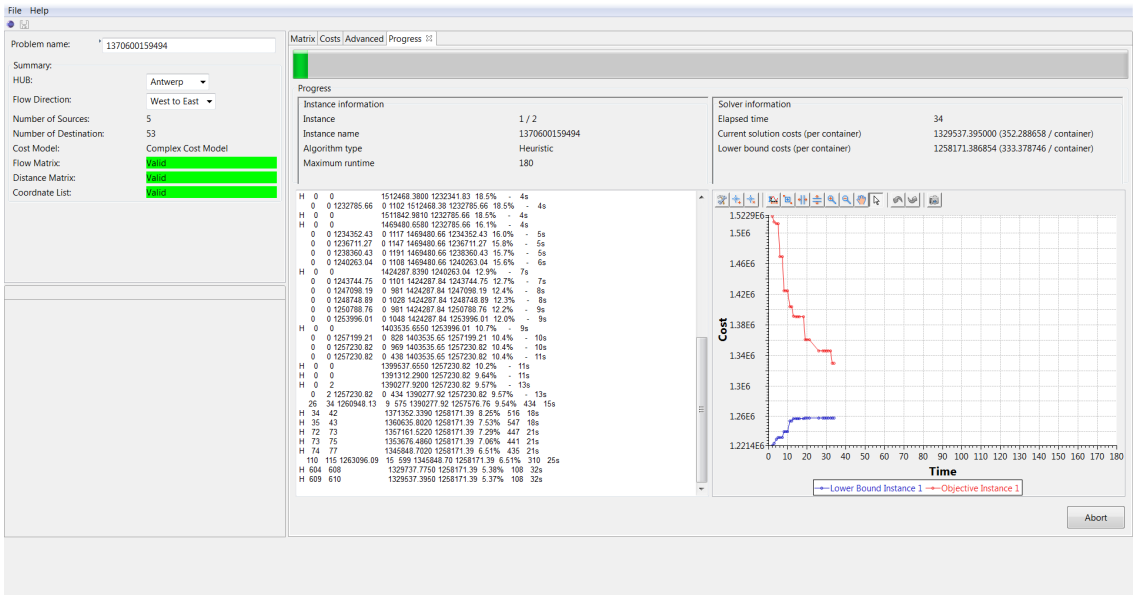
Indien alle drie de validatieblokken groen gekleurd zijn, kan de gebruiker op Run drukken.

3.3 Functionaliteiten van het processcherm

Wanneer alle data succesvol is ingevoerd en op de Run knop is gedrukt, zal de applicatie een probleeminstantie genereren en doorvoeren aan de Gurobi Solver. Vervolgens verschijnt het processcherm dat tijdens de werking van de solver de voortgang weergeeft aan de hand van een progressbar, samenvatting, tekst printout en een grafiek.
Bovenaan staat de progressbar, deze geeft aan hoe ver de Solver is ten opzichte van de maximale looptijd. Deze looptijd is standaard 1800 seconden (30 minuten) en is aan te passen in de Advanced tab van het invoergedeelte.

Onder de progressbar bevindt zich een samenvatting van de instantie waar de Solver nu aan bezig is en de laatste gevonden resultaten die uit de Solver zijn gekomen. Alle output van de Solver wordt geprint in de tekstruimte links. Op deze manier kunnen gebruikers inzicht krijgen over alles wat de Solver meldt.

Rechts wordt er een grafiek geplot met de kosten van de gevonden resultaten ten opzichte van de tijd. De rode lijn stelt de kosten van de gevonden oplossing voor, de blauwe lijn zijn de kosten van de gevonden ondergrens. De gebruiker kan door te werken met de toolbar die boven de grafiek te vinden is, de grafiek beter bekijken, notities maken en de opslaan als een PNG-bestand.



Figuur 3.2: Weergave tijdens de werking van de Solver.

3.4 Functionaliteiten van het uitvoerscherm

Wanneer de gebruiker de Solver heeft stopgezet of wanneer de Solver zijn maximale looptijd bereikt heeft, opent de applicatie automatisch de laatst berekende oplossing. Allereerst opent de applicatie hetzelfde uitvoerscherm als de applicatie van Sebastiaan Meijer, waar een overzicht gegeven wordt van de karakteristieken van de oplossing en de gebruiker de mogelijkheid heeft om de 'boom' van de oplossing uit te klappen. Ook kan de gebruiker door met de rechtermuisknop op een bepaalde trein/truck te klikken de details hiervan opvragen. Deze worden geopend in een nieuw tabblad.

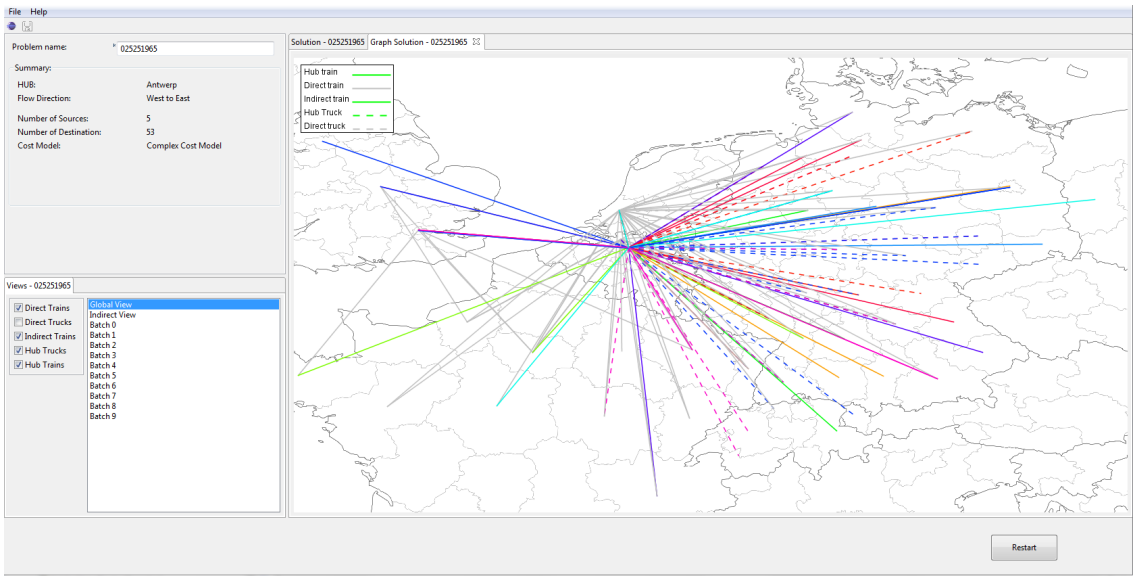
Wij hebben dit basisscherm uitgebreid met twee knoppen: 'View Solution Graph' en 'View Solution Parameters'. Beide knoppen openen een nieuw tabblad. Wanneer er op de tweede knop wordt gedrukt kan de gebruiker nalezen welke parameters hij de probleeminstantie heeft gegeven. Wanneer er op de eerste knop wordt gedrukt wordt de Netwerk View geopend.

Het is mogelijk om op elk moment een solutionfile in te laden. Automatisch wordt dan het uitvoerscherm geopend. Indien er al een solution is geopend, worden er nieuwe tabbladen toegevoegd. Op deze manier is het mogelijk om meerdere oplossingen open te hebben in de applicatie. Indien de gebruiker een nieuwe probleeminstantie wil invoeren wordt op de Restartknop rechtsonderin geklikt.

Het gehele netwerk met focus op de Batches

De applicatie kan verschillende soorten netwerkweergaven geven. De allereerste weergave die de applicatie opent is die van de totaal oplossing met de focus op de verschillende batches. Een voorbeeld is gegeven in figuur 3.3. Dit is een algemene weergave: Alle transportlijnen die worden gebruikt in de oplossing worden weergegeven, zonder duidelijke weergave van de verschillende Nodes. Zo krijgt de gebruiker snel een overzicht van het totale netwerk. De doorgetrokken lijnen zijn de verschillende soorten treinen, stippellijnen zijn vrachtwagens. Alle lijnen die bij een dezelfde batch horen krijgen eenzelfde kleur. Alle directe verbindingen zijn grijs gekleurd.

Met het menu linksonder kan de gebruiker zich door de verschillende views werken. Naast dit keuzemenu kan met de checkboxes gekozen worden welke verbindingen er getoond worden. Zo zijn in deze totale weergave de directe vrachtwagens uitgeschakeld.



Figuur 3.3: Het Uitvoerscherm met een totale netwerkweergave geopend

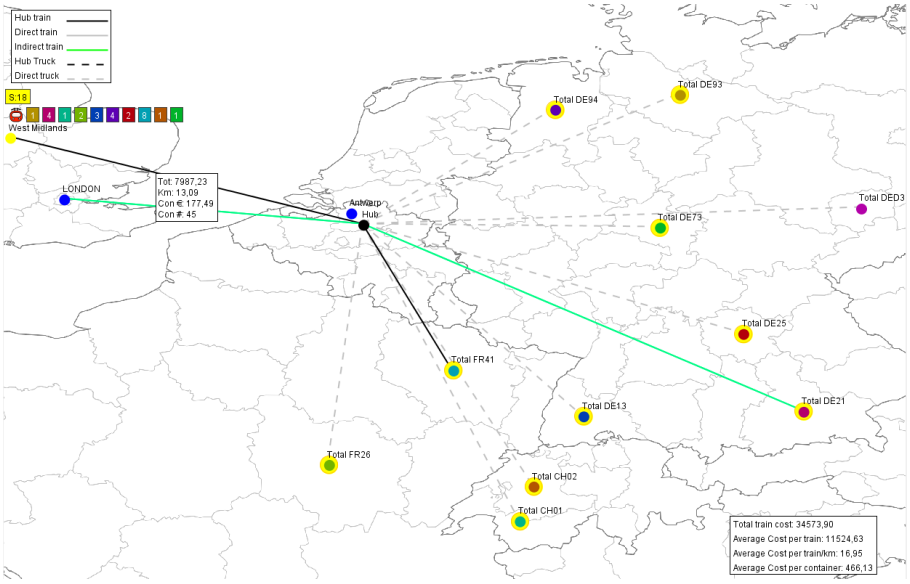
Het gehele netwerk met focus op de Indirecte Treinen

Een tweede totaalweergave die de applicatie heeft is de weergave van het totale netwerk met de focus op indirecte treinen. Indirecte treinen zijn het hart van het TwinHub Netwerk Project en deze worden verduidelijkt door ze een eigen kleur te geven. De vorm van de lijnen zijn hetzelfde als de andere weergaven. Treinen zijn doorgetrokken strepen en vrachtwagens hebben stippellijnen. Verschil is dat alle transporten behorende tot een batch zwart gekleurd zijn en elke indirecte trein een eigen kleur krijgt, zodat elke indirecte trein onderling van elkaar te scheiden is en er zo in één oogopslag te zien is hoeveel indirecte treinen de oplossing bevat.

Figuur 3.4: Totale netwerkweergave met de focus op Indirecte Treinen

Batch detailweergave

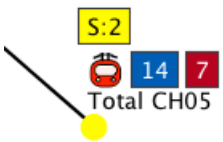
Naast de twee totale overzichten kan de gebruiker langs alle batches lopen. Hieronder is de weergave van Batch 3 uit het hiervoor getoonde totaal netwerk te zien:



Figuur 3.5: Batchweergave met details van West Midlands

De verbindingen hebben dezelfde vorm als in de totale weergave. Een batch bevat transporten die allemaal langs de Hub moeten en bevat dus geen directe verbindingen. Daarom wordt er met andere kleuren gewerkt. De vrachtwagenverbindingen (zowel van als naar de Hub) zijn grijs en alle treinverbindingen zijn zwart. De indirecte treinverbindingen hebben elk een individuele kleur.

In de batchweergave is het mogelijk in te zoomen op de locaties binnen de weergave zelf. Wanneer er op een Node wordt geklikt waar de goederen vandaan komen (hier: West Midlands) wordt getoond hoeveel containers er vervoerd worden en waar deze naar toe gaan. De Nodes van de bestemmingen hebben verschillende kleuren, zodat direct duidelijk wordt hoeveel containers naar welke precieze locatie gebracht moeten worden. Het treinlogo geeft aan dat deze containers op een trein vervoerd worden. Het gele blokje met S:18 geeft aan dat er ook 18 containers meereizen naar de hub die daar op de stack worden gelegd. Deze worden een andere batch opgehaald. Wanneer er een bestemmingsnode bekeken wordt, wordt op eenzelfde manier aangegeven waar de containers vandaan komen.



Figuur 3.6: Node details

Hiernaast is een uitvergrote Node als voorbeeld gegeven. De locatie, CH05, krijgt in totaal 23 containers via een treinverbinding binnen. Hiervan zijn 2 containers afkomstig van de stack en $14+7$ containers van twee verschillende bronlocaties.

In de rechter onderhoek van de weergave is een klein overzicht gegeven van de Batchkosten. Daarnaast heeft elke treinverbinding een kostenblok dat verschijnt als er op de verbonden Node geklikt wordt. Hierin staan de totale kosten van de verbinding, het aantal kilometers, het aantal containers en de gemiddelde kosten per container. Losse kostenblokken van vrachtwagens zijn niet gegeven, maar zijn eventueel wel via de details van een batch in de weergave van Sebastiaan nog op te vragen.

Door shift ingedrukt te houden is het mogelijk om meerdere nodes tegelijkertijd te openen. Het is niet mogelijk een oorsprong- en bestemmingsnode tegelijkertijd te bekijken. Met het scrollwheel van de muis kan worden in- en uitgezoomd op de kaart. Wanneer er niet op een Node geklikt wordt is het mogelijk om de kaart te verslepen.

4 Ontwerp

In dit hoofdstuk zullen we dieper ingaan op het ontwerp en implementatie van onze applicatie. Allereerst zullen we een kort overzicht geven van ons globaal ontwerp. Daarna zullen we uitleggen hoe we de implementatie hebben aangepakt en beschrijven we verschillende klassen. In de uitleg maken we ook duidelijk hoe we onze applicatie hebben gekoppeld aan de bestaande Solver van Sebastiaan Meijer.

4.1 Globaal Ontwerp

Het globale ontwerp dat we de opdrachtgever hebben gepresenteerd is te vinden in het Requirementsdocument. Onze uiteindelijk applicatie is hier enigszins van afgeweken, maar volgt wel dezelfde grote lijnen.

Onderdelen als de kosteninvoer en -weergave hebben wij direct overgenomen uit de applicatie van Sebastiaan Meijer. Wij hebben zijn applicatie uitgebreid door zowel aan de invoer- als de uitvoerkant verschillende tabbladen toe te voegen zodat meer onderdelen aangepast of bekeken kunnen worden. Deze implementatie maakt intensief gebruikt van de RCP onderdelen waarmee een GUI gebouwd kan worden.

Naast de drie schermen die de gebruiker zal zien was het nodig een Graaf-datastructuur op te zetten vanuit de oplossing die de Solver van Sebastiaan geeft. Dit is een eigen geïmplementeerde datastructuur geworden die we besproken wordt in 4.2. Tussen deze datastructuur en de GUI zit nog een aantal klassen die zorgen voor de visualisatie hiervan. Deze wordt besproken in 4.3.

Vooral het invoergedeelte heeft te maken met verschillende I/O-klassen, omdat in ons ontwerp naar voren was gekomen dat de gebruiker zelf een Flow Matrix moet kunnen maken en de Distance Matrix vastgelegd is binnen de tool, evenals de coördinatenlijst. Alle I/O-klassen worden besproken in 4.4.

4.1.1 Implementatie

Voor de implementatie van het ontwerp zijn we uitgegaan van de bestaande applicatiestructuur. Deze is opgedeeld in verschillende losse Eclipse projecten (plug-ins) die samen de gehele applicatie vormen. Dit is de gangbare manier van ontwikkelen met het RCP platform. Zelf hebben wij hier onze eigen onderdelen aan toegevoegd en zo een nieuwe applicatie opgebouwd. Daarnaast hebben we projecten toegevoegd die tests bevatten voor de code (test plug-ins). Hiermee kan elke plug-in afzonderlijk getest worden. Tabel 4.1 is een overzicht van de projecten die wij hebben toegevoegd.

Project	**Omschrijving**
twinhubbundlingtool2	Het hoofdproject, bevat de grafische schil.
twinhubbundlingtool2.tests	Unit tests voor de grafische schil.
twinhubbundlingtool2.graph	De implementatie van de graaf.
twinhubbundlingtool2.graph.tests	Unit tests voor de graaf implementatie.
twinhubbundlingtool2.graph.ui	De grafische weergave van de graaf.
twinhubbundlingtool2.graph.ui.tests	Unit tests voor de graaf weergave.
twinhubbundlingtool2.io	Input/output klassen, o.a. de flowmatrix.
twinhubbundlingtool2.io.tests	Unit tests voor input/output.

Tabel 4.1: Opdeling van de applicatie in verschillende projecten.

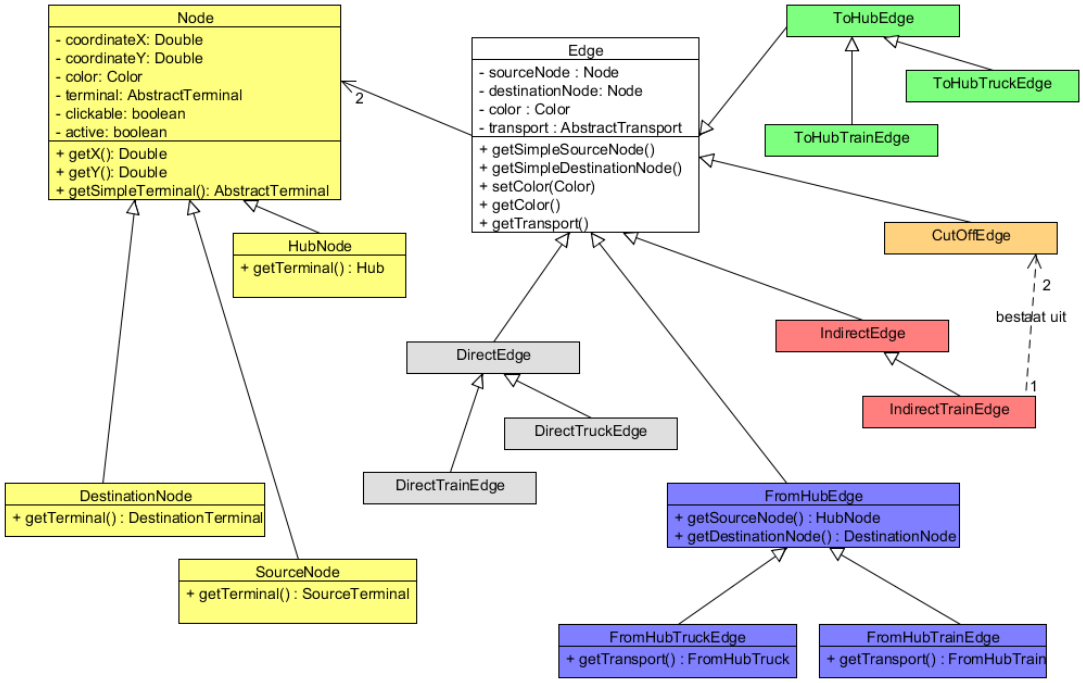
4.2 Ontwerp van de Graaf

Opzet van onze applicatie was om de uitkomst van de Solver, dat wil zeggen de verdeling van het transport tussen verschillende locaties gedaan door verschillende vervoerstypen, weer te geven in een graaf. Hieronder zullen we uitleggen hoe we de koppeling tussen de bestaande code en onze opgezette graaf hebben aangepakt.

4.2.1 Nodes en Terminals

De standaard datastructuur van een graaf bestaat uit Nodes en Edges (knopen en kanten). Echter, de informatie die de oplossing met zich meebrengt zijn gedefinieerd als 'Terminals' en verschillende soorten 'Transports'. De koppeling tussen Terminals en Nodes was gemakkelijk realiseerbaar. In onze applicatie worden er Nodes opgezet waarvan gezegd kan worden dat hierop een Terminal 'ligt'. Omdat er verschillende soorten Terminals zijn, zijn er ook verschillende soorten Nodes, waarna via de methode 'getTerminal()' de bijbehorende Terminal kan worden opgevraagd. Hierdoor kom je direct in de informatieverschaffing van de oplossing. Een compact overzicht van de relaties tussen de klassen is in het klassendiagram van Nodes en Edges te zien (figuur 4.1).

Doordat een Terminal zich op een Node bevindt, konden we de Nodes gemakkelijk ook een set coördinaten meegeven, waardoor we later precies weten waar deze Node getekend moet worden. Daarnaast hebben we de Nodes verschillende andere eigenschappen meegegeven die we later konden gebruiken in onze applicatie zonder dat we hiermee aan de Terminal hoeven te sleutelen.



Figuur 4.1: Klassendiagram rondom Nodes en Edges

4.2.2 Edges & Transports

De Edges voor een graaf hebben we op soortgelijke wijze aangepakt: Een Edge, met aan beide uiteinden een Node, heeft een transport dat hier overheen loopt. In de hoofdklasse Edge stelt dit nog een AbstractTransport voor. Om uiteindelijk bij de correcte soorten Nodes en Transports uit

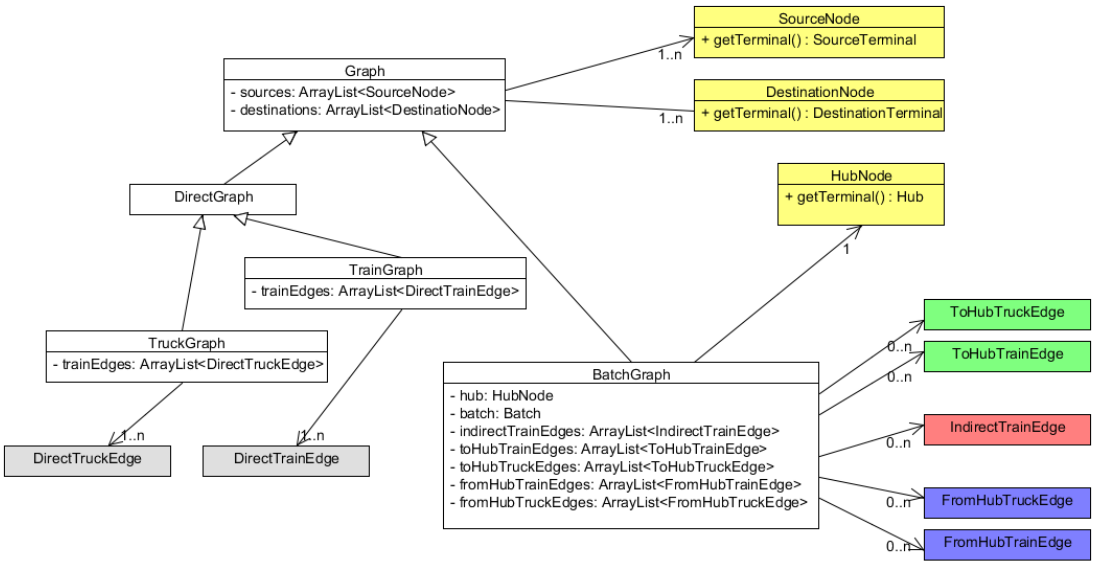
te komen hebben we de verschillende Edges hiërarchisch uitgebouwd. Het resultaat is te vinden in het vereenvoudigde klassendiagram hiernaast.

Bij de FromHubEdge klasse staan de methoden weergegeven waardoor er gemakkelijk bij de originele oplossingen gekomen kan worden. Van een FromHubEdge is het bekend welke soort Node de bron- (de Hub) en eindbestemming is, vandaar dat de methoden zich daar bevinden. Het correcte soort Transport is een laag later te vinden, er bestaat namelijk een FromHubTruck en een FromHubTrain. Wanneer je via de Edge bij het Transport bent gekomen kan je gemakkelijk de rest van de details opvragen, bijvoorbeeld hoeveel containers er zijn vervoerd en wat de kosten hiervan zijn.

Een klasse die enigzins buiten deze logica valt is CutOffEdge. De achterliggende gedachte is dat een IndirectTrainEdge kan worden opgedeeld in twee stukken: Het stuk naar de Hub en het stuk van de Hub naar de eindbestemming. Hier wordt CutOffEdge gebruikt. Deze klasse wordt slechts gebruikt voor de weergave van de graaf. De rest van de informatie wordt opgevraagd via de originele set IndirectTrainEdges.

4.2.3 Graaf

Omdat een Solution bestaat uit een verzameling directe treinen, directe vrachtwagens en verschillende Batches waarbij de Hub een rol speelt, hebben wij ervoor gekozen ook verschillende soorten Graphs te maken: Een TruckGraph, TrainGraph en BatchGraph. Alle soorten Graphs worden gemaakt m.b.v.de klasse 'GraphParser'. De hiërarchie en koppeling met Nodes en Edges is hier te vinden in het (vereenvoudigde) klassendiagram.



Figuur 4.2: Datastructuur Graph en bijbehorende Nodes en Edges

De TruckGraph en TrainGraph zijn gemakkelijk te verkrijgen, zij bestaan slechts uit de verzameling bron- en eindbestemmingen en de verzameling van respectievelijk de DirectTruckEdges en DirectTrainEdges. Via deze verzameling is bij de details van de oplossing te komen.

Een BatchGraph is ingewikkelder opgebouwd. Allereerst maakt een Batch gebruik van de Hub. In de oplossing kan de Batch bestaan uit een verzameling van vijf verschillende vervoerstypen, waardoor er ook vijf verschillende verzamelingen Edges zijn. Via deze verzamelingen is bij de details van de transporten te komen. Daarnaast bevat elke BatchGraph zijn originele Batch die hij representeert. Via deze kan nog meer informatie worden opgevraagd.

4.3 Weergave van de Graaf

Voor het weergeven van de graaf hebben we diverse mogelijkheden onderzocht. Belangrijk was dat knooppunten op een geografisch logische manier weergegeven konden worden, al dan niet met een kaart. Hiervoor hebben we diverse bibliotheken bekeken die een graaf kunnen tekenen en die geschikt zijn voor integratie in een Java applicatie. Belangrijke aspecten bij het maken van een keuze waren o.a. beschikbare documentatie, uitbreidbaarheid en de licentie. Dit met het oog op verdere ontwikkeling van de tool en gebruik in een commerciële context. Zie tabel 4.2 voor een vergelijking van de onderzochte bibliotheken.

4.3.1 JUNG Bibliotheek

We hebben uiteindelijk gekozen voor de JUNG bibliotheek. De uitbreidbaarheid en interactie mogelijkheden maakten het makkelijk deze bibliotheek aan onze wensen aan te passen en te integreren in de applicatie. Daarnaast is de BSD licentie gunstig vanuit commercieel oogpunt. Code van JUNG die aangepast wordt hoeft namelijk niet vrijgegeven te worden.

Omdat de documentatie beperkt is, was het wel noodzakelijk de code goed te begrijpen. JUNG is echter zeer modulair opgebouwd en het was relatief eenvoudig zelf functionaliteit toe te voegen of te veranderen. Functionaliteiten die wij toegevoegd hebben zijn het tekenen van een kaart onder de graaf en het nauwkeurig plotten van Nodes op de kaart op basis van geografische coördinaten. Daarnaast tekenen we het aantal containers dat vervoerd wordt per Node per transport en geven we de kosten weer. De Edges worden op basis van transportsoort op een verschillende manier getekend.

Bibliotheek	**Voordelen**	**Nadelen**
GeoTools [4]	Eenvoudig renderen van kaarten met Nodes, interactief.	Slechte documentatie, gecompliceerde structuur, traag.
Gephi [5]	Uitgebreide features, mooie graphics, animaties.	Slechte documentatie, lastig uit te breiden, OpenGL noodzakelijk.
Graphviz [6]	Uitgebreide features, goede documentatie.	Niet interactief.
JGraphT [12]	Simpel te implementeren, goede documentatie.	Te veel dependencies, te weinig features.
JGraphX [8]	Uitgebreide features.	Meer gericht op web applicaties dan op Java applicaties.
JUNG [9]	Interactief, makkelijk uit te breiden, BSD licentie.	Slechte documentatie.
Prefuse [10]	Interactief, animaties, goede documentatie, BSD licentie.	Niet actief in ontwikkeling.

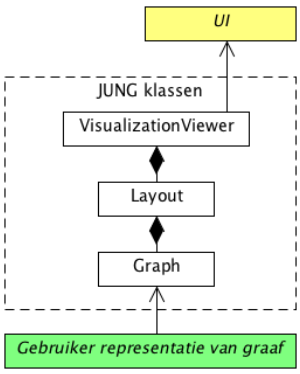
Tabel 4.2: Vergelijking tussen diverse graaf bibliotheken.

4.3.2 Implementatie van graafweergave

Voor de integratie van JUNG in onze applicatie hebben we een aparte Eclipse plug-in geschreven met de klassen die de graaf zullen weergeven.

Een JUNG graaf wordt getekend door een VisualizationViewer object. Dit object is verantwoordelijk voor het tekenen van de graaf op het scherm door middel van verschillende renderers en de interactie met de graaf door middel van event handlers. Het positioneren van de Nodes kan met behulp van een Layout Object. Dit object bepaalt waar de Nodes getekend worden, in ons geval op een vast punt gebaseerd op coördinaten. Het Layout Object bevat de eigenlijke graaf die

we willen tekenen, dit graaf object is afkomstig uit onze eigen interne graaf representatie, zie ook figuur 4.3.



Figuur 4.3: JUNG structuur

Aanpassingen

Om aan onze eisen te voldoen moesten er verschillende wijzigingen doorgevoerd. Zo worden de Edges op basis van type transport dat ze representeren getekend. De Nodes moeten meer informatie geven dan nu het geval is. Om dit mogelijk te maken zijn nieuwe renderer klassen geschreven voor de Nodes en Edges. Deze zijn gebaseerd op de standaard JUNG implementatie en verder uitgebreid. Zie figuur 4.5 voor een klassendiagram.

Voor het tekenen van de Edge kosten is een aparte Edge Label Renderer gemaakt die de bestaande renderer vervangt. Hiermee worden de kosten van treinverbindingen weergegeven.

Voor het tekenen van de kaart, legenda en totale kosten moeten zogenaamde Paintable's gemaakt worden en toegevoegd worden aan de VisualizationViewer. Deze worden aangemaakt in de VisualHelpers klasse.

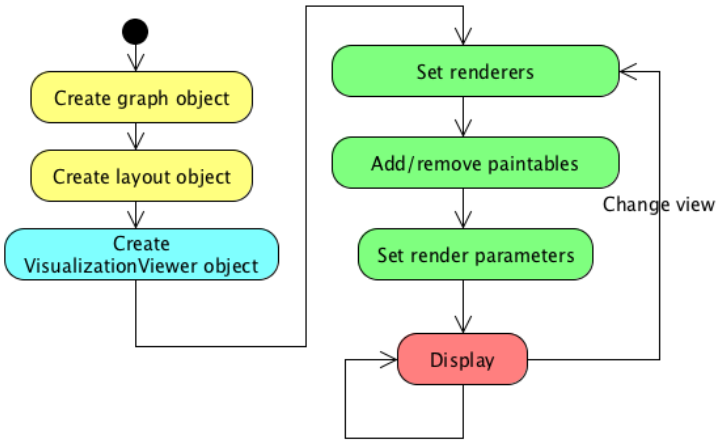
Het aanleveren van onze weergave aan de GUI gebeurt via het SolutionVisualizer object. Dit object bevat diverse methoden om verschillende graaf weergaven te genereren en af te wisselen. Hieruit wordt ook de VisualizationViewer opgevraagd zodat deze getekend kan worden.

In figuur 4.4 staat de uitvoering van de SolutionVisualizer beschreven. In het linker gedeelte wordt vanuit de graaf een layout gecreeërd en een VisualizationViewer object opgebouwd. Vervolgens worden alle Node en Edge renderers geinitialiseerd en de Paintable klassen toegevoegd (legenda, kaart en kosten view). Nu kan de graaf worden getekend door de interface. Wanneer de view wordt aangepast, bijvoorbeeld van het wisselen van een globale view naar een specifieke batch view, hoeven alleen de stappen in groen opnieuw doorlopen te worden omdat er aan de objecten daarvoor niets veranderd. Dit scheelt in de performance bij het wisselen van views bij grote netwerken.

Kaart achtergrond

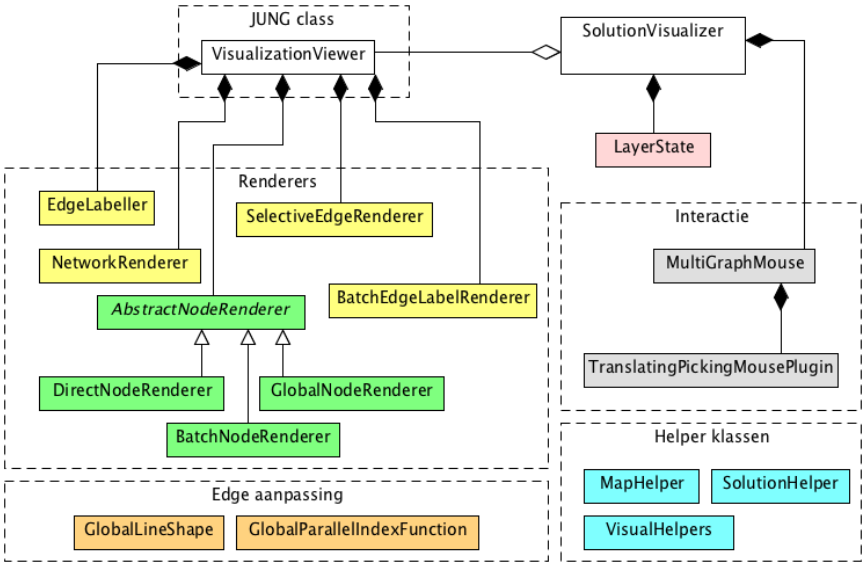
Voor het tekenen van de kaart hebben we de geografische coördinaten van de Nodes nodig, deze worden ingeladen als de graaf geladen wordt. Daarnaast hebben we een referentiepunt nodig op de kaart zodat we onze Nodes op de juiste positie kunnen tekenen.

De kaart is geëxporteerd vanuit een geografisch informatiesysteem (GIS), waarbij de kaart zelf in PNG formaat is opgeslagen. Daarnaast is er een zogenaamde ESRI World File gegenereerd die de coördinaten en grootte van de kaart bevat. Met de gegevens in deze file kunnen we de transformatie van geografische longitude/latitude coördinaten naar pixel coördinaten op het scherm maken.



Figuur 4.4: Executie flow van SolutionVisualizer

In de MapHelper class voeren we deze transformatie automatisch uit. Hiermee is het eenvoudig in een later stadium de kaart te vervangen. Het enige wat aangepast hoeft te worden zijn de kaart en worldfile. Daarnaast moeten de coördinaten van de Nodes natuurlijk wel op de kaart vallen.



Figuur 4.5: Klassendiagram van de graaf weergave

Kleur Algoritme

Voor het genereren van kleuren voor de batches en Nodes maken we gebruik van een algoritme dat een kleur selecteert op basis van de Gulden Snede [7]. Hierdoor is er voldoende contrast tussen de kleuren van nodes en edges zodat gemakkelijk onderscheid gemaakt kan worden.

4.4 Input/Output

Voordat de Solver een oplossing kan berekenen moeten de juiste onderdelen worden ingevoerd. Hier zullen we bespreken wat we hebben veranderd aan het I/O gedeelte van de applicatie van Sebastiaan en hoe we zijn bestaande onderdelen in de applicatie hebben geïntegreerd. Een overzicht van alle I/O klassen is aan het einde van deze paragraaf gegeven.

4.4.1 Kosten- en Capaciteitenmodel

Het invoeren van alle kosten gaat nog hetzelfde als in de oude applicatie. We hebben hiervoor een eigen tabblad gemaakt, die dezelfde interface heeft als de oorspronkelijke applicatie.

4.4.2 Advanced Settings

Het veranderen van de Advanced Settings heeft ook een eigen tabblad gekregen in onze applicatie. Hier kun je bepaalde vervoersopties aan/uit zetten en een bekende MIP startfile kiezen.

4.4.3 Flow Matrix

De grootste feature van het invoeren van de input in onze applicatie is het specificeren van de Flow Matrix (O/D Matrix). Het is mogelijk een bestaande Flow Matrix te importeren en deze alsnog aan te passen in de applicatie, maar je kan ook een vanuit de applicatie een compleet nieuwe Flow Matrix maken. De applicatie heeft knoppen om rijen en kolommen toe te voegen en te verwijderen. Daarnaast kan de gebruiker zijn gecreëerde Flow Matrix als een CSV-betand opslaan.

De Flow Matrix kent ten opzichte van de oude applicatie nieuwe eisen waar hij aan moet voldoen. In de applicatie van Sebastiaan was de Flow Matrix (slechts) een rechthoekige matrix met alleen getallen, waarvan de bijbehorende bron- en eindbestemmingen uit de Distance Matrix gehaald moesten worden. Dit maakt het voor de gebruiker onoverzichtelijk om deze aan te passen of fouten op te sporen. De nieuwe Flow Matrix heeft in de eerste kolom zijn bronbestemmingen en in de eerste rij de eindbestemmingen gespecificeerd, waarbij logischerwijs cell (0,0) leeg blijft. Deze opzet is noodzakelijk wanneer de Flow Matrix geïmporteerd wordt en blijft behouden als de Flow Matrix wordt gestart / aangepast in de applicatie.

4.4.4 Distance Matrix

De opbouw van de Distance Matrix, met verschillende 'blokken' die de verschillende soorten afstanden voorstellen, is in de nieuwe applicatie hetzelfde gebleven. Echter, het bestand dat de gebruiker kan aanpassen is zo opgezet dat het duidelijk is welk getal welke soort afstand tussen welke twee locaties voorstelt. De klasse 'CompactDistance MatrixCSV' heeft de methode 'getCompactDistance Matrix' die van deze uitgebreide Distance Matrix eenzelfde compacte versie maakt die uiteindelijk ingelezen kan worden om vervolgens door het algoritme gebruikt te worden. De opzet van de Distance Matrix is gegeven in appendix A.

Het idee is dat de Distance Matrix alle afstanden tussen de bestaande bron- en eindbestemmingen bevat, zodat deze matrix nooit hoeft te worden aangepast door de gebruiker. Hij kan wel bij dit bestand komen als er een nieuwe locatie toegevoegd moet worden, maar verder hoeft de gebruiker in feite niets met deze matrix te doen.

De gebruiker kan ook kiezen wat de transportrichting van de containers is. Standaard volgt de Distance Matrix een 'van west naar oost' richting. Indien de andere richting wordt gekozen, zal de Distance Matrix d.m.v. de 'transpose()' methode van Sebastiaan Meijer omgezet worden naar een getransponeerde Distance Matrix die overeenkomt met deze transportrichting. Een tweede keuzemogelijkheid binnen de Distance Matrix is de Hub. In dit stadium van het Twin Hub Netwerk-Project wordt er gewerkt met één hub, namelijk Antwerpen. Wanneer er voor Rotterdam ook probleeminstanties worden gemaakt is het idee een tweede Distance Matrix-bestand te creëren,

waarbij de applicatie het juiste bestand selecteert en deze gebruikt aan de hand van de keuze van de gebruiker (Zie ook het hoofdstuk Aanbevelingen).

4.4.5 Koppeling Flow Matrix - Distance Matrix

Omdat de gebruiker in eerste instantie niets te maken heeft met de Distance Matrix en we de gebruiker de vrijheid willen geven alle flows in een zelfgekozen volgorde in te voeren is er in het programma de methode 'getFlow Matrix' in 'JTableToFlow Matrix' geschreven die de Flow Matrix zo converteert dat deze in de juiste vorm wordt opgeslagen om overeen te komen met de Distance Matrix. Deze methode maakt logischerwijs dan ook direct gebruik van de Distance Matrix.

Aan de hand van de bronbestemmingen van de Distance Matrix wordt er een HashMap gecreëerd waaruit opgezocht kan worden in welke rij de bronbestemmingen uit de Flow Matrix opgeslagen moeten worden. Bestemmingen die niet genoemd worden in de Flow Matrix zullen in de uiteindelijke Flow Matrix allemaal flows van nul krijgen. Wanneer alle rijen correct staan wordt met een soortgelijke HashMap opgezocht in welke kolommen de flows moeten komen staan om bij de juiste eindbestemmingen terecht te komen. Ook hier worden lege velden met nullen ingevuld. Op deze manier krijg je een Flow Matrix die de juiste dimensie en data op de juiste plek heeft.

4.4.6 CoordinateList

Een nieuw onderdeel in onze applicatie is de coördinatenlijst (in het programma CoordinateList). Om alle locatie op de juiste plaats op de wereldkaart te tekenen is het nodig van elke locatie de wereldcoördinaten (latitude en longitude) te kennen. Deze lijst wordt in een CSV-bestand opgeslagen.

Net als de Distance Matrix is het voor de gebruiker niet direct nodig om deze lijst te kennen. Het is voor de gebruiker wel mogelijk om bij dit bestand te komen, om eventueel coördinaten en locatienamen aan te passen of nieuwe locaties toe te voegen.

4.4.7 Validatie van de Inputs

Om fouten tijdens het berekenen van de oplossing te voorkomen en de gebruiker te ondersteunen in het creëren van een kloppende probleeminstantie, moeten de grote 'input'-onderdelen eerst gevalideerd worden in de applicatie voordat er op 'Run' gedrukt kan worden. Door een uitgebreide validatie kan er gedetailleerd aangegeven worden aan de gebruiker waar het mis gaat.

Het gaat hier voornamelijk om de coördinatenlijst, Distance Matrix en Flow Matrix. Voordat er gekeken wordt of de koppeling tussen de verschillende bestanden klopt, vindt er losse validatie plaats.

Validatie van de Flow Matrix

Om ervoor te zorgen dat er een logische probleem instantie gecreëerd kan worden, wordt de Flow Matrix op de volgende eisen gecontroleerd:

- Cell (0,0) moet leeg zijn. Deze eis is niet direct functioneel, maar op deze manier kan er snel gekeken worden of er een Flow Matrix is ingevoerd i.p.v. een andere random file.
- Alle bron- en eindbestemmingen moeten zijn ingevuld. Lege bestemmingen zullen nooit voorkomen in de Distance Matrix en coördinatenlijst.
- Alle flows moeten integers zijn.

Validatie van de Distance Matrix

De Distance Matrix (die in de meeste gevallen niet aangeraakt zal worden door de gebruiker) wordt op de volgende punten gecontroleerd:

- De vorm van de Distance Matrix (de verschillende afstandsblokken) moet kloppen. De blokindeling van de Distance Matrix is te vinden in appendix A.
- Alle afstanden moeten zijn ingevuld (met getallen), evenals alle bestemmingen.
- De file moet bestaan (de csv-file mag niet verwijderd of hernoemd worden).

Validatie van de CoordinateList

De coördinatenlijst (die in de meeste gevallen ook niet aangepast wordt door de gebruiker) wordt op de volgende punten gecontroleerd:

- Het bestand moet rechthoekig zijn en maar 3 breed (naam, x-coordinaat, y-coordinaat).
- Alle coördinaten moeten Doubles zijn.
- De file moet bestaan (de csv-file mag niet verwijderd of hernoemd worden).

Koppeling tussen de Distance Matrix en CoordinateList

Wanneer alle drie de onderdelen voldoen aan de bovenstaande eisen, kunnen deze worden gecreëerd en kan gekeken worden of alle koppelingen kloppen. Er wordt door de InputValidator na de losse validaties als eerst gekeken of alle locaties genoemd in de Distance Matrix wel in de CoordinateList voorkomen. In principe is dit altijd waar, omdat de gebruiker beide bestanden niet hoeft aan te passen. Echter, op deze manier kan de gebruiker gewaarschuwd worden wanneer hij een nieuwe locatie aan de Distance Matrix heeft toegevoegd maar deze nog niet heeft 'gedeclareerd' in de CoordinateList.

Het kan natuurlijk zijn dat terwijl deze koppeling niet klopt, er wel een geldige Probleem Instantie gecreëerd kan worden, wanneer de locaties met missende coördinaten ook niet in de Flow Matrix voorkomen. Echter, we hebben voor deze validatie gekozen om ervoor te zorgen dat wanneer de gebruiker de Distance Matrix aanpast direct ook de coordinatenlijst moet aanpassen. Het kan wel zijn dat er locaties in de coördinatenlijst staan die niet genoemd zijn in de Distance Matrix.

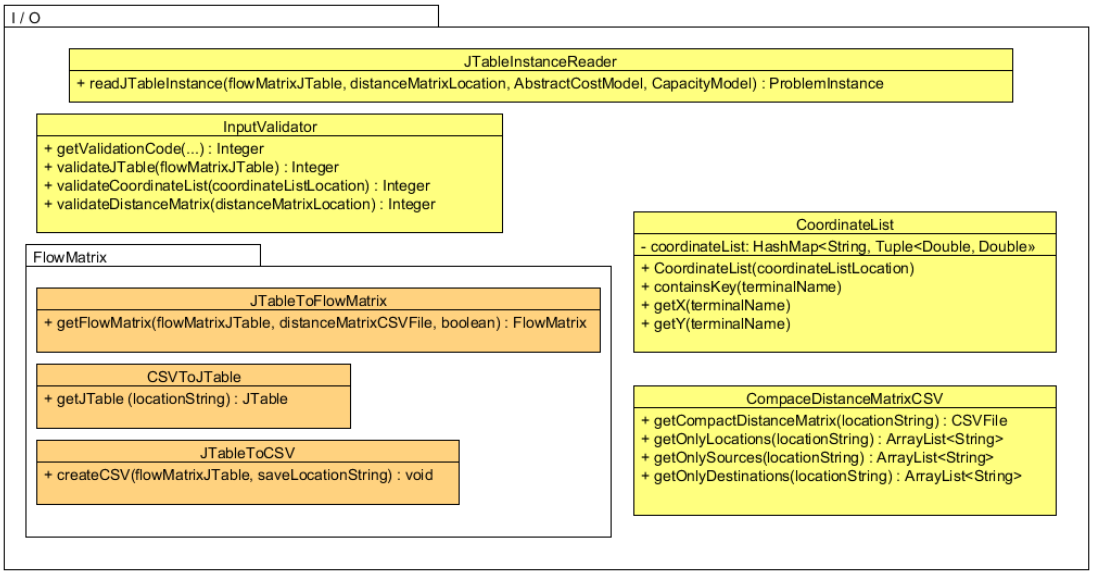
Koppeling tussen de Flow Matrix en Distance Matrix

Het laatste dat gecontroleerd wordt is of alle bronbestemmingen ook als bronbestemmingen gedefinieerd zijn in de Distance Matrix en of alle eindbestemmingen ook als eindbestemmingen zijn gedefinieerd in de Distance Matrix. Wanneer dit het geval is, kan zonder enige problemen een probleeminstantie worden aangemaakt en kan de Solver een oplossing uitrekenen.

4.5 Tijdens de werking van de Gurobi Solver

Terwijl de Solver bezig is oplossingen te vinden wordt bijna hetzelfde weergegeven als bij de originele Bundlingtool. Aan deze weergave hebben wij een grafiek toegevoegd die de kosten ten opzichte van de tijd plot. De grafiek zal twee lijnen plotten; De kosten van de lower bound en van de gevonden oplossing(en). De grafiek die wij gebruiken is een SWT Widget XYGraph. Deze heeft ook een toolbar waarmee kan worden gezoomd, notities worden gemaakt en de grafiek worden opgeslagen als PNG-bestand.

Voor dit gedeelte van de tool hebben we geen klassen moeten schrijven en toevoegen omdat het grotendeels hetzelfde is gebleven en omdat de werking van de Solver buiten onze handen plaatsvindt.



Figuur 4.6: (Vereenvoudigd) Overzicht van alle I/O klassen

4.6 Opzet van de Applicatie / Interface

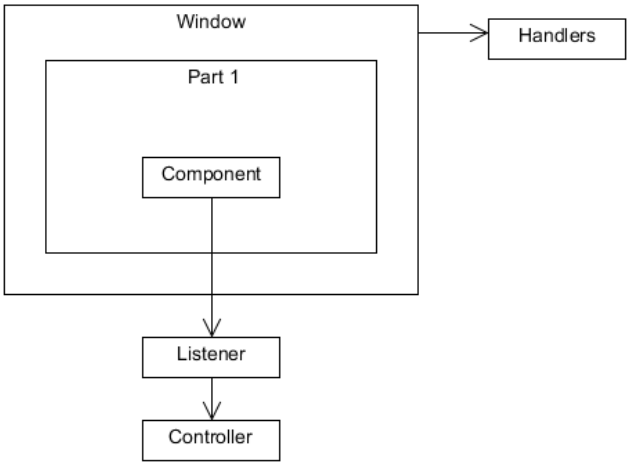
De Twinhub Bundeling Tool is geprogrammeerd in het Eclipse Rich Client Platform. Het Rich Client Platform zorgt ervoor dat er snel een professioneel uitziende applicatie gemaakt kan worden. Zo kan er meer focus liggen op het ontwikkelen van de functionaliteit en minder op het afstellen van de interface. Het RCP framework is gebaseerd op plug-ins. Dit houdt in dat onderdelen als plug-in worden doorgevoerd. In de originele Twinhub Bundling Tool waren al verschillende aspecten onderverdeeld in projecten en stelden in de Eclipse RCP omgeving plug-ins voor. Wij hebben bij het uitbreiden van de applicatie nog meer plug-ins (ofwel projecten) toegevoegd en het originele GUI gedeelte omgebouwd.

Met Eclipse RCP is het eenvoudig de GUI vorm te geven via het RCP applicatie model. Dit bestaat uit 'Windows' (de schermen van de interface) en de 'Parts' (de lossen GUI onderdelen zoals widgets en frames) die erin behoren. Een Part verwijst naar de implementatie (klasse) van een bijbehorend onderdeel. We hebben dit model gevolgd en de GUI onderverdeeld in verschillende Parts die worden weergegeven in tabbladen in de nieuwe Twinhub Bundling Tool.

Binnen de Parts zijn Listeners en Controllers nodig die de werking van functies in de tool kunnen uitvoeren. De Listeners vangen events op van componenten in de Parts. De Controllers zorgen vervolgens voor het uitvoeren van de gewenste functie. Ook zijn er Handlers die handelingen buiten de Parts kunnen uitvoeren. Zie figuur 4.7.

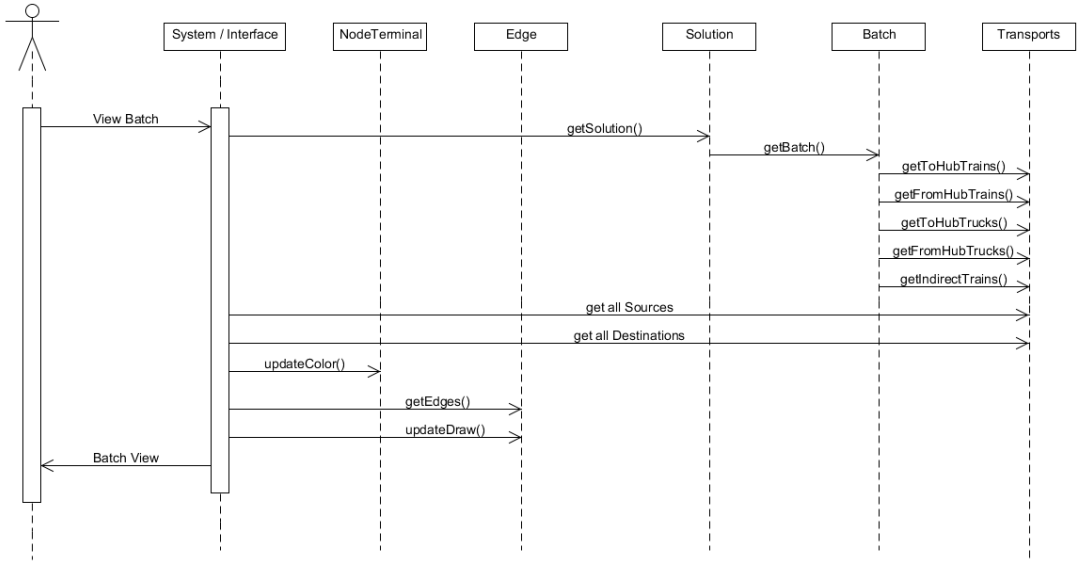
4.7 Sequencediagrammen

Tijdens de ontwerpfase hebben we verschillende sequencediagrammen gemaakt die als leidraad fungeerden over hoe we bepaalde onderdelen van de applicatie wilden implementeren. De drie voornaamste modellen die we hebben gebruikt zijn het sequencediagram voor het weergeven van een BatchGraph, het verkrijgen van details van een Node en het invoeren van de Matrices. Hieronder tonen we deze diagrammen en leggen we kort uit welke methoden en klassen de applicatie gebruikt om het weergegeven verloop te realiseren.



Figuur 4.7: Diagram van het Application Model

4.7.1 Graph Weergave



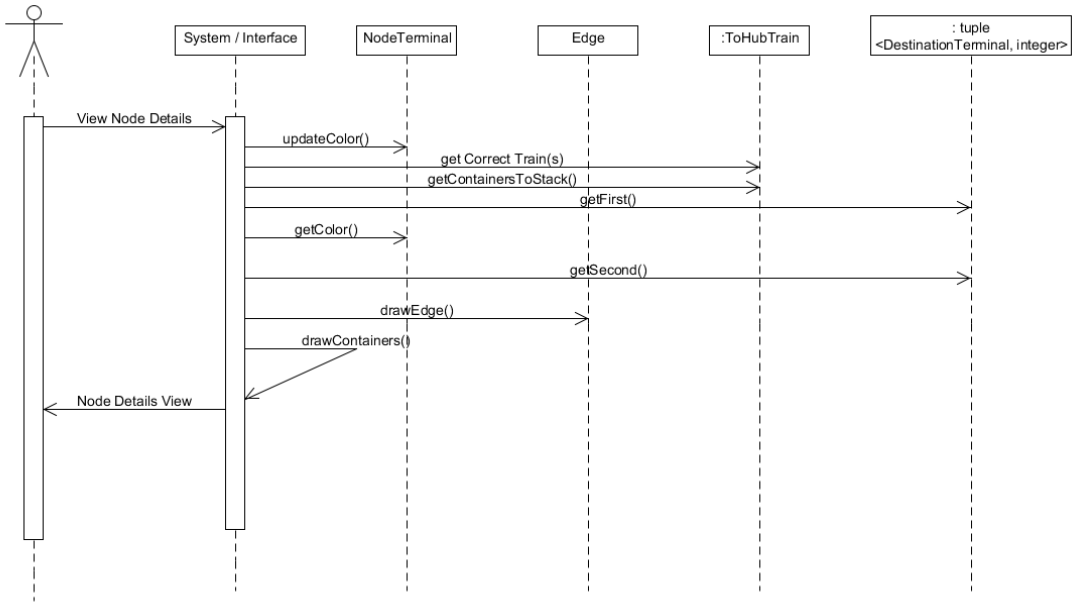
Figuur 4.8: Sequencediagram voor de Graph weergave

Bij het creëren van een BatchGraph hebben we figuur 4.8 gevolgd. Uiteindelijke stappen zijn terug te vinden in de methode 'createBatchGraph(..)' in de klasse 'GraphParser'. Deze methode haalt (met behulp van submethoden die allemaal ook in de GraphParser te vinden zijn, allereerst alle Destination- en SourceNodes op en creëert dan één voor één de lijsten met verschillende soorten Edges die nodig zijn om de BatchGraph te creëren.

Deze opzet is in volgorde net iets anders dan we in het ontwerp hadden bedacht, maar het is nodig om eerst Nodes te creëren omdat het anders niet mogelijk is om Edges samen te stellen. De methode is nog wel uitgebreid door naar de gevonden Edges te kijken en vervolgens alleen de relevante Source- en DestinationNodes door te sturen naar de constructor van de BatchGraph.

De weergave van het BatchGraph Object wordt gerealiseerd in de methode 'createBatch-Graph(..)' van de klasse 'SolutionVisualizer'.

4.7.2 Node Details Weergave



Figuur 4.9: Sequencediagram voor de Node details weergave

Voor het verkrijgen van de details van een bepaalde Node volgen we figuur 4.9. Hier bespreken we de implementatie voor details rond een SourceNode, maar die rond DestinationNode is bijna geheel hetzelfde.

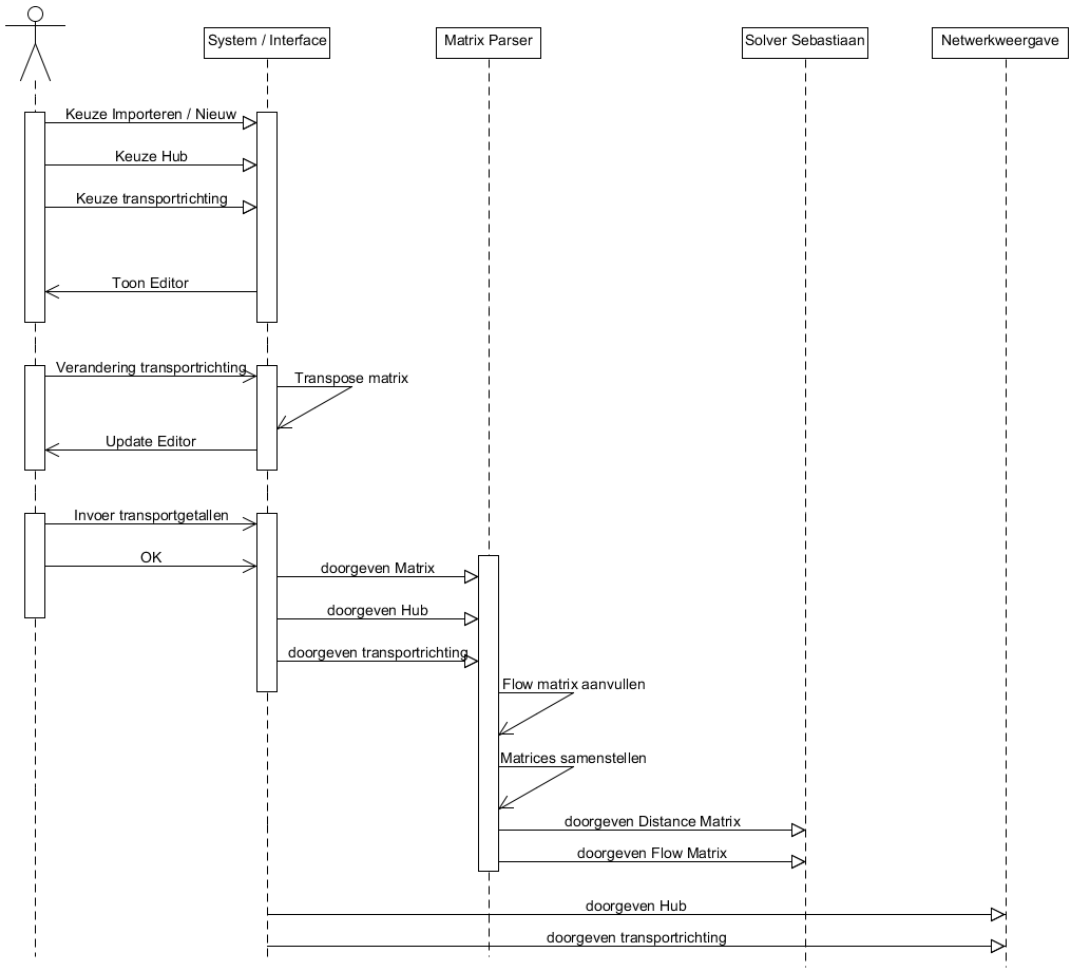
De informatie die de gebruiker wil zien wordt verzameld in de klasse BatchGraph, middels de methode 'getSourceDetails(..)', die gebruik maakt van de methode 'getToHubTransportDetails(..)'. Er worden verschillende transporten bekeken, in dit diagram de ToHubTrain. Van de totale verzameling ToHubTrains wordt gekeken welke er relevant zijn voor de aangeklikte Node, ofwel welke treinen er vanaf deze Node vertrekken. Van deze treinen worden alle containers verzameld. Een container is een 'koppel' (een zelfgemaakte Tuple klasse) met een DestinationTerminal en een aantal en de methode slaat deze op als een nieuwe Tuple met een DestinationNode en hetzelfde aantal.

De methode 'drawDetailBlock(..)' in de SolutionVizualizer gebruikt deze verzamelde informatie om vervolgens een blok te tekenen. De methode gebruikt de kleur van de DestinationNode en het aantal containers dat hier naartoe gaat om deze te tekenen. De totale rij verschillende containerblokken wordt getekend met de methode 'drawBlockRowSource(..)', die eveneens in de SolutionVizualizer klasse te vinden is.

4.7.3 Matrix Invoer

Aan het begin van het ontwerp was al duidelijk dat de applicatie rekening moest kunnen houden met verschillende Hubs en transportrichtingen en moest er een Editor gebouwd worden om de Flow Matrix aan te passen. Vooral het onderste gedeelte van het diagram in figuur 4.10 hebben we gevolgd bij het opzetten van onze 'JTableInstanceReader' klasse.

In de 'readJTableInstance(..)' methode worden de door de gebruiker ingevoerde matrices om de beurt doorgevoerd. De Flow Matrix die in de Editor wordt weergegeven is op dat moment nog



Figuur 4.10: Sequencediagram voor de Matrix Invoer

een JTable Object. Aan de hand van de FlowDirection (0 of 1) wordt er in de klasse 'JTableToFlow Matrix' de bijbehorende methode aangeroepen die van deze JTable een Flow Matrix Object volgens het model van Sebastiaan creëert, die aangevuld en opgesteld wordt aan de hand van de Distance Matrix. Vervolgens worden deze twee matrices doorgevoerd aan de Solver.

5 Ontwikkelproces

In dit hoofdstuk bespreken we hoe we de ontwikkeling hebben aangepakt, welke ontwikkelmethode we hebben gevolgd en welke hulpmiddelen we hebben gebruikt. We zullen eerst ingaan op onze ervaring in het project. Vervolgens bespreken we de hulpmiddelen en hoe we het gebruik van deze hebben ervaren.

5.1 Projectproces

Tijdens het project is deels gebruik gemaakt van de principes van 'Scrum'. Zo hebben we de totaalperiode ingedeeld in kleine perioden met eigen deadlines. Bij 'Scrum' heet dit 'Sprints', wij noemden het Milestones.

Elke periode duurde ongeveer twee weken, waarbij we deze periode afsloten met een aantal doelen (Milestones) wie we wilden bereiken. Tijdens deze perioden was er één persoon aangewezen als eindverantwoordelijke. Er werd niet dagelijks vergaderd, maar elke werkdag hebben we wel met zijn drieën aan de ontwikkeling gewerkt. Er werd veel onderling overlegd en besproken welke taken er nodig waren om de deadlines van de Milestone te halen. Elke week was er een vergadering met Michel die ons kon controleren en ons tips gaf wat we verder nog konden doen / aanpassen.

Het was fijn de lange periode op te delen in verschillende perioden met eigen deadlines. Op deze manier konden we goed controleren of we op schema liepen. Daarnaast maakte dit het makkelijker om taken te bepalen en te verdelen.

Hieronder zullen we we verschillende kleine perioden los bespreken. Hierin bespreken we de doelen die we voor deze perioden hadden gesteld en hoe de periode verliep.

5.1.1 Oriëntatie- en ontwerpfase

De eerste drie weken zijn we bezig geweest met de oriëntatie van het project en het ontwerp van de applicatie. Resultaat hiervan zijn het oriëntatieverslag en requirementsdocument (appendix B en C). In deze periode hebben we tweemaal vergaderd met de opdrachtgever. Allereerst werd er gesproken over het probleem en werden de eerste wensen besproken. Aan de hand van dit gesprek zijn we ons gaan verdiepen in het algoritme van het Twinhub Bundling Probleem - op welke manier kunnen wij uit een oplossing verkrijgen wat we nodig hebben? - en hebben we een eerste ontwerp opgemaakt van de applicatie. Hierbij hebben we verschillende opties onderzocht en gekeken of het, zoals de wens van de opdrachtgever was, mogelijk was om de oplossing overzichtelijk in een graaf te tonen. De onderzochte alternatieven en de eisen van het programma hebben we besproken met de opdrachtgever. Hierna zijn we begonnen met programmeren.

5.1.2 Milestonemoment 1

Het eerste milestonemoment viel twee weken nadat we begonnen waren met programmeren. Hiervoor hadden we drie duidelijke doelen. Allererst moesten we een koppeling creëren van de oplossing van Sebastiaan naar een datastructuur die we konden gebruiken voor de weergave van een graaf (implementatie hiervan is gepresenteerd in hoofdstuk 4.2). Daarnaast werd onderzocht met behulp van welke librairies we de graaf konden tekenen in de applicatie. Het derde doel was aan het eind van deze periode de opzet van de applicatie werkende te hebben in RCP, zodat de verschillende onderdelen er daarna gemakkelijk ingezet konden worden.

Deze eerste periode verliep erg productief. Na de eerste week was de implementatie van de datastructuur klaar en kon deze gebruikt worden om te experimenteren met het tekenen. Daarnaast kon er al begonnen worden aan de implementatie van methoden om details van de graaf op

te vragen, zodat deze aan het eind van de periode ook klaar waren. Ook de opzet van de applicatie was op tijd af; de basis van de weergave van de graaf was hier nog niet in geïntegreerd, maar de opzet bevatte al wel alle basisonderdelen uit de vorige applicatie.

5.1.3 Milestonemoment 2

De tweede periode hadden we nieuwe doelen gesteld: Van de graaf wilden we nu ook de details kunnen opvragen en met kleuren de weergave duidelijker te maken. Tegelijkertijd wilden we deze weergaven gaan integreren in de applicatie (zie hoofdstuk 4.3). Ook zijn we begonnen met de invoer: Het vastleggen van de Distance Matrix werd uit de handen van de gebruiker gehaald, en we hebben gewerkt aan de mogelijkheid een Flow Matrix op te zetten en/of te wijzigen in de applicatie zelf. Omdat dit redelijk soepel verliep is dit laatste ook uitgebreid met de validatie van de verschillende onderdelen (zie hoofdstuk 4.4).

Het tweede milestonemoment viel anderhalve week na het eerste milestonemoment, zodat de donderdag en vrijdag daarop gekeken kon worden hoe we onze code naar het SIG konden sturen. Omdat we hierover verkeerd waren ingelicht heeft dit wat vertraging opgelopen en hebben we de twee dagen een opzet van het verslag gemaakt en aan kleine verbeteringen gewerkt.

5.1.4 Milestonemoment 3

De derde programmeerperiode was er op aanraden van Michel afgesproken geen grote extra functionaliteiten aan de applicatie toe te voegen, maar ons te richten op het 'afmaken' van de applicatie. Er werd voor gezorgd dat alle losse onderdelen soepel in elkaar overliepen en we richtten onze aandacht op het verhelpen van kleine foutjes, zodat de applicatie echt 'af' was.

Deze laatste periode verliep moeizaam, omdat het niet direct afgebakend was wat er nog kon gebeuren. Doordat er gewacht werd op feedback van zowel de opdrachtgever als de SIG kon er niet elke keer hard doorgewerkt worden. Doordat we vastliepen op het programmeren was er wel de tijd om een opzet voor dit verslag te maken, zodat dit ons uiteindelijk stress zou voorkomen tijdens de afronding.

5.1.5 Afronding

De laatste week van de implementatiefase (tot aan een week voor de eindpresentatie) waren we druk met het afronden van het project. Er was aan de opdrachtgever een demo gegeven waarop hij feedback kon geven over wat hij nog miste aan de applicatie of net anders wilde zien. Op zijn wens hebben wij een extra globale graafweergave toegevoegd aan de applicatie waar de indirecte treinen de hoofdrol spelen. Ook hebben we onderzocht hoe we altijd konden laten zien wat de transportrichting was en hoe we dit nog konden verwerken in de applicatie. Aan de hand van een random gekozen bronterminal uit een oplossing wordt gekeken of deze in de set bron- of eindbestemmingen van de Distance Matrix valt. Hieruit valt de transportrichting te achterhalen.

Naast deze kleine implementatie wijzigingen hebben we dit verslag afgemaakt. Wanneer de applicatie en het verslag worden ingeleverd zullen we gaan werken aan de eindpresentatie en een tutorial voor de opdrachtgever, zodat hij na het project direct aan de slag kan met de applicatie.

5.2 Hulpmiddelen

Tijdens het project hebben we gebruik gemaakt van verschillende hulpmiddelen. Welke hulpmiddelen we in eerste instantie wilden gebruiken hebben we uiteengezet in het Oriëntatieverslag (appendix B). In deze paragraaf zullen we bespreken hoe we deze hulpmiddelen uiteindelijk in het project hebben gebruikt. Ook zullen we onze mening geven over deze hulpmiddelen.

5.2.1 Java / RCP

Onze applicatie is geschreven in Java in combinatie met het Rich Client Platform (RCP) van Eclipse. Dit hebben wij gekozen omdat Sebastiaan al zijn code ook in deze taal had geschreven en op deze manier de koppeling tussen zijn en onze code het gemakkelijkst was.

Het programmeren in Java beviel ons zeer, omdat we allen de meeste programmeerervaring hadden met deze taal. Echter, RCP was geheel nieuw voor ons. Het was hierom dat het belangrijk was in de eerste periode goed kennis te maken met RCP. Carlo heeft hier de meeste tijd ingestoken. Door te kijken naar de code van Sebastiaan en tutorials te volgen hebben we RCP redelijk onder de knie gekregen. Indien de rest met vragen zat konden ze terecht bij Carlo of konden we onze vragen aan Sebastiaan stellen.

Wij vinden dat Eclipse RCP een hoge leer drempel heeft omdat het een lange tutorial kost om eraan te beginnen. Nadat we RCP enigszins onder de knie hadden was het redelijk gemakkelijk de tool van Sebastiaan aan te passen en nieuwe onderdelen toe te voegen. Echter, omdat RCP de SWT (Standard Widget Toolkit) gebruikt voor het maken van de GUI ondervonden we soms moeilijkheden wanneer we al bestaande onderdelen wilden toevoegen die bestonden uit andere toolkits. De dependencies van een plug-in in RCP vonden wij soms verwarrend omdat deze dependencies soms meerdere keren moeten wordt gedefinieerd.

5.2.2 Checkstyle

Zoals afgesproken hebben we er naar gestreefd onze code zo netjes mogelijk af te leveren volgens checkstyle. Deze plug-in heeft ons geholpen de code overzichtelijk te maken en ons er aan te herinneren dat we Javadoc-commentaar moesten schrijven.

Het was fijn om met Checkstyle te werken. Toch waren er een aantal 'eisen' van de plug-in die wij niet hebben gevolgd. Zo was het onmogelijk voor ons om de lijnlengte onder de 80 karakters te houden. We kozen ervoor om langere maar veelzeggende namen toe te wijzen aan variabelen en methoden, waardoor de lijnlengte vaker de 80 karakters overschreed.

5.2.3 Gurobi

Om de applicatie van Sebastiaan werkende te krijgen moesten we (naast de RCP ondersteunende versie van Eclipse) de software van Gurobi Optimization installeren. De Solver gebruikt de software van Gurobi om de oplossing te berekenen. Op dit moment werkt de Solver met Gurobi 5.1.1.

We hebben ons tijdens de oriëntatieperiode een dag moeten verdiepen in het werkende krijgen van de originele code van Sebastiaan, inclusief het installeren van de benodigde software. Sebastiaan heeft een duidelijke handleiding geschreven om het werkende te krijgen, dus het is ons relatief makkelijk gelukt om de basis werkende te krijgen. Verder hebben we niets hoeven doen met de Gurobi Software.

5.2.4 Mantis

Als ticketsysteem hebben we Mantis gebruikt. Tijdens de oriëntatie en eerste twee implemenatieperioden hebben we goed gebruik gemaakt van dit hulpmiddel. Taken die aan het begin van de week werden verdeeld werden hier opgezet. Zo konden we bijhouden hoever we al waren.

Tijdens de derde implementatieperiode zijn we afgeweken van Mantis en hebben we in een Google Docs Spreadsheet een lijst met kleine taken gemaakt die moesten worden voltooid. Van deze lijst kon iemand die er tijd voor had een taak die hij wilde / kon doen kiezen. We hebben dit niet meer in Mantis gedaan omdat we op deze manier sneller konden zien wie waar mee bezig was en duidelijk bleef hoe groot de 'to do list' was. Voor een groter team of een groter project is Mantis misschien een betere oplossing, maar voor ons was het eenvoudiger om dit in een spreadsheet bij te houden.

5.2.5 Headless builds met Tycho/Maven

Voor het automatiseren van het build proces om de integriteit van de code te bewaken, moesten we eerst een manier vinden om de build uit te voeren buiten de Eclipse interface (headless). Dit is nodig omdat de build server zelf geen Eclipse interface kan aansturen. Hiervoor hebben we Maven gebruikt in combinatie met de Tycho build plug-in. Tycho is een speciale build tool voor het builden van Eclipse RCP applicaties. Omdat het vrij nieuwe software betreft zonder veel features en documentatie, was het erg lastig dit in eerste instantie werkend te krijgen. Hiervoor zijn verschillende XML files (pom.xml) geschreven die de definities van de build bevatten, zodat deze automatisch uitgevoerd kan worden.

5.2.6 Automatische builds met Jenkins Continuous Integration

Nadat het gelukt was om de code te builden zonder Eclipse interface, kon dit proces geautomatiseerd worden met Jenkins. We hebben hiervoor een Jenkins Continuous Integration Server opgezet. Hiermee wordt alle code van de teamleden samengevoegd en gecompileerd. Een paar keer per dag wordt de laatste code opgehaald uit de repository en gebuild, indien er een build failure is wordt een mail verstuurd naar de verantwoordelijke ontwikkelaar. Zo is er meteen feedback wanneer de code niet werkt en kunnen de fouten direct gecorrigeerd worden.

5.3 Testen

Voor het testen van de applicatie hebben we ons vooral gericht op het schrijven van Unit Tests. Dit om tijdens verdere ontwikkeling na ons project de integriteit van de bestaande code te garanderen.

5.3.1 JUnit

Om de code beter onderhoudbaar te maken en regressieproblemen in de toekomst te voorkomen, hebben we diverse Unit Tests geschreven. De meest gangbare oplossing hiervoor was het gebruiken van JUnit, een uitgebreid test framework voor Java. Hier hebben wij allemaal al veel ervaring mee en heeft een goede integratie in Eclipse.

5.3.2 Mockito

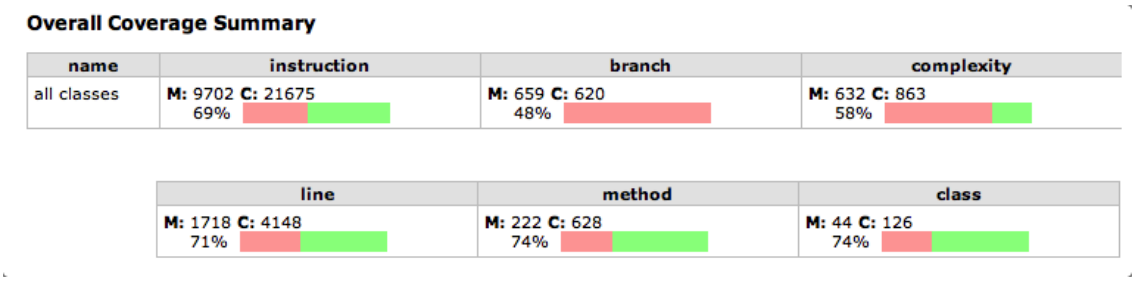
Bij sommige tests is het noodzakelijk bepaalde objecten te gebruiken. Om deze niet helemaal op te hoeven tuigen hebben we gebruik gemaakt van Mockito voor het mocken van deze objecten. Dit houdt in dat er een nep object wordt aangemaakt wat bepaalde acties van het echte object kan nabootsen, maar zonder dat de hele instantiëring van het object hoeft te worden uitgevoerd. Zo blijven de tests relatief simpel omdat alleen de strict noodzakelijke interactie voor de test hoeft te worden geïmplementeerd in een mock object. Voor andere tests was het wel noodzakelijk een volledig object te gebruiken. Bij het parsen van bestaande oplossingen hebben we zelf een eenvoudig netwerk opgezet voor de tests, hierbij was het niet nodig mocking te gebruiken.

5.3.3 Jenkins test automation en RCP

Voor het automatiseren van de tests wilden we ook graag Jenkins gebruiken, omdat we dit ook al gebruikten voor automatische builds. Om onze JUnit Tests met Tycho uit te kunnen voeren, was het belangrijk dat alle tests in hun eigen project kwamen te staan wat een speciaal eclipse-test-plug-in test project is. We hebben elk project zijn eigen test project gegeven waar de bijbehorende Unit Tests in komen te staan. Tycho herkent deze projecten en voert automatisch de tests uit. Zo hebben we niet alleen automatische feedback over de build maar ook de Unit Tests via Jenkins.

5.3.4 Code coverage

Om de effectiviteit van de Unit Tests te meten en overbodige tests te voorkomen, hebben we gebruik gemaakt van code coverage. In Eclipse is dit eenvoudig te meten met de EclEmma tool die een duidelijk overzicht geeft van welke statements gecovered worden door de tests. Om dit ook enigszins te automatiseren hebben we in Jenkins de JaCoCo plug-in toegevoegd die ook de code coverage bijhoudt en een grafiek laat zien van de code coverage over tijd. Zo kunnen we zien hoeveel van de code nog getest moet worden en of we daar progressie mee boeken. Een overzicht van onze code coverage is gegeven in figuur 5.1.



Figuur 5.1: Code coverage percentages van totale code.

5.4 Versiebeheer

Voor versiebeheer hebben we gebruik gemaakt van Git met een remote repository bij de repository site Assembla. We hebben in onze eigen branch gewerkt naast de branch van Sebastiaan, zodat er parallel aan bestaande code ontwikkeld kon worden. Binnen onze groep waren er soms wel merge conflicten bij het samenvoegen van code onderling, maar dit leverde geen problemen op.

5.5 Software Improvement Group

Drie weken voor de einddeadline hebben we onze code voor het eerst opgestuurd naar de Software Improvement Group. Diezelfde week hebben we een reactie gekregen. Hieronder zullen we bespreken wat de reactie van het SIG was en hoe we dit nog hebben kunnen verwerken.

5.5.1 Algemene Analyse

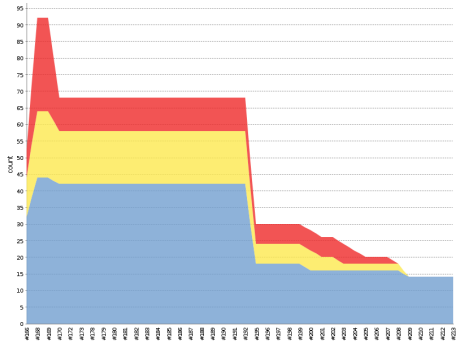
Het algemene verdict van de SIG was een score van drie sterren op het onderhoudbaarheidsmodel, wat betekent dat de code marktgemiddeld onderhoudbaar is. Vooral op de onderdelen 'duplication','unit size' en 'unit interfacing' konden we nog verbetingen doorvoeren.

Wat de SIG ook was opgevallen was dat onze code duidelijke componentisering heeft. We voldeden niet direct aan het standaard opdelen van de bron- en testcode. Hierop hebben we gereageerd richting de SIG dat het in ons geval onmogelijk was om deze code bij elkaar in één project te zetten, omdat wij werken met een Eclipse RCP applicatie (verder uitgelegd in paragraaf 5.3.3). Het automatiseren van het build/test proces met Tycho/Maven en Jenkins CI mislukte als we de test- en productiecode niet in aparte bundels zetten. De SIG heeft hierop gereageerd deze keuze te begrijpen.

De SIG was te spreken over het feit dat we JavaDoc hebben geschreven, maar voegde daaraan toe dat het weglaten van commentaar wanneer het geen toegevoegde waarde geeft soms een betere keuze is. Daarnaast vonden ze de aanwezigheid van de testcode veelbelovend.

5.5.2 Duplicatie

Het grootste probleem met onze code was de aanwezigheid van duplicaatcode. Hierbij gebruiken we dezelfde code in verschillende klassen. Dit maakt de code slecht onderhoudbaar, omdat een fout in een klasse niet noodzakelijk wordt opgelost in de andere klasse. Om hier grip op te krijgen hebben we na de feedback een CPD (Copy Paste Detection) plug-in geïnstalleerd in Jenkins CI. Deze detecteert automatisch de aanwezigheid van duplicaatcode en geeft een uitgebreide analyse weer. Hiermee konden we snel de pijnpunten in de code vinden en oplossen. Een overzicht van de aanwezigheid van duplicatie code is te zien in figuur 5.2. Naast duplicataatcode in verschillende klassen kwam er ook veel duplicaatcode voor binnen BatchGraph, gaf de SIG aan. Dit ging over de methoden 'get(Source/Destination)Details', waarin de verschillende Edges op dezelfde manier werden doorzocht. De SIG gaf aan dat deze blokken gemakkelijk in een methode gestopt kon worden en vervolgens meerdere keren kan worden aangeroepen. Het was wat gepuzzel door de verschillende 'soorten' Edges, maar het is uiteindelijk gelukt deze methode compact te maken door de blokken in eigen methoden te zetten.



Figuur 5.2: Code duplicatie per build in Jenkins, geel en rood geven kritieke problemen aan.

5.5.3 Unit Size

Voor de Unit Size is gekeken naar het percentage van de code dat bovengemiddeld lang is. Het argument is hier dat kleine methodes makkelijker te begrijpen en te testen zijn, waardoor de code ook beter onderhoudbaar is. Sommige methodes waren bij ons inderdaad te lang. Drie methoden die de SIG noemde zijn BatchGraph.getEdges(), MatrixPart.setValidationText() en SelectForcedBranchesPart.checkConsistency(). We zijn deze methoden afgegaan en hebben deze opgesplitst in meerdere kleine methoden om het overzichtelijker te maken.

5.5.4 Unit Interfacing

Bij unit interfacing is gekeken naar het aantal methodes met een bovengemiddeld aantal parameters. Het argument is dat een methode met veel parameters duidt op een gebrek aan abstractie. Ook leidt dit vaak tot complexere methoden en verwarring bij het aanroepen. Daarom is het wenselijk veel parameters te vermijden. We hebben geprobeerd om zo min mogelijk parameters te gebruiken voor methoden, maar dit is niet in alle gevallen gelukt. Voorbeelden hiervan zijn AlgorithmController.processRun(..) en RunConfigurationPart.createFileInputField(..). Ook de constructor van BatchGraph heeft 9 parameters nodig, dit komt door de verschillende soorten Edges die deze Graph heeft. Dit is ook onveranderd gebleven.

5.6 Integratie en acceptance

Het integreren van de diverse onderdelen ging redelijk gemakkelijk. Omdat we al unit tests hebben geschreven voor elk onderdeel, ligt het redelijk vast hoe deze zich zullen gedragen en konden deze eenvoudiger in andere delen worden geïntegreerd.

Nadat de ontwikkeling klaar was, zijn we door alle requirements heen gelopen om te controleren dat aan alle gestelde eisen is voldaan. We kunnen concluderen dat alle “Must-have” requirements zijn voldaan, maar dat bij “Should-have” en “Could-have” requirements nog zaken open zijn blijven staan. Sommige hiervan zijn opgenomen in het hoofdstuk aanbevelingen voor toekomstige ontwikkeling. We kunnen hiermee echter wel concluderen dat er een complete applicatie is opgeleverd die voldoet aan de minimale eisen en dus de acceptance test doorstaat.

6 Conclusie

Voor dit project hebben we een functionele applicatie opgeleverd die het Twinhub Bundling Probleem kan visualiseren. Hiermee is het in één oogopslag te zien hoe een Twinhub netwerk zal presteren en kunnen op basis hiervan conclusies worden getrokken. Zo is het ook eenvoudiger voor mensen zonder diepgaande kennis van het probleem om een mening te vormen over bepaalde oplossingen.

We hebben kunnen voldoen aan al onze vooraf gestelde "Must-have" requirements. Daarnaast is er nog aan een paar overige "Should-have" en "Could-have" requirements voldaan. De overige requirements zijn wellicht een goed startpunt voor verdere ontwikkeling van de applicatie. In het volgende hoofdstuk worden de aanbevelingen verder besproken.

Na het ontwerpen van onze applicatie zijn we begonnen met de ontwikkeling in drie fasen. We hebben voor het grootste deel aan deze planning vast kunnen houden maar hadden uiteindelijk niet veel tijd om nog extra features te implementeren. Door het gebruik van Unit Tests en Continuous Integration is de integriteit van de code goed bewaakt tijdens het proces en wij raden het gebruik hiervan dan ook zeker aan voor toekomstige projecten. Fouten worden hiermee in een vroeg stadium gedetecteerd en zorgen niet voor problemen achteraf.

De samenwerking verliep goed en we hadden allemaal al veel ervaring met programmeren in Java vanuit het bachelor programma. Het grootste struikelblok was het leren omgaan met Eclipse RCP, de combinatie van gebrekkige documentatie en een ingewikkelde structuur gaf dit een steile leercurve. Dit heeft ons meer tijd gekost dan gedacht.

We hebben in dit project veel kennis uit het bachelorprogramma in de praktijk kunnen brengen en zijn tevreden over het opgeleverde product.

7 Aanbevelingen

Ondanks dat wij voor het eindproject een complete applicatie hebben afgeleverd, zijn er nog zaken die in de toekomst uitgebreid kunnen worden. In dit hoofdstuk geven we een aantal aanbevelingen rondom de applicatie en de Solver.

7.1 Kleine Uitbreidingen

We noemen als eerst een aantal kleine onderdelen waarmee de applicatie kan worden uitgebreid. Hiermee laten we zien wat verder nog relatief eenvoudig geïmplementeerd kan worden.

Meedere Hubs

Op dit moment maakt het algoritme alleen nog maar gebruik van de Hub Antwerpen. Indien de Solver wordt uitgebreid kan de applicatie ook gemakkelijk op de volgende manier worden uitgebreid: Er moet een nieuw Distance Matrix-bestand worden opgezet met daarin de de afstanden met gebruik van Rotterdam. Bij de keuze van de Hub wordt dan het originele bestand of dit nieuwe bestand gekozen. Ook moet de juiste Hubnaam worden gevonden voor het samenvattingsgedeelte in het uitvoerscherm wanneer er een oplossing wordt geladen.

Aanpassen van de Distance Matrix binnen de applicatie

Op dit moment gebeurt het aanpassen van de Distance Matrix - ook al hoeft dit in feite nooit - buiten de applicatie. De applicatie zou kunnen worden uitgebreid door opties toe te voegen om details van locaties per locatie los te bekijken of aan te passen. Dit betreft dan de afstand tussen deze locatie en anderen, maar de locatie zou ook gemakkelijk een andere naam moeten kunnen krijgen of toegevoegd / verwijderd worden.

Keuzemogelijkheden bestaande locaties in Flow Matrix

Op dit moment moet de gebruiker, wanneer hij een Flow Matrix aanmaakt of aanpast, zelf bedenken welke locaties er bestaan in het netwerk en deze correct intypen. De applicatie zou kunnen worden uitgebreid door de Source- en Destinationblokken van de Flow Matrixtabel in de applicatie dropdown-mogelijkheden te geven, waarna er een lijst komt met alle locaties die kunnen worden gekozen. We hebben dit niet geïmplementeerd omdat we er vanuit gaan dat de gebruiker van de applicatie bekend is met het Twin Hub Netwerk en dus de te gebruiken locaties weet. Daarnaast is de weergave van de Flow Matrixtabel voornamelijk gecreëerd om de gebruiker een geïmporteerde Flow Matrix te kunnen controleren.

Extra weergave mogelijkheden graaf

Omdat de graaf weergave relatief eenvoudig uit te breiden is, zouden hier nog meer mogelijkheden ingebouwd kunnen worden. Gedacht kan worden aan de dikte van transportlijnen variëren op basis van aantal containers en het toevoegen van animaties voor het afspelen van de gehele oplossing. Ook zouden er in combinatie met betere interactie meer details weergegeven kunnen worden in de graaf, bijvoorbeeld een lijst met details die opent bij mouse-over op een Node.

7.2 Fictief Netwerk

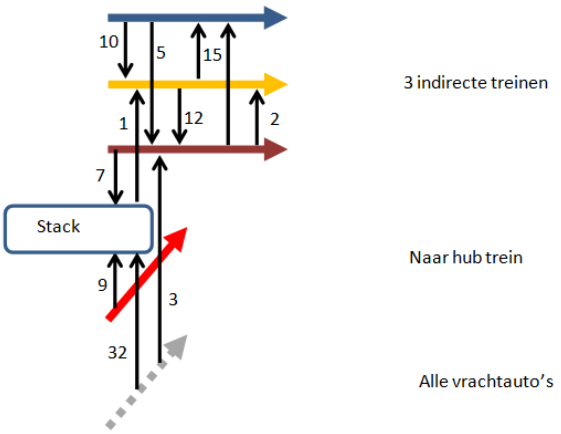
De applicatie die wij hebben afgeleverd maakt gebruikt van een 'vaste' Distance Matrix en Coördinatenlijst. De gebruiker kan deze twee bestanden buiten de applicatie aanpassen om zo een correct, 'vast' netwerk te creëren en te wijzigen. Op zich is het voor de gebruiker mogelijk om een fictief netwerk op te zetten, door een compleet nieuwe Distance Matrix en Coördinatenlijst op te zetten en deze onder de naam van de originele bestanden op te slaan. Echter, de kans is groot dat de applicatie dan minder goed werkt dan met het vaste netwerk. Allereerst is de kaart van de netwerkweergave er een van Noordwest-Europa. Wanneer de gebruiker een netwerk opzet in bijvoorbeeld Amerika, moet ook de kaart worden aangepast. Daarnaast is werken met relatieve coördinaten lastig, omdat de applicatie er nu vanuit gaat dat de doorgegeven waarden wereldcoördinaten zijn.

Er zijn ook punten die met het gebruik van een fictief netwerk niet netjes overkomen, maar geen directe problemen opleveren voor de functionaliteit. Allereerst wordt door de applicatie altijd aangegeven dat Antwerpen (of Rotterdam) als Hub wordt gebruikt. Daarnaast moet de transportrichting aangegeven worden, in dit geval is dat nog van West naar Oost en vice versa, maar van Noord naar Zuid en vice versa wordt op dit moment nog niet aangegeven. In principe kan dezelfde onderscheiding gebruikt worden - immers, Sources en Destinations worden omgedraaid - maar het komt niet overeen met wat er staat in de interface.

Het zou leuk zijn als de applicatie wordt uitgebreid dat het gebruiken van een fictief netwerk goed ondersteund worden. Het zou goed zijn als er een losse optie naast het gebruiken van het vaste netwerk zou bestaan waarbij de gebruiker in een paar stappen een fictief netwerk kan opzetten en gebruiken. In dit geval moet ook de kaartweergave worden aangepast als er een netwerk wordt opgezet dat niet op een kaart hoeft te worden weergegeven, maar relatieve coördinaten gebruikt, omdat dat zoals eerder gezegd problemen zal opleveren.

7.3 Hub Detail Weergave

Wij hebben ons dit project gefocused op de weergave van het totale netwerk. Een tweede gewenste weergave is de detailweergave van de Hub. Dit houdt in dat de gebruiker (per batch) kan zien hoe de goederen verwisseld worden van het ene transport (bijvoorbeeld een ToHubTrain) naar het andere transport (bijvoorbeeld een IndirectTrain). Het uitvoerscherm zou een extra tabblad kunnen krijgen waar deze weergave (per batch) kan worden getoond. Deze weergave is een wens van de opdrachtgever, in figuur 7.1 staat een schets die hij van de weergave heeft gemaakt:



Figuur 7.1: Details containerwisselingen op de Hub

7.4 Licenties

Voor het ontwikkelen van deze applicatie hebben we gebruik gemaakt van bibliotheken en een kaart die onder bepaalde Copyright licenties vallen. We hebben ervoor gezorgd dat deze voor onderzoeksdoeleinden gebruikt kunnen worden. Indien deze applicatie of delen ervan echter commercieel gebruikt gaan worden, is het belangrijk dat er nog steeds aan de licentie wordt voldaan. We zullen kort de onderdelen bespreken die onder licenties vallen en wat de impact hiervan is. Voor de exacte voorwaarden is het belangrijk de licentie zelf te raadplegen.

SWT-XY-graph

Voor het tekenen van de progress grafiek gebruiken we de swt-xy-graph bibliotheek. Deze valt onder de Eclipse Public License, hieronder is commercieel gebruik toegestaan [2].

JUNG

Voor het tekenen van de graaf van het netwerk gebruiken we de JUNG bibliotheek. Deze valt onder de BSD licentie. Alle aangepaste code en de bibliotheek zelf mogen commercieel gebruikt worden [1].

Kaart

De achtergrondkaart bij het netwerk is gebaseerd op informatie over NUTS gebieden samengesteld door Eurostat [3]. Commercieel gebruik van deze informatie is niet toegestaan en de kaart zal in dat geval vervangen moeten worden.

Gurobi

Voor het oplossen van het Twinhub probleem wordt gebruik gemaakt van de Gurobi Solver. Indien deze applicatie commercieel gebruikt wordt, is het noodzakelijk hiervoor een licentie aan te schaffen.

Bibliografie

[1] The bsd 3-clause license. http://opensource.org/licenses/BSD-3-Clause. [Online; bezocht 24-Juni-2013].

[2] Eclipse public license - v1.0. http://www.eclipse.org/legal/epl-v10.html. [Online; bezocht 24-Juni-2013].

[3] Eurostat home. http://epp.eurostat.ec.europa.eu/. [Online; bezocht 24-Juni-2013].

[4] Geotools the open source java gis toolkit. http://www.geotools.org/. [Online; bezocht 18-Juni-2013].

[5] Gephi, an open source graph visualization and manipulation software. https://gephi.org/. [Online; bezocht 18-Juni-2013].

[6] Graphviz - graph visualization software. http://www.graphviz.org/. [Online; bezocht 18-Juni-2013].

[7] How to choose colours procedurally (algorithms). http://devmag.org.za/2012/07/29/how-to-choose-colours-procedurally-algorithms/. [Online; bezocht 24-Juni-2013].

[8] Jgraphx. http://www.jgraph.com/jgraph.html. [Online; bezocht 18-Juni-2013].

[9] Jung - java universal network/graph framework. http://jung.sourceforge.net/. [Online; bezocht 18-Juni-2013].

[10] prefuse | interactive information visualization toolkit. http://prefuse.org/. [Online; bezocht 18-Juni-2013].

[11] Twin hub network. http://www.twinhubnetwork.eu/. [Online; bezocht 3-Mei-2013].

[12] Welcome to jgrapht - a free java graph library. http://jgrapht.org/. [Online; bezocht 18-Juni-2013].

[13] S. Meijer. Solving the twin hub bundling problem. Master's thesis, Delft University of Technology, 2012.

Appendices

A Opbouw van de Distance Matrix

Een Distance Matrix dient als hieronder opgebouwd te worden. Hierbij geldt dat tussen elk blok een witregel wordt geplaatst.

TREINEN	
DIRECTE AFSTAND	B
A	Directe afstand van A naar B met de trein

NAAR HUB AFSTAND	A
HUB	Afstand tussen de Hub en A met de trein

VAN HUB AFSTAND	B
HUB	Afstand tussen de Hub en B met de trein

TRUCKS	
DIRECTE AFSTAND	B
A	Directe afstand van A naar B

NAAR HUB AFSTAND	A
HUB	Afstand tussen de Hub en A met de vrachtwagen

VAN HUB AFSTAND	B
HUB	Afstand tussen de Hub en B met de vrachtwagen

	A
PREHAULAGE AFSTAND	Prehaulage afstand bij A

	B
POSTHAULAGE AFSTAND	Posthaulage afstand bij B

SOURCES	A
DESTINATIONS	B

B Orientatie Document

TI3800 Bachelorproject

Twin-Hub Network Project

Oriëntatieverslag

Plan van Aanpak

4008847 Marlou Pors
1170902 Jonathan Tetteroo
1308378 Carlo van der Valk

18 juni 2013

Inhoudsopgave

1 Introductie **5**

I Oriëntatieverslag **7**

2 Het Twin Hub Netwerk Project **8**
2.1 Het Twin Hub Netwerk-Project 8
2.2 Het Twin Hub Bundling Problem 9

3 Opdrachtgever **11**
3.1 Opdrachtgever 11
3.2 Supervisor(s) 11
3.3 Overig 11
3.4 Contact Gegevens 11

4 Voorgaand werk **12**
4.1 Probleem analyse 12
4.2 Algoritme 12
4.3 Twin Hub bundling Tool 12
4.3.1 Implementatie 12

II Plan van Aanpak **13**

5 Opdrachtopstelling **14**
5.1 Probleemstelling 14
5.2 Opdracht en doelstelling 14
5.3 Doelgroep 15

6 Aanpak **16**
6.1 Methodiek 16
6.2 Technieken 16
6.3 Werkzaamheden 16
6.4 Planning 17

7 Projectinrichting **19**
7.1 Betrokkenen 19
7.2 Faciliteiten 19

8 Kwaliteitsborging **20**
8.1 Kwaliteit 20
8.1.1 Ticket systeem 20
8.1.2 Continuous Integration 20
8.1.3 Checkstyle 20
8.2 Documentatie 20
8.3 Versiebeheer 20
8.3.1 Git 21
8.3.2 Assembla 21

8.4 Testen 21
8.4.1 Test driven development 21
8.4.2 Unit testing 21
8.4.3 Test coverage 21
8.5 Evaluatie 22
8.5.1 SIG 22

Bibliografie **22**

Appendices **23**

A Overzicht Output THB-Solver **24**
A.1 Algemene informatie over de oplossing 24
A.2 Oplossingsstatistieken 24

1 Introductie

Voor het Bacheloreindproject van de studie Technische Informatica is het onze opdracht een applicatie af te leveren die de uitwerking van een algoritmische solver voor het zogenaamde Twin Hub Bundling Probleem aantrekkelijk weergeeft en daarnaast gebruiksvriendelijk is voor personen die geen achtergrond hebben met deze solver. Dit verslag dient als resultaat van de oriëntatieperiode van twee weken aan het begin van dit project.

Het Twin Hub Netwerk-project is een samenwerkingsproject tussen zeehavens, inter-modale spoorwegondernemingen en universiteiten, met het doel inter-modaal goederenvervoer per spoor te stimuleren in Noordwest-Europa. Dit kan door het creëren van een netwerk dat winstgevend is voor alle betrokken partijen. Op welke manieren de goederen (vervoerd in containers) ingedeeld kunnen worden is een zeer beknopte beschrijving van het Twin Hub Bundling Probleem. Een algoritme om dit probleem op te lossen is al bedacht, maar tot op heden is er geen interface waarin oplossingen duidelijk getoond kunnen worden aan investeerders om zo duidelijk te maken wat het project inhoudt en wat de voordelen hiervan zijn. Dit is onze opdracht.

Dit verslag bevat onze verkenning van het Twin Hub Netwerk-project en de eerste stappen van het ontwerp voor de interface rondom de Solver van het Twin Hub Bundling Probleem. Het document zal uit twee delen bestaan: Het oriëntatiegedeelte en het Plan van Aanpak. Allereerst zullen we het Twin Hub Bundling Probleem uitleggen. Andere onderdelen van het oriëntatieonderzoek zijn een uiteenzetting van de personen en instanties waar we mee samenwerken en welke werken al voorafgaand van dit project zijn gedaan. In het Plan van Aanpak zullen we uiteenzetten op welke manier we dit project zullen aanpakken. We zullen de probleemstelling kort herhalen zodat we dit kunnen koppelen aan het doel en doelgroep van dit project. Ook presenteren we de planning en hulpmiddelen die we gebruiken.

Naast dit verslag zal er een Requirements Document gepresenteerd worden. Hierin zullen we een morfologische kaart presenteren en de requirements uiteenzetten. We eindigen met de uiteindelijke opzet van onze applicatie die geïllustreerd wordt met diagrammen.

Deel I

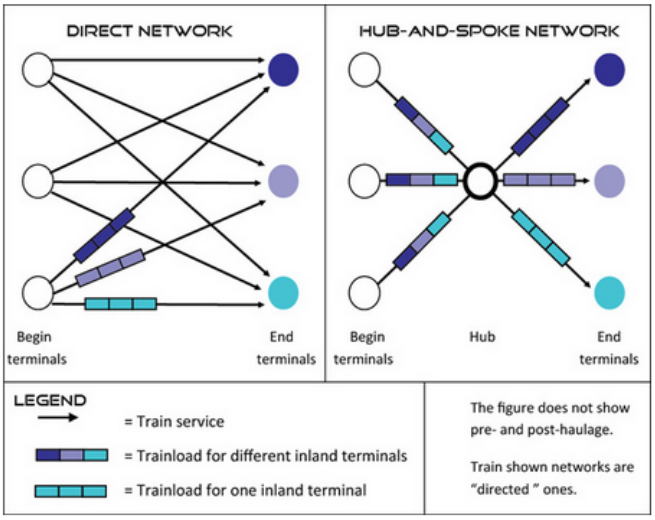
Oriëntatieverslag

2 Het Twin Hub Netwerk Project

2.1 Het Twin Hub Netwerk-Project

Het Twin Hub Netwerk-project is een samenwerkingsproject tussen zeehavens, inter-modale spoorwegondernemingen en universiteiten (o.a. de TU Delft), met het doel inter-modaal goederenvervoer per spoor te stimuleren in Noordwest Europa. Op dit moment vindt veel goederenvervoer in dit gebied nog plaats door de inzet van vrachtwagens, maar het Twin Hub Netwerk-project wil de inzet van treinen promoten zodat o.a. het transport duurzamer en het transportnetwerk robuuster wordt. Ook verhoogt dit de netwerk connectiviteit en territoriale samenhang binnen Noordwest-Europa.

Goederentransport met treinen kan alleen aantrekkelijk gemaakt worden wanneer er een netwerk gecreëerd wordt dat winstgevend is voor alle betrokken partijen. Dit betekent dat treinservices op een hoge frequentie moeten kunnen worden ingezet en de netwerkverbinding goed is. Het Twin-Hub-Netwerk Project hanteert in twee beslissingen om dit mogelijk te maken. Allereerst kiest men ervoor om het netwerk te bouwen in de vorm van een 'hub-and-spoke' netwerk (zie illustratie hieronder):



Figuur 2.1: Een direct en 'hub-and-spoke' netwerk; Twin Hub Netwerk Project site [Twi]

In een direct netwerk zullen treinen met een bepaalde lading een individuele route naar de eindbestemming volgen. Deze opzet is inefficiënt omdat er veel treinen nodig zijn (ladingen worden niet gecombineerd) en er veel verbindingen (treinsporen) nodig zijn. Het idee van het 'hub-and-spoke' netwerk is dat een gecombineerde trein eerst naar een zogenaamde hub reist, waarna de containers met verschillende eindbestemmingen gecombineerd worden met containers met dezelfde bestemming (afkomstig van een andere trein), waarna ze doorreizen naar de eindbestemming. Deze oplossing heeft minder treinen nodig, waardoor er services met hogere frequenties kunnen worden vervoerd. Daarnaast kunnen containers van en naar locaties die dicht naast elkaar liggen op deze manier ook met de trein vervoerd worden, zonder dat hier een directe verbinding tussen nodig is. In het voorbeeld hierboven is maar een derde van de treinen van het directe netwerk nodig voor het 'hub-and-spoke' netwerk.

De tweede beslissing die genomen is, is dat men uitgaat van twee hubs in Noordwest-Europa.

Zij hebben hier voor Rotterdam en Antwerpen gekozen. Hierover dient gezegd te worden dat deze transnationale samenwerking ervoor zorgt dat er grotere stromen kunnen ontstaan zonder dat daarbij het bestaan van concurrentie tussen zeehavens negeert.

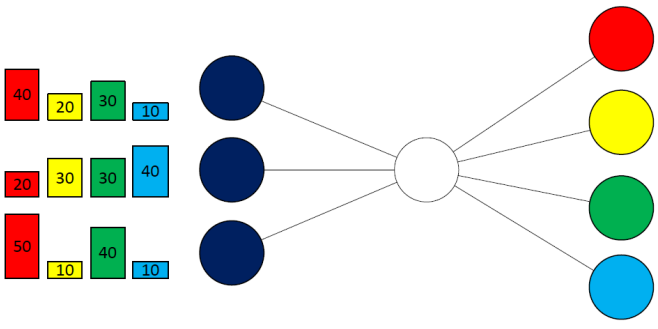
Naast het creÃ«ren van het totale treinnetwerk in Noordwest-Europa, moet er per subset berekend worden op welke manier het transport zal plaatsvinden. Dit is het samenstellen van zogenaamde Bundling Netwerken. We zullen hieronder beschrijven wat dit inhoudt.

2.2 Het Twin Hub Bundling Problem

Omdat het doel van het project is om op grote schaal te worden doorgevoerd (Noordwest-Europa), zijn de potentiële besparingen ook groot. Echter, er moet met veel factoren rekening worden gehouden om te berekenen of het transport nu goedkoper is per trein of per vrachtwagen. Extra kosten die treinvervoer met zich meenemen zijn bijvoorbeeld de kosten die plaatsvinden door het combineren van de treinen op de hubs en de extra kosten voor het vervoer van het treinstation naar de 'echte' eindbestemming (daar waar een vrachtwagen hier nog direct naar toe kan rijden en nu een extra vrachtwagen voor nodig is).

Het berekenen van de totale kosten voor opgegeven begin- en eindbestemmingen met daarbij behorende stromen tussen deze bestemmingen, waarna het bijbehorende Bundling Netwerk kan worden opgezet, is het Twin Hub Bundling (THB) Probleem. Om een geschikte oplossing voor het totale netwerk te vinden, kan het THB probleem worden opgedeeld in verschillende subproblemen. Allereerst kan worden aangenomen dat het transport over de hub Rotterdam en de hub Antwerpen niet van elkaar afhankelijk zijn, zodat probleeminstanties van deze hubs los van elkaar berekend kunnen worden. Daarnaast kan het probleem worden opgedeeld in verschillende groepen wat betreft afstand. Als voorbeeld: Het is zo dat transport dat over een afstand van twee dagen gaat niet gecombineerd kan worden met transport dat over een afstand van 3 dagen gaat. Een derde opdeling is dat er vervoer in één richting kan worden bekeken, omdat deze verschillende stromen ook onafhankelijk van elkaar zijn. Het vervoer van het binnenland naar de zeegebieden levert dus een losstaand probleem op en hangt niet samen met het vervoer van de zeegebieden naar het binnenland.

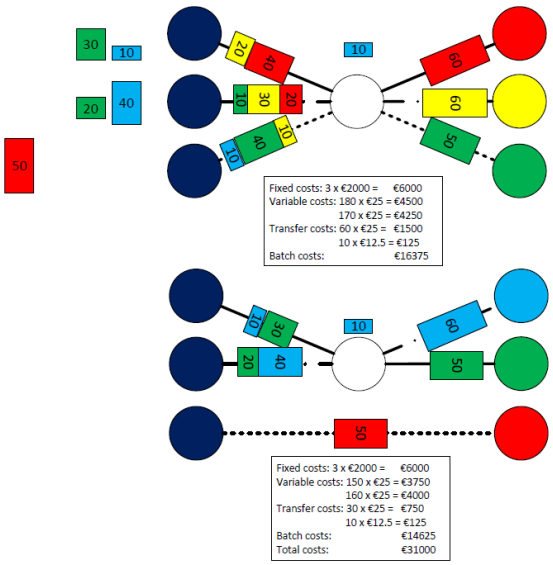
Een voorbeeldinstantie van het probleem is hieronder weergegeven in een graafvorm. Dit is in feite een subprobleem van het totale THB probleem, de hierboven genoemde afwegingen meegenomen:



Figuur 2.2: Een instantie van het THB Problem; Thesis S. Meijer [Mei12]

Om een Bundling Netwerk voor dit (sub)probleem te vinden kan er gebruik gemaakt worden van vijf verschillende vervoersopties: Treinen van beginbestemming naar de hub, treinen van de hub naar de eindbestemming, treinen van de beginbestemming naar de eindbestemming die via de hub reizen (indirecte treinen), vrachtwagens van beginbestemming naar de hub en vrachtwagens

van de hub naar eindbestemming. Andere voorbeelden van wat er te maken heeft met het THB probleem zijn dat het vervoer van alle containers kan worden opgedeeld in verschillende batches, oplossingen die behorende tot elkaar worden doorgevoerd (dan wel parallel met of direct na een andere batch). De optimale oplossing is deze die de minste kosten met zich meebrengt en er rekening gehouden moet worden met de capaciteit van de treinen (in dit voorbeeld 60 containers) en vrachtwagens (in dit voorbeeld één container).



Figuur 2.3: De optimale oplossing; Thesis S. Meijer [Mei12]

De optimale oplossing voor het voorbeeld is hierboven opnieuw in graafvorm weergeven. Ook is de kostenberekening weergegeven. Een algoritme dat een oplossing berekend voor een THB instantie moet rekening houden met veel meer punten dan hierboven genoemd. Dit is slechts een schets van wat het probleem inhoudt. Voor een formele definitie van het probleem verwijzen wij graag naar [Mei12]. Hierin staat ook het bewijs dat het THB probleem NP-hard is.

3 Opdrachtgever

3.1 Opdrachtgever

INTERREG NWE is een financieel instrument van het European Union Cohesion Policy waarin het Twin Hub Network Project zich bevindt. Het Twin Hub Network project heeft als doel om intermodaal spoorvervoer binnen en vanaf Noordwest-Europa meer concurrerend te maken. Het project opdracht is gegeven door Ekki Kreutzberger, coördinator van het Twin Hub Network project. Ekki is onze bedrijfsmentor die de eisen en wensen van het project eindproduct aangeeft.

3.2 Supervisor(s)

Cees Witteveen is Coach en Supervisor. Als Coach zal hij begeleiding geven aan de groep en de samenwerking en voortgang in de gaten houden. Als supervisor houdt hij een oog op het ontwerp en ontwikkeling van het software en geeft eventuele feedback. Michel Wilson is een betrokken PhD kandidaat die ook rol van coach biedt.

3.3 Overig

Sebastiaan Meijer is de auteur van het voorgaand werk in zijn master thesis “Solving the Twin Hub Bundling Problem". Wij zullen hem om advies vragen over zijn algoritme en bundling tool die nog verder wordt ontwikkeld door een andere afstudeerder.

3.4 Contact Gegevens

Dr. Ir. E. Kreutzberger
E.D.Kreutzberger@tudelft.nl

Prof. dr. C. Witteveen
C.Witteveen@tudelft.nl

M. Wilson MSc
M.Wilson@tudelft.nl

S. Meijer MSc
sjjmeijer@hotmail.com

4 Voorgaand werk

Dit project is een gebaseerd op de master thesis van Sebastiaan Meijer genaamd "Solving the Twin Hub Bundling Problem". Hierin wordt het Twin Hub probleem geanalyseerd, een algoritme is bedacht en een Twin Hub Bundling Tool die dient als solver voor het probleem is gebouwd. Wij zullen hier een korte samenvatting van geven.

4.1 Probleem analyse

In zijn thesis heeft Sebastiaan Meijer bewezen dat het Twin Hub Bundling probleem een NP-Hard optimalisatie probleem is door te bewijzen dat de beslissingsvariant van het Twin Hub probleem NP-Compleet is. Een optimale oplossing vinden met een algoritme zou exponentiële tijd kosten.

4.2 Algoritme

Het probleem wordt gemodelleerd als een Integer Linear Programming probleem om een oplossing te kunnen vinden.

Daarnaast wordt een relaxatie van het ILP probleem gebruikt, die geen geldige eind oplossingen geven maar wel helpen met de antwoord ruimte te begrenzen en een winst in tijd zal leveren. De oplossingen gevonden met de relaxatie kunnen weer als startpunt gebruikt worden voor het gewone ILP algoritme zodat hiermee sneller oplossingen gevonden kunnen worden. Door deze twee te combineren kan een tijdswinst behaald worden t.o.v. alleen het gewone ILP algoritme terwijl de resultaten vergelijkbaar blijven.

4.3 Twin Hub bundling Tool

De tool die Sebastian Meijer heeft geleverd werkt door middel van invoer van de gebruiker. Deze bestaat uit netwerk informatie en flow informatie ingevoerd als matrices in kommagescheiden bestanden (CSV).

Tijdens de werking van het algoritme zal de tool oplossingen vinden die geldig zijn maar niet optimaal. Elk oplossing wordt opgeslagen en kan worden bekeken. Wanneer de tool klaar is wordt het best gevonden resultaat weergegeven. Deze kan tot in de kleinste details worden bekeken. Een overzicht van alle output gegeven door de tool kan gevonden worden in Appendix A.

4.3.1 Implementatie

De tool is geïmplementeerd met het Eclipse Rich Client Platform, wat het mogelijk maakt eenvoudig user interfaces op te zetten in Java. Indien mogelijk zullen wij hier ook gebruik van maken tijdens de ontwikkeling. Daarnaast communiceert de code met de Gurobi Solver via een Java API.

Deel II

Plan van Aanpak

5 Opdrachtopstelling

Hier zullen we onze probleemstelling en doelgroep definiëren. Aan de hand hiervan kunnen we uiteenzetten wat onze opdracht en doelstelling is voor die project.

5.1 Probleemstelling

Het Twin Hub Netwerk-project is een samenwerkingsproject dat het doel heeft inter-modaal goederenvervoer per spoor te stimuleren in Noordwest-Europa. Op dit moment vindt nog veel goederenvervoer in dit gebied plaats door de inzet van vrachtwagens. Door de inzet van treinen wordt het transport duurzamer en het transportnetwerk robuuster.

Om het goederentransport met treinen aantrekkelijk te maken moet er een netwerk gecreëerd worden dat winstgevend is voor alle betrokken partijen. Treinservices moeten op hoge frequentie kunnen worden ingezet en netwerkverbindingen moeten goed zijn. Het Twin Hub Netwerk-project heeft als doel d.m.v. 'hub-and-spoke' netwerken zo'n groot netwerk op te zetten. Over dit netwerk moeten zogenaamde Bundling netwerken worden berekend en samengesteld. Deze netwerken geven aan op welke manier het vervoer van bekende stromen met containers moet worden ingedeeld, waarbij gebruik wordt gemaakt van treinen en vrachtwagens.

Op dit moment bestaat er een Solver die aan de hand van bekende begin- en eindbestemmingen en bijbehorende stromen kan berekenen wat de optimale indeling van het transport is (waarbij de kosten zo laag mogelijk worden gehouden). Echter, het invoeren van probleeminstanties en het bekijken van de oplossing (en daarbij behorende netwerk) is niet overzichtelijk. Het is een van de doelen van het Twin Hub Netwerk-project om investeerders en bedrijven over te halen zich bij het project aan te sluiten en hen duidelijk te maken hoe het transport door Noordwest Europa eruit komt te zien. Hier ligt dan ook onze opdracht.

5.2 Opdracht en doelstelling

Onze opdracht voor het Bacheloreindproject is een interface te bouwen die ervoor zorgt dat de invoer van data voor de Solver van het THB probleem en de uitvoer van de oplossing overzichtelijk zijn. Het is de bedoeling dat deze interface kan worden meegenomen in de promotie voor het Twin Hub Netwerk-project. Om dit te bewerkstellingen hebben we een aantal doelstellingen (ideeën) in gedachten:

- **De invoer van twee matrices wordt samengevoegd**
  Op dit moment werkt de Solver voor het THB probleem met twee matrices die moeten worden ingevoerd. Een matrix die vastlegt wat de afstanden zijn tussen verschillende bestemmingen en een matrix die vastlegt wat de stromen (per jaar/per dag) tussen verschillende bestemmingen zijn. In de reële wereld veranderen afstanden tussen bestemmingen niet, dus willen we het invoeren van deze matrix uit handen nemen van de gebruiker.

- **De Invoer moet kunnen worden geverifieerd**
  We willen de gebruiker de mogelijkheid geven de invoer van de matrix (die tot op heden heel groot en onoverzichtelijk kan worden) te kunnen checken binnen de interface. Daarnaast willen we de gebruiker de mogelijkheid geven binnen de interface een nieuwe 'flow-matrix' te kunnen maken.

- **Het opzetten van een vast netwerk, weergegeven als een graaf**
  We zullen een reëel netwerk vastleggen met alle bekende bestemmingen dat direct kan worden gebruikt (van het gehele netwerk kunnen subnetwerken gebruikt worden). Dit netwerk, en dus de invoer en uitvoer, zal worden weergegeven als een graaf. Van deze graaf zullen de bestemmingen op een 'logsiche' positie liggen om een duidelijk beeld voor de gebruiker te scheppen hoe het Twin Hub Netwerk er nu uitziet. Als voorbeeld: De bestemming Manchester zal links van Antwerpen liggen, met een langere afstand dan tussen Rotterdam en Antwerpen.

- **Mogelijkheden voor verder onderzoek in de interface**
  Een wenselijke doel, maar niet direct het belangrijkste, is het kunnen tonen van verschillende informatie die uiteindelijk gebruikt kan worden voor verder onderzoek van verschillende oplossingen. Hierbij bedoelen we dat verschillende oplossingen van de Solver (met verschillende kosten en netwerkopzet) naast elkaar kunnen worden gelegd en vergeleken. Ook zal er door ons onderzoek gedaan worden welke informatie uit de Solver kan worden gehaald die nuttig is voor onderzoek.

5.3 Doelgroep

De interface die we zullen bouwen zal voldoen aan de doelen die we hierboven hebben gesteld. Het vergemakkelijken van de invoer en het duidelijk kunnen tonen van de uitkomst van de Solver zijn doelen met verschillende doelgroepen in gedachten. We gaan er vanuit dat de interface gebruikt gaat worden door mensen die niets afweten van de Solver zelf. Voor de invoer gaan we er vanuit dat de gebruiker wel veel afweet van het probleem zelf en dus weet wat hij moet invoeren (alle data behorende tot het probleem). We willen het daar de gebruiker vooral gemakkelijker maken.

De uitkomst van de Solver willen we overzichtelijk kunnen tonen zodat deze uitkomsten getoond kunnen worden aan personen / bedrijven die niets van de details van het probleem afweten. Hen moet, aan de hand van de uitvoer, gemakkelijk kunnen worden uitgelegd wat het probleem inhoudt en welke oplossingen het Twin Hub Network-Project in gedachten heeft. Welke aspecten van de uitvoer getoond worden wordt bestuurd door iemand die wel meer afweet van het probleem (eenzelfde persoon als die de data invoert).

6 Aanpak

In dit hoofdstuk bespreken we de uiteindelijke structuur van het project. Er wordt gekeken naar de software ontwikkelings methodiek, welke technieken er gebruikt worden, welke werkzaamheden er uitgevoerd moeten worden en de uiteindelijke planning van alle werkzaamheden.

6.1 Methodiek

Het ontwikkelproces is georganiseerd met de SCRUM methodiek in gedachten. Dit is een bekende vorm van een Agile Software Development framework. Hierbij worden verschillende rollen verdeeld binnen het team, het project opgedeeld in fasen en taken en regelmatig overlegd tussen de teamleden en de opdrachtgevers. Hier zullen wij grotendeels aan vasthouden.

Na elke fase vindt overleg plaats met de opdrachtgever. De ontwikkelfase is opgedeeld in meerdere iteraties met bijbehorende milestones. Elke milestone heeft doelen die in die iteratie bereikt moeten zijn. Als niet alle gedefinieerde doelen zijn bereikt schuiven deze door naar de volgende milestone.

6.2 Technieken

We zullen ons project baseren op de al bestaande code van Sebastiaan Meijer. Deze tool is gebouwd in Java met het Eclipse Rich Client Platform binnen de Eclipse ontwikkelomgeving. Hier zullen we ook gebruik van maken.

6.3 Werkzaamheden

De fasen van het project zien er als volgt uit:

- **Planning en oriëntatiefase**
  Hierin wordt het oriëntatieverslag geschreven, een gedetailleerde planning gemaakt, een plan van aanpak geschreven en een set requirements opgesteld.
- **Ontwerpfase**
  Hierin wordt eerst een ruw ontwerp gemaakt en besproken met de opdrachtgever. Daarna wordt het ontwerp verder uitgewerkt in UML en een definitieve requirements lijst opgesteld.
- **Ontwikkelfase**
  De ontwikkelfase is opgedeeld in drie milestones waarbij na elke milestone verschillende doelen behaald moeten zijn. In de gedetailleerde planning wordt dit verder uitgewerkt.
- **Verslag en presentatiefase**
  In deze fase wordt het product opgeleverd aan de opdrachtgever. Ook wordt de eindpresentatie gegeven en het eindverslag ingeleverd.

6.4 Planning

De planning voor het project ziet er als volgt uit:

Tabel 6.1: Planning opgedeeld in fasen

Week	17	18	19	20	21	22	23	24	25	26	27
Planning											
Oriëntatie											
Ontwerp											
Milestone 1											
Milestone 2											
Milestone 3											
Verslag											
Presentatie											

Week 17 - 18:
Planning en oriëntatie
- Planning maken
- Probleemstelling
- Requirements analyseren
- Ontwikkelomgeving bepalen
- Oriëntatieverslag maken
Deliverables: Planning, oriëntatieverslag

Week 17 - 19:
Ontwerpfase
- Globaal ontwerp
- Interface mockups
- Use cases
- Plan van Aanpak maken
- Requirements Analysis Document maken
Deliverables (week 18): Plan van aanpak, Requirements Analysis Document
Deliverables (week 19): Ontwerp, Use cases, Mockups

Week 19 - 21:
Eerste milestone
- Implementeer ontwerp in UML
- Identificeer alle onderdelen en verdeel ontwikkeltaken
- Start ontwikkeling
Deliverables: Gedetailleerd ontwerp in UML

Week 22 - 23:
Tweede milestone
- Ontwikkelen software prototype
Deliverables: prototype, SIG code opsturen

Week 24 - 25:
Derde milestone
- Feedback van SIG verwerken in code en conceptverslag
- Eerste concept software afronden

Deliverables: Conceptversie software

Week 26 - 27:
Afrondingsfase
- Uiteindelijke versie SIG code 5 dagen voor presentatie inleveren
- Conceptverslag maken en inleveren
- Feedback conceptverslag verwerken in eindverslag
- Eindpresentatie maken

Deliverables: Presentatie, conceptverslag (week 26), eindverslag (week 27), final versie software, final versie SIG code, final SIG feedback

7 Projectinrichting

In dit hoofdstuk wordt ingegaan op de betrokken personen bij dit project en de faciliteiten die gebruikt worden voor het voltooien van dit project.

7.1 Betrokkenen

Als opdrachtgever treedt Ekki Kreutzberger op namens OTB. Daarnaast is Cees Witteveen onze supervisor namens de TU Delft en ook de projectcoach samen met Michel Wilson. Gerd Gross is de projectcoördinator. Verder biedt Sebastiaan Meijer ondersteuning bij vragen over de solver en het algoritme.

7.2 Faciliteiten

We maken gebruik van gereserveerde meetingrooms beschikbaar op de faculteit EWI van de TU Delft. Verder beschikken alle groepsleden over een eigen laptop die gebruikt wordt voor de ontwikkeling. Alle verdere ontwikkelsoftware is opensource beschikbaar, behalve de Gurobi solver. Hiervoor is een academische licentie aangevraagd die beschikbaar is voor studenten.

8 Kwaliteitsborging

In dit hoofdstuk bespreken we kort hoe de kwaliteit van het product bewaakt wordt tijdens het project.

8.1 Kwaliteit

8.1.1 Ticket systeem

Om een overzicht te houden over alle issues die spelen binnen het project houden we tickets bij in het ticket systeem Mantis. Op deze manier worden er geen aspecten vergeten en is er een papertrail beschikbaar van alle verrichte werkzaamheden.

8.1.2 Continuous Integration

Het is belangrijk om tests regelmatig uit te voeren tijdens de ontwikkelingsfase zodat de kwaliteit van bestaande code gewaarborgd wordt. Het is daarom een goed idee om te proces te automatiseren. Hiervoor maken we gebruik van continuous integration. Continuous Integration (CI) is het proces waarbij alle code regelmatig wordt gecompileerd en getest op fouten.

Jenkins

Voor het automatiseren van het integratie proces maken we gebruik van de CI server Jenkins. Deze server haalt regelmatig alle code op uit de repository en compileert deze. Daarna worden alle unit tests uitgevoerd. Indien er tests falen, wordt de gebruiker automatisch op de hoogte gesteld van het probleem. Het voordeel hierbij is dat fouten in een vroeg stadium gedetecteerd worden voordat er meer code wordt geschreven die afhankelijk is van deze slecht functionerende programmatuur.

8.1.3 Checkstyle

Om de stijl van de code consequent te houden, maken we gebruik van de checkstyle plugin voor Eclipse. Deze houdt een vast aantal regels in de gaten m.b.t. code commentaar, naamgeving, whitespace, etc. Hiermee blijft de code netjes en leveren alle programmeurs gelijkwaardige code af.

8.2 Documentatie

Alle code wordt van commentaar voorzien. Hieruit wordt automatische documentatie gegenereerd worden die als referentie voor andere developers kan dienen. Daarnaast zal er voor de gebruiker een handleiding gemaakt worden. Hierin wordt uitgelegd hoe de software werkt en gebruikt kan worden.

8.3 Versiebeheer

Voor het beheren van alle code die geschreven wordt, maken we gebruik van versiebeheer software. Hierbij worden verschillende versies van de code bijgehouden in een repository en kunnen er makkelijke wijzigingen worden doorgevoerd. Er zijn verschillende oplossingen beschikbaar hiervoor maar wij hebben gekozen voor Git.

8.3.1 Git

Git is een distributed version control systeem, dit betekent dat de repository lokaal op de pc van de gebruiker is opgeslagen, alle code die wordt geschreven wordt ook in deze lokale repository opgeslagen. Wanneer de lokaal ontwikkelde code moet worden samengevoegd met de code van andere teamleden, wordt de code naar de remote repository gestuurd op het internet. Een van de belangrijke voordelen naast het decentrale karakter van Git is de verbeterde branch functionaliteit t.o.v. SVN. Dit is de voornaamste reden waarom wij van Git gebruik zullen maken. Zie [Cha12] voor meer informatie.

8.3.2 Assembla

Als remote git repository gebruiken we Assembla. Dit is een internet dienst die een grafische interface om een git repository biedt. Hiermee kan de repository worden beheerd en zaken als toegangscontrole worden ingesteld. We zullen onze applicatie parallel aan bestaande code ontwikkelen waar ook nog aan gewerkt wordt door anderen. Via dit systeem kunnen we veranderingen in de software over en weer doorvoeren. Door in verschillende Git branches te werken kunnen deze ontwikkeltrajecten naast elkaar plaatsvinden.

8.4 Testen

Als onderdeel van het ontwikkelproces zullen we verschillende tests uitvoeren op de software tijdens de ontwikkeling. Dit zorgt ervoor dat de kwaliteit van de software consistent blijft en er geen nieuwe bugs in bestaande code ontstaan.

8.4.1 Test driven development

Het is een wens vanuit de opdrachtgever om tests te integreren in de software. Als na afloop van het project verder wordt ontwikkeld aan de software, kunnen de tests waarborgen dat er geen nieuwe bugs worden geïntroduceerd in bestaande code. Om dit te faciliteren zullen wij tijdens dit project zoveel mogelijk gebruik maken van test-driven development. Hierbij worden eerst tests geschreven voordat de eigenlijke code wordt geschreven. Zo wordt gegarandeerd dat elk stuk code getest wordt.

8.4.2 Unit testing

Voor het testen van de code zullen we gebruik maken van unit tests. Met deze testen wordt elk stuk code getest op de juiste werking. Door voor elk stuk code een test te schrijven, kan op een structurele manier gecontroleerd worden of de software nog steeds aan de eisen voldoet.

JUnit

Als framework voor unit tests maken we gebruik van JUnit. Dit framework is zeer geschikt voor het testen van Java applicaties en heeft goede integratie in onze ontwikkelomgeving Eclipse.

8.4.3 Test coverage

Om de effectiviteit van de unit tests te beoordelen kunnen we de code coverage van de tests controleren. Dit doen we met de Corbertura plugin voor onze Jenkins server. Corbertura is een tool voor het analyseren van code coverage en hiermee kunnen we code identificeren die nog niet getest is. Omdat deze tool geïntegreerd is in Jenkins, zal deze analyse regelmatig worden uitgevoerd tijdens ontwikkeling.

8.5 Evaluatie

Iedere week wordt de voortgang kort besproken met Michel Wilson, hierin bekijken we of onze implementatie nog in lijn ligt met de wensen van de opdrachtgever. Daarnaast zijn er drie milestone momenten gepland waarin we overleggen met de coach en opdrachtgever over de voortgang van het project en ontwerpen/prototypes presenteren.

8.5.1 SIG

De code zal worden aangeboden aan de Software Improvement Group (SIG) ter evaluatie. Zij zullen tweemaal feedback geven, tijdens en na afloop van de ontwikkeling. De eerste feedback zal door ons verwerkt worden in de code en het eindverslag.

Bibliografie

[Cha12] S. Chacon. Why You Should Switch from Subversion to Git. `http://blog.teamtreehouse.com/why-you-should-switch-from-subversion-to-git`, 2012. [Online; bezocht 3-Mei-2013].

[Mei12] S. Meijer. Solving the twin hub bundling problem. Master's thesis, Delft University of Technology, 2012.

[Twi] Twin Hub Network. `http://www.twinhubnetwork.eu/`. [Online; bezocht 3-Mei-2013].

A Overzicht Output THB-Solver

Overzicht van output uit huidige solver tool.

A.1 Algemene informatie over de oplossing

- Instantienaam
- Aantal Sources
- Aantal Destinations
- Maximum / Totaal aantal Batches
- Aantal O / D flows
- Aantal Containers
- Totale Kosten van de oplossing (ook per container)

A.2 Oplossingsstatistieken

Modal Split:

- Containers per batch trein (min/mean/max/stdev)
- Max / min verschil per batch (min /mean/max/stdev)
- (off)loaded containers per batch (min /mean/max/stdev)
- (in)directe trein containers (min /mean/max/stdev)
- Hub trein containers (min /mean/max/stdev)
- Destination trein containers (min /mean/max/stdev)

Gedetailleerde algemene informatie per batch:

- Batch kosten (ook per container)
- Aantal containers
- Containers per trein
- (off)loaded containers per trein

Gedetailleerde informatie per trein / truck per batch:

- Begin- en eindlocatie
- Aantal containers per trein / truck

Totale kosten voor dit vervoer met specificaties:

- Door – Hub (hub trein) = Prehaulage + Vervoerskosten (Transport + Terminal Costs)
- Hub – Door (dest. trein) = Vervoerskosten (Transport + Terminal Costs) + Posthaulage

C Requirements Document

TI3800 Bachelorproject

Twin-Hub Network Project

Requirementsdocument

4008847 Marlou Pors
1170902 Jonathan Tetteroo
1308378 Carlo van der Valk

19 juni 2013

Inhoudsopgave

1 Introductie **3**

2 Requirements **4**
2.1 Must Have . . . 4
2.2 Should Have . . . 5
2.3 Could Have . . . 5
2.4 Won't Have . . . 5

3 Bekeken oplossingen **6**
3.1 Overzicht problemen en oplossingen . . . 6
3.2 Uitleg problemen en oplossingen . . . 6
3.2.1 Probleem 1: Het 'stappenplan' van de invoer . . . 6
3.2.2 Probleem 2: Het combineren van de matrices . . . 8
3.2.3 Probleem 3: De weergave van de probleeminstantie . . . 9
3.2.4 Probleem 4: De weergave van gedetailleerde informatie . . . 9
3.2.5 Probleem 5: Keuzemogelijkheden rondom verschillende soort informatie . . . 11

4 Globale Opzet **13**
4.1 Invoer . . . 13
4.1.1 Tabbladen . . . 13
4.1.2 Matrix . . . 13
4.1.3 Solver parameters . . . 13
4.1.4 Controle/Netwerk . . . 14
4.1.5 Samenvatting Invoer . . . 14
4.1.6 Run Knop . . . 14
4.2 Proces . . . 14
4.2.1 Grafiek . . . 14
4.2.2 Solver parameters . . . 15
4.2.3 Tekst output . . . 15
4.2.4 Stop knop . . . 15
4.2.5 Andere oplossingen weergeven . . . 15
4.3 Uitvoer . . . 15
4.3.1 De verdeling van het scherm . . . 15
4.3.2 Netwerkweergave . . . 16
4.3.3 Kostenweergave . . . 17
4.3.4 Statistieken . . . 17
4.4 Matrix . . . 17
4.5 Distance matrix . . . 17
4.5.1 Vaste distance matrix binnen de tool . . . 17
4.5.2 Geografische informatie . . . 17
4.5.3 Import van distance matrix . . . 18
4.6 Flow matrix . . . 18
4.6.1 Inputtabel in de tool . . . 18
4.6.2 Import van flow matrix . . . 18

1 Introductie

Dit verslag is, net als het Oriëntatieverslag en Plan van Aanpak, een resultaat van de onderzoeksperiode van het Bacheloreindproject. In dit verslag gaan we in op de basis die onze applicatie nodig heeft.

In dit verslag zullen we de lijst van Requirements presenteren en zetten we uit een welke problemen we de afgelopen periode hebben onderzocht. Dit zijn problemen die enerzijds te maken hebben met hoe we willen dat de interface er voor de gebruiker uit zal zien, anderzijds met keuzes die we moeten maken om het achter de schermen ook te stroomlijnen. Grootste onderdeel hierin is het gebruikersvriendelijk maken van de invoer van grote matrices. Dit willen we vergemakkelijken voor de gebruiker, terwijl de Solver nog steeds dezelfde informatie ontvangt. Wij implementeren namelijk alleen een interface die als schil rond de Solver werkt. Naast ons onderzoek geven we een globale opzet die ons haalbaar en goed lijkt voor het eindproduct. Hierin zullen we het hebben over de invoer, het scherm tijdens de berekening en de uitvoer; Echter blijven we ook met vragen zitten, die we ook zullen stellen.

Dit document reflecteert hoe wij de huidige tool willen verbeteren en welke wensen wij denken dat de opdrachtgever heeft. Aan de hand van de reactie van de opdrachtgever kunnen de requirements en opzet nog veranderen.

2 Requirements

Om de eisen en prioriteiten van onze software te definiëren zullen wij de MoSCoW methode gebruiken. Bij de MoSCoW methode geven we aan wat onze software moet hebben (Must Have), hoort te hebben(Should Have), kan hebben(Could Have) en niet zal hebben(Won't Have). Bij de Must Have zullen we ook een duidelijke onderscheiding tussen de invoer, processing en uitvoer maken.

2.1 Must Have

1 **Invoer**

- De gebruiker moet op een gebruiksvriendelijke manier gegevens kunnen invoeren die de Solver zal gaan gebruiken.
- De parameters van het algoritme kunnen worden aangepast zoals nu dat ook kan in de interface rondom de Solver. Hieronder vallen ook de Solver Parameters en eventuele MIP start bestand.
- De Gebruiker moet op een gebruiksvriendelijkere manier de distance en flow matrix kunnen invoeren.
- Alle plaatsen die in het Twin Hub Netwerk Project gebruikt worden zullen in het systeem gedefinieerd zijn. Vanuit deze set locaties kan de gebruiker kiezen welke hij wil gebruiken voor zijn (sub)probleem.
- Voor het transportverkeer geldt: De flows kunnen worden ingevoerd binnen de tool of er wordt een flow matrix geïmporteerd.
- Voordat de Solver aangezet wordt, moet er een Hub worden geselecteerd (Rotterdam of Antwerpen).
- Er moet een nieuwe locatie worden ingevoerd en toegevoegd aan de collectie locaties.
- Locaties die al in het systeem staan moeten kunnen worden gewijzigd of verwijderd.

2 **Processing**

- De gebruiker moet de Solver kunnen stoppen en vervolgens de laatste oplossing kunnen bekijken.
- Tijdens de uitvoer van het algoritme, wordt er een grafiek weergegeven van de kosten van gevonden oplossingen.

3 **Uitvoer**

- De uitvoer kan weergegeven worden als een graaf met een logische ordening.
- De gebruiker kan vanuit een graaf de verdeling van het transport figuurlijk aflezen.
- De weergave van de oplossing kan worden opgesplitst in verschillende batches, directe treinen en directe trucks.
- Naast de grafische weergave is er een tekstweergave met een overzicht van de oplossing. Als voorbeeld is dat de batch- of algemene informatie.
- Naast de grafische weergave is er een gedetailleerd kostenoverzicht.
- De Kostenweergave wordt afgerond. Bij de huidige tool worden tot vijf decimalen achter de comma weergegeven.

– Het moet voor de gebruiker mogelijk zijn ook andere oplossingen te bekijken. Standaard wordt de optimale oplossing weergegeven. De huidige tool geeft meerdere oplossingen en wijst de best gevonden oplossing aan.

2.2 Should Have

– De totale invoer moet kunnen worden gecontroleerd door de gebruiker voordat de Solver te werk gaat.
– De invoer moet halverwege kunnen worden opgeslagen als een bestand, zodat de gebruiker deze later weer kan inladen en verder kan gaan.
– Als de solver tijdens verwerken is gestopt, moet de gebruiker deze kunnen herstarten.
– De uitvoerweergave is in vorm van een graaf met een geografische ordening.

2.3 Could Have

– De uitvoerweergave is in vorm van een graaf met geografische ordening en een geografische kaart op de achtergrond.
– De gebruiker moet de mogelijkheid hebben om een fictief netwerk te construeren.
– Verschillende oplossingen zijn te vergelijken doordat de verschillen in de netwerken als een graaf worden weergegeven.
– De Statistische informatie wordt in een grafiek weergegeven.
– De gebruiker moet de mogelijkheid hebben het transport te laten afspelen in de weergave. Er wordt een visuele presentatie gegeven om te construeren wat er bij de oplossing gebeurt.
– De oplossing kan ge-exporteerd worden naar een bruikbaar bestand, bv. Excel.
– Er is ondersteuning voor meerdere talen.

2.4 Won't Have

– De gebruiker kan geen distance matrix invoeren in de tool en kan deze ook niet vanuit een bestand importeren. Dit omdat er gewerkt wordt met een vast netwerk.
– Tijdens de werking van de Solver kan er geen weergave gedaan worden van de tot nu toe gevonden (deel) oplossingen.
– Er kan geen 'Round Trip' vraag gesteld worden. Dit is niet in de huidige Solver geïmplementeerd.

3 Bekeken oplossingen

Voordat we onze globale opzet voor de applicatie presenteren, zetten we de grootste problemen op een rijtje. Tijdens ons onderzoek zijn we vijf grote problemen tegengekomen. Sommigen zijn ook gerelateerd aan ontwerpkeuzes die we hebben gemaakt na onze eerste gesprekken met de opdrachtgever. In dit hoofdstuk zullen we eerst in tabelvorm de problemen en oplossingen presenteren en vervolgens per probleem uitleggen wat het inhoudt en wat de door ons gevonden oplossingen zijn. Sommige oplossingen zullen we illustreren met een plaatje.

3.1 Overzicht problemen en oplossingen

Hieronder staat het overzicht van de door ons onderzochte problemen en een korte omschrijving van de gevonden oplossingen. Dit is de zogenaamde 'Morfologische' kaart.

	Oplossing 1	Oplossing 2	Oplossing 3	Oplossing 4	Oplossing 5
Het weergeven van het 'stappenplan' voor de invoer van het probleem	Een wizard	Een systeem met tabbladen	Een systeem met een grote lijst		
Het combineren van de distance / flow matrix	De Flow matrix aanvullen met nullen	Een Subset selecteren van de distance matrix	Een mapping zodat alle locatienamen kloppen	Een graaf-editor om flows in te voeren op een kaart	
De weergave van de probleeminstantie	Graaf	Geografische kaart			
De weergave van de gedetailleerde informatie van de oplossing	Informatie direct langs de edges van de graaf	De Informatie in een tabelbuiten de graaf	Naast locaties beschijving geven.	Figuurlijke uitbeelding d.m.v. blokken	Details tonen en verbergen d.m.v. klikken
De keuzemogelijkheden voor het tonen van de verschillende informatie	Een systeem met tabbladen	Een systeem met dropdown boxes	Losse vensters		

Figuur 3.1: de Morfologische kaart

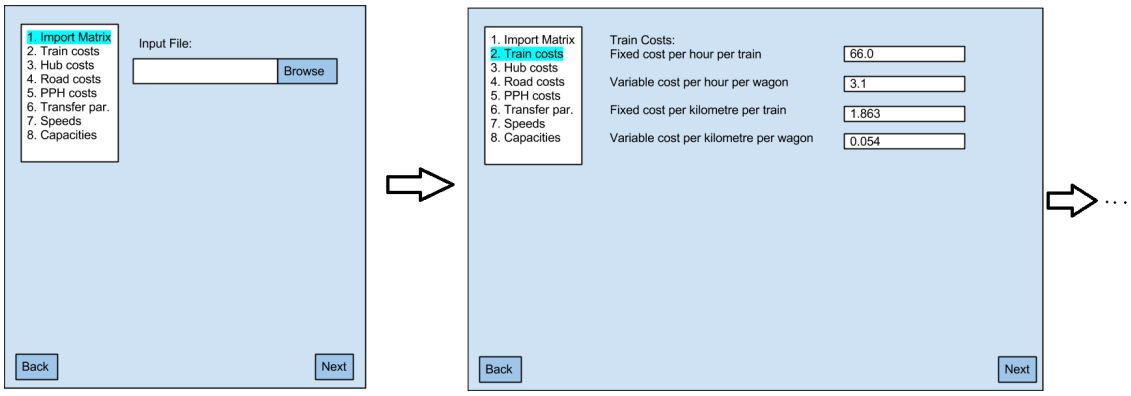
3.2 Uitleg problemen en oplossingen

3.2.1 Probleem 1: Het 'stappenplan' van de invoer

We willen dat de invoer van de benodigde informatie voor de Solver gebruiksvriendelijker gaat. Hierbij doelen we voornamelijk op de invoer van de verschillende matrices. Op dit moment moet je de Solver een voorafgemaakte flow en distance matrix voeren, die je binnen de interface ook niet kan aanpassen / controleren. Dit aspect willen binnen de interface halen. Het aanpassen van de verschillende parameters is best duidelijk in de huidige interface van de Solver. Hieronder staan oplossingen op welke manier we de invoer willen presenteren aan de gebruiker.

Oplossing 1: Een Wizard

We kunnen de gebruiker een wizard laten volgen om er zo voor te zorgen dat de gebruiker volgens een strikt stappenplan alle benodigde informatie kan invoeren.
Voordelen: Met een Wizard komen er verschillende schermen voor bv. de invoer van de flow matrix en de invoer van alle Solver parameters, zodat deze onderdelen duidelijk gescheiden worden.

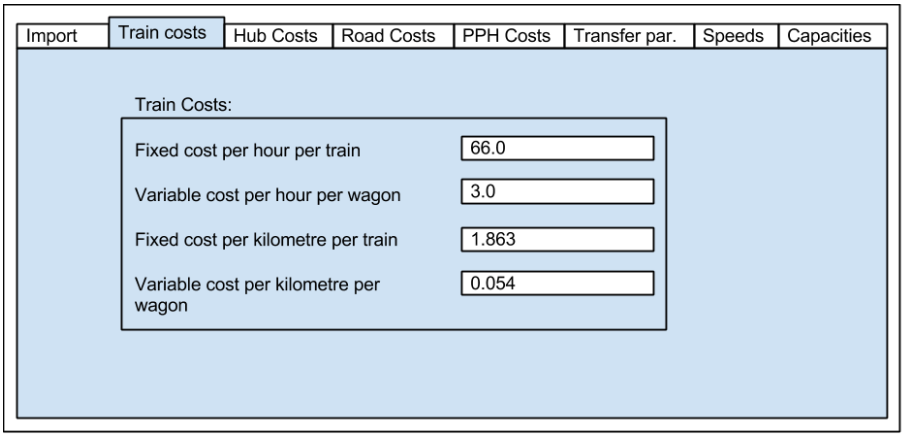


Figuur 3.2: De invoer m.b.v. een Wizard

Zo kan ook gemakkelijk gekozen worden met welke hub wordt gewerkt (requirement Hub Selectie).
Nadelen: In een Wizard kan de gebruiker niet gemakkelijk schakelen tussen de verschillende aspecten van de invoer als hij hier en daar iets is vergeten, of snel onderdelen wil overslaan omdat hierbij de standaard settings voldoende zijn. Daarnaast is de implementatie van een Wizard veel werk.

Oplossing 2: Een systeem met tabbladen

We kunnen de gebruiker een invoer met verschillende tabbladen geven, zodat hij zelf kan kiezen welke onderdelen hij wanneer wil aanpassen.



Figuur 3.3: De invoer m.b.v. een systeem met tabbladen

Voordelen: Ook op deze manier worden de verschillende aspecten van de invoer duidelijk van elkaar gescheiden. Daarnaast kan de gebruiker gemakkelijk wisselen tussen de onderdelen van de invoer die hij wil aanpassen. De implementatie van deze oplossing is ook gemakkelijker.
Nadelen: Het werken met tabbladen is niet voor iedereen vanzelfsprekend. Daarnaast moeten verschillende stappen van een bepaalde invoer op een tabblad worden samengevoegd, waardoor het minder duidelijk kan worden.

Oplossing 3: Een grote lijst

In de basis kunnen alle onderdelen van de invoer die aangepast kunnen worden onder elkaar geplakt worden (eventueel aan en uit te vinken waardoor de lijst groter dan wel kleiner wordt) en kan de

gebruiker hier doorheen scrollen om alles in te voeren.
Voordelen: De gebruiker heeft alle onderdelen op één pagina en kan gemakkelijk in te voeren onderdelen zichtbaar of onzichtbaar maken. De implementatie van deze oplossing is zeer gemakkelijk.
Nadelen: Alle informatie op één pagina wordt getoond kan dit snel onoverzichtelijk worden. Daarnaast zal de gebruiker veel moeten scrollen om bij de gewenste onderdelen te komen.

3.2.2 Probleem 2: Het combineren van de matrices

De Solver van het THB Probleem heeft een zogenaamde distance en flow matrix nodig als invoer om de oplossing te berekenen. Om het de gebruiker makkelijker te maken en omdat we hebben besloten met een vaste wereld te werken (waarin afstanden niet veranderen), willen we het invoeren van de distance matrix laten wegvallen bij de gebruiker. Echter, om aan de juiste invoer te komen voor de Solver, moet hiervoor een ander soort van invoer in de plaats komen.

Hoe dit probleem wordt opgelost is niet (direct) zichtbaar / belangrijk voor de gebruiker. Er wordt gedacht het invullen van het probleem op twee manieren te presenteren, d.m.v. een matrix (oplossing 1 t/m 3) of d.m.v. een graaf (oplossing 4). Er wordt nog niet ingegaan hoe de in te vullen matrix aan de gebruiker wordt gepresenteerd.

Oplossing 1: De Flow matrix aanvullen met nullen

Het achterliggende idee van deze oplossing is dat d.m.v. de invoer van een aangepaste matrix (met hierin de namen van de locaties meegenomen) en de flow hiertussen, de benodigde flow en distance matrix kan worden aangepast zodat deze klopt voor de invoer van de Solver. Bij deze oplossing wordt er gewerkt met een distance matrix die álle bestaande bestemmingen bevat en die dus nooit aangepast hoeft te worden. Na invoer van de aangepaste matrix wordt de flow matrix aangevuld met nullen tussen de locaties waarvoor geen flow voor is ingevoerd.
Voordelen: De distance matrix hoeft nooit te worden aangepast omdat we werken met een distance matrix die alle bestemmingen bevat. Het aanvullen van de flow matrix met nullen is gemakkelijk te implementeren.
Nadelen: De flow matrix kan aangevuld met nullen erg groot zijn, waardoor het probleem dat aan de Solver wordt gevoerd groter lijkt dan het in werkelijkheid is. Dit kan ervoor zorgen dat de berekening langer duurt. Daarnaast moet er rekening gehouden worden met mapping (Oplossing 3).

Oplossing 2: Een subset selecteren van de Distance matrix

Ook bij deze oplossing wordt met bovenstaande uitgelegde aanname gewerkt. Echter, na invoer van de aangepaste matrix wordt deze matrix de flow matrix (die hoeft niet meer worden aangepast na de invoer) en wordt er op basis van de ingevulde locatienamen een subset gepakt van de totale distance matrix met alle bestaande bestemmingen.
Voordelen: De flow matrix volgt direct uit wat is ingevuld en hoeft niet verder worden aangepast. Het totale probleem dat aan de Solver wordt gevoerd is kleiner dan bij de vorige oplossing.
Nadelen: Het berekenen van een subset is minder gemakkelijk te implementeren dan het aanvullen van nullen. Daarnaast moet er rekening gehouden worden met mapping (Oplossing 3).

Oplossing 3: Een mapping tussen de twee matrices

Er moet opgelet blijven worden dat, wanneer de gebruiker zelf de flow matrix samenstelt - of deze importeert vanuit Excel - dat de kans groot is dat de volgorde van de locatienamen hierin kunnen verschillen met de volgorde van de locaties in de distance matrix. In de combinatie distance- flow matrices (waarin alleen de locatienamen worden meegegeven in de distance matrix) moeten deze wel overeenkomen. Hiervoor moet dus een mapping worden bedacht. Een voorbeeld hiervan is de locaties te sorteren op alfabetische volgorde.

Oplossing 4: Flows invoeren met een graaf-editor

Er kan ook voor gekozen worden dat er geen invoer in matrices wordt gedaan, maar dat de gebruiker een graaf voorgeschoteld krijgt waar hij op de edges kan aangeven wat de flow is tussen de twee locaties. De afstand tussen deze locaties is dan achter de schermen opgeslagen. Na de invoer van alle flows wordt dan vanuit de graaf een distance en flow matrix samengesteld.
Voordelen: Bij deze oplossing zal er geen rekening gehouden hoeven te worden met mapping (Oplossing 3), omdat elke keer een flow matrix en distance matrix worden aangemaakt. Het totaalplaatje van het netwerk kan overzichtelijk werken.
Nadelen: Het aanmaken van een flow en distance matrix kan tijd kosten en is niet direct makkelijk te implementeren. Indien het netwerk groot is, is het invoeren langs de kanten wellicht niet meer overzichtelijk. De gebruiker heeft waarschijnlijk de flow al paraat in een lijst, waarna het 'kriskras' moeten invullen van flows onhandig / langdradig kan worden.

3.2.3 Probleem 3: De weergave van de probleeminstantie

Het doel is de gebruiker zo overzichtelijk mogelijk te laten zien wat het Twin Hub Netwerk Project inhoud. Het netwerk zelf moet dus zo duidelijk mogelijk weergegeven worden. Hoewel we uitgaan van een graaf, zijn er verschillende manieren hoe we deze graaf kunnen presenteren. Dat wordt hieronder besproken.

Oplossing 1: Slechts een graaf

Het netwerk kan heel gemakkelijk in vorm van een graaf worden weergegeven. De nodes zijn de locaties / gebieden waar de containers vandaan komen of naartoe moeten en de edges zijn de verbindingen tussen de locaties. De lengte van de edges geeft de afstand weer om het netwerk overzichtelijk en realistisch te maken. Voor het duidelijkste, meest realistische resultaat staat een node die Rotterdam voorstelt ook rechts van een node die Manchester voorstelt. Op deze manier kan de graaf gekoppeld worden aan een geografische kaart.
Voordelen: Overzichtelijke oplossing. Aan de locaties, in vorm van nodes, kan veel informatie worden meegegeven in de implementatie.
Nadelen: Misschien niet realistisch genoeg. Daarnaast kan het netwerk erg groot worden. Zonder te werken met coördinaten is het lastig om de relatieve ligging van de locaties te bepalen.

Oplossing 2: Een graaf i.c.m. een geografische kaart

De locaties die verbonden zijn met het THN zijn te vinden op een geografische kaart. Immers, dit is een realistisch project. Deze locaties kunnen direct van een geografische kaart gepakt worden, met ook realistische verbindingen tussen verschillende locaties, maar het is meer gewenst dat de locaties op voorgrond van een geografische kaart kunnen worden getoond om de graaf realistischer te laten lijken.
Voordelen: Ook in deze oplossing kunnen er rechte verbindingen worden getrokken tussen de locaties. Het maakt de oplossing realistischer.
Nadelen: Er moet gewerkt worden met coördinaten van de verschillende locaties. Het netwerk is erg groot, waardoor het onoverzichtelijk kan overslaan. Er moet dan nagedacht worden over in- en uitzoomen dat ook plaatsgebonden is. Dit is ingewikkeld.

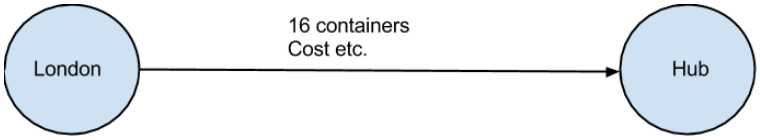
3.2.4 Probleem 4: De weergave van gedetailleerde informatie

Uit de solver komen oplossingen met aanzienlijk hoeveelheid van informatie die we overzichtelijk en duidelijk willen weergeven. De uitvoer zal worden weergeven in de vorm van een of meerdere grafen en/of tabellen. Deze opstellingen geven op zich geen duidelijk beeld van de uitvoer. De volgende oplossingen zijn mogelijkheden om de details van de oplossing (kostenplaatje, containeraantallen etc.) duidelijk te weergeven voor de gebruiker. In de huidige tool worden de details altijd weergeven

in een tabel. Daarnaast wordt er nergens een beeld gegeven van het netwerk. De weergave hiervan werd in het vorige probleem onderzocht.

Oplossing 1: Informatie langs de edges van de graaf

De uitvoer wordt als een graaf weergeven waarbij de belangrijke informatie langs de edges wordt weergegeven. Dit kan in de vorm van tekst, maar ook kunnen kleine pictogrammen worden gebruikt bij de edges om het type verkeer weer te geven, bijvoorbeeld een klein trein of truck.



Figuur 3.4: Informatie langs de edges van de graaf

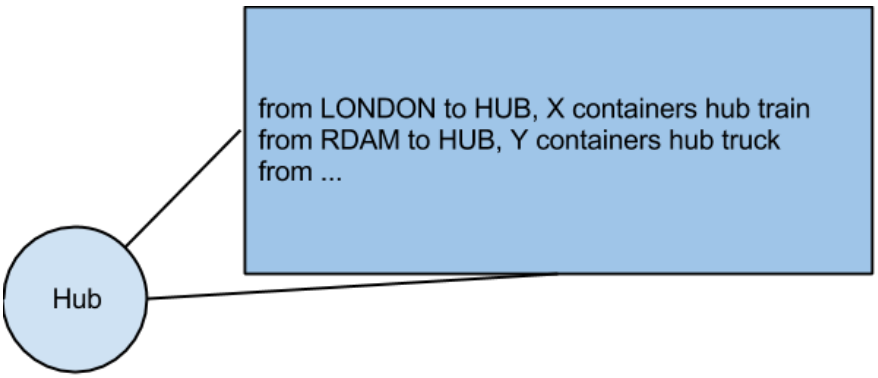
Voordelen: Dit schetst voor de gebruiker een beeld van trein en truck vervoer tussen de locaties.
Nadelen: Bij grote netwerken met veel edges wordt het minder overzichtelijk om bij edges details weer te geven. Bij dicht op elkaar liggende edges kan tekstoverlap ontstaan bijvoorbeeld.

Oplossing 2: Informatie in een tabel buiten de graaf

Alle informatie die de Solver geeft van de oplossing wordt in een tabel getoond om een overzicht te geven van containervervoer en -kosten. Dit is de weergave die de bestaande tool al gebruikt.
Voordelen: Informatie in tabelvorm maakt vergelijken van uitvoer makkelijker.
Nadelen: Een tabel geeft minder duidelijk beeld van de uitvoer (het netwerk) in zijn geheel.

Oplossing 3: Informatie naast de nodes in de graaf

Een gedetailleerde beschrijving wat er per locatie gebeurd kunnen we naast de locatie geven, zoals de beschrijving van het aantal containers dat van en naar de geselecteerde locatie gaat. Dit resulteert in een samenvatting van alle informatie op de edges die verbonden zijn met deze locatie.

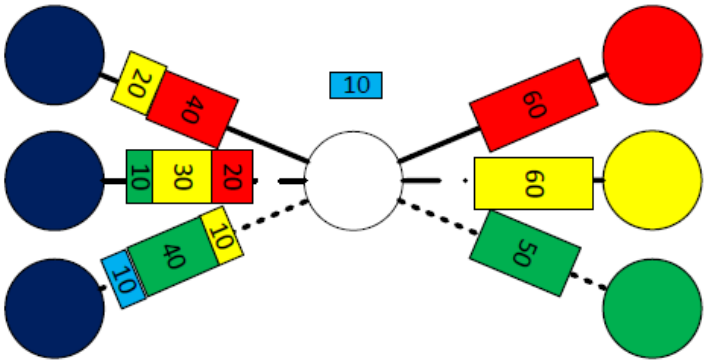


Figuur 3.5: Informatie naast de nodes in de graaf

Voordelen: Details zo tonen geeft de gebruiker een idee van het verkeer op een locatie.
Nadelen: Bij drukke locaties is het teveel informatie voor deze weergave.

Oplossing 4: Figuurlijke uitbeelding

Het transport van en tussen de node kan grafisch en met kleur weergegeven worden met blokken om het transport en de container kwantiteit te representeren. Dit is een meer grafische weergave van het verkeer ten opzichte van de eerdere oplossingen.



Figuur 3.6: Grafische weergave van het verkeer

Voordelen: Hiermee kan gevolgd worden hoe het verkeer zich over de netwerk verdeelt.
Nadelen: Kosten en type vervoer worden niet getoond; hiervoor is dan een aanvullende weergave nodig.

Oplossing 5: Click and Show

Een graaf van de oplossing wordt gegeven zonder alle gedetailleerde informatie waar de gebruiker door middel van klikken op een node of edge details van kan oproepen. De informatie kan op alle boven beschreven manieren worden getoond.
Voordelen: De gebruiker kiest welke informatie te tonen, zodat ze zich kunnen bezighouden met relevante informatie.
Nadelen: De gebruiker moet op zoek gaan naar de details door informatie zichtbaar te maken.

3.2.5 Probleem 5: Keuzemogelijkheden rondom verschillende soort informatie

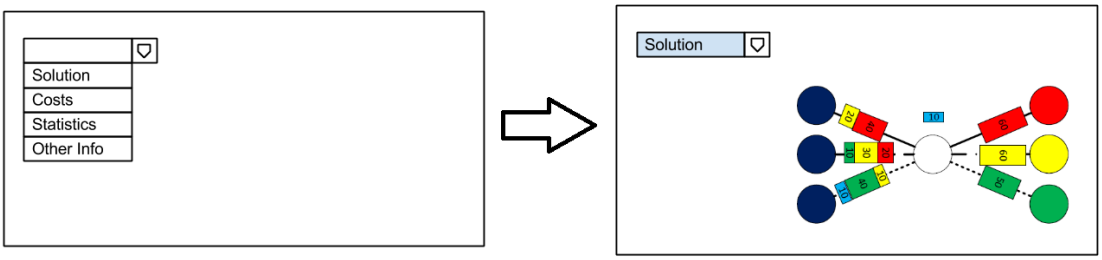
Alle informatie over de uitvoer moet voor de gebruiker bereikbaar zijn maar moet niet op tegelijkertijd op het scherm getoond worden. Dit zorgt ervoor dat de gebruiker alle nodige details ziet maar niet overspoeld wordt met informatie. De volgende oplossingen zijn manieren om verschillende informatie te tonen op het scherm.

Oplossing 1: Werken met tabbladen

Bij deze informatie wordt de informatie verdeeld over verschillende tabbladen en kan de gebruiker kiezen welke soort informatie wil bekijken.
Voordelen: De gebruiker kan snel relevante informatie vinden door het tabblad te kiezen met de informatie die ze zoeken. Ook kan per soort informatie een geschikte weergave gekozen worden zonder dat het een grote pagina met verschillende weergaven wordt.
Nadelen: De gebruiker kan minder makkelijk verschillende informatie vergelijken. Bijvoorbeeld, de gebruiker kan grafiek en kosten niet tegelijkertijd bekijken.

Oplossing 2: Dropdown Boxes

De informatie kan worden opgevraagd door middel van dropdown boxes waarin een selectie gemaakt kan worden uit verschillende aandachtspunten van de uitvoer.

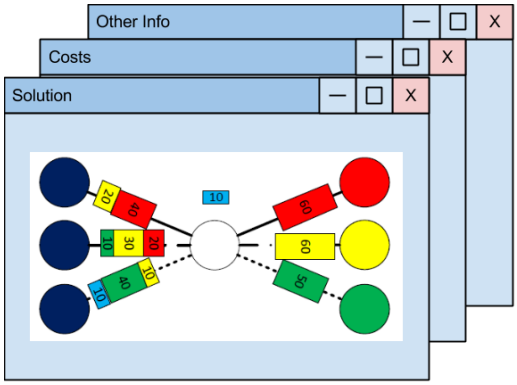


Figuur 3.7: Dropdown Boxes

Voordelen: De gebruiker kan snel relevante informatie vinden door een keuze te maken in de dropdown box.
Nadelen: Het vergelijken van informatie wordt moeilijker voor de gebruiker. En het wisselen tussen dropdown box selecties is minder handig dan tabbladen.

Oplossing 3: Losse vensters

De informatie wordt verdeeld onder verschillende vensters.



Figuur 3.8: Losse vensters

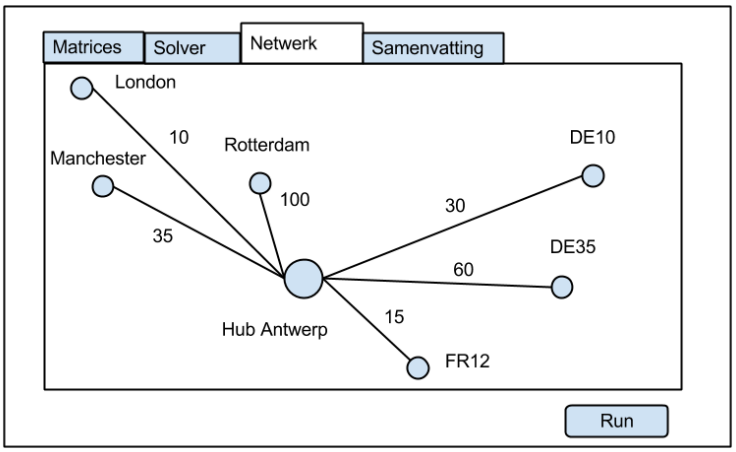
Voordelen: Verschillende informatie kan tegelijkertijd worden bekeken op het scherm.
Nadelen: Bij een oplossing met veel informatie zullen niet alle windows tegelijkertijd in hun geheel op de scherm kunnen worden getoond en venster elkaar overlappen.

4 Globale Opzet

Dit is de globale opzet van de user interface. We hebben het opgesplitst in vier onderdelen.

4.1 Invoer

Het eerste deel van het programma bestaat uit de invoer. In dit scherm heeft de gebruiker de mogelijkheid van alles aan te passen en in te voeren voordat de Solver aan het werk wordt gezet. Onderstaande geeft onze keuzes weer rondom de problemen 1,2 en 3 van het vorige hoofdstuk.



Figuur 4.1: Schets van invoer netwerk met flows

4.1.1 Tabbladen

Wij hebben gekozen om de invoer te verdelen over verschillende tabbladen. De verschillende tabbladen zullen zijn: Het invoeren van de matrix, het invoeren van de Solver parameters en de controle van de invoer in vorm van het Netwerk. Naast deze tabbladen zal er op het schermen verschillende informatie en knoppen altijd zichtbaar zijn voor de gebruiker.

4.1.2 Matrix

Het eerste tabblad is voor de invoer van de flow matrix. Als eerste moet de hub gekozen worden (Rotterdam of Antwerpen). Daarna kan de gebruiker kiezen om een matrix uit een bestand te importeren of een matrix handmatig in te voeren in een tabel binnen de applicatie. Ook kan een matrix worden geïmporteerd en vervolgens worden bewerkt binnen de applicatie. Verder kan de gebruiker een ingevoerde / bijgewerkte matrix weer opslaan als een bestand.

Zie hiervoor ook de paragraaf over de matrix (4.4).

4.1.3 Solver parameters

In het tweede tabblad kunnen de parameters voor de randvoorwaarden van het probleem worden ingevoerd, zoals het kostenmodel, snelheid en capaciteiten van de hub. Deze zullen ingevoerd kunnen worden zoals bij de bestaande interface al kan.

4.1.4 Controle/Netwerk

In het laatste tabblad zal een graaf met alle relevante informatie worden getoond. Deze visuele representatie geeft de gebruiker wat inzicht over het probleem en kan worden gebruikt als controlemiddel.

4.1.5 Samenvatting Invoer

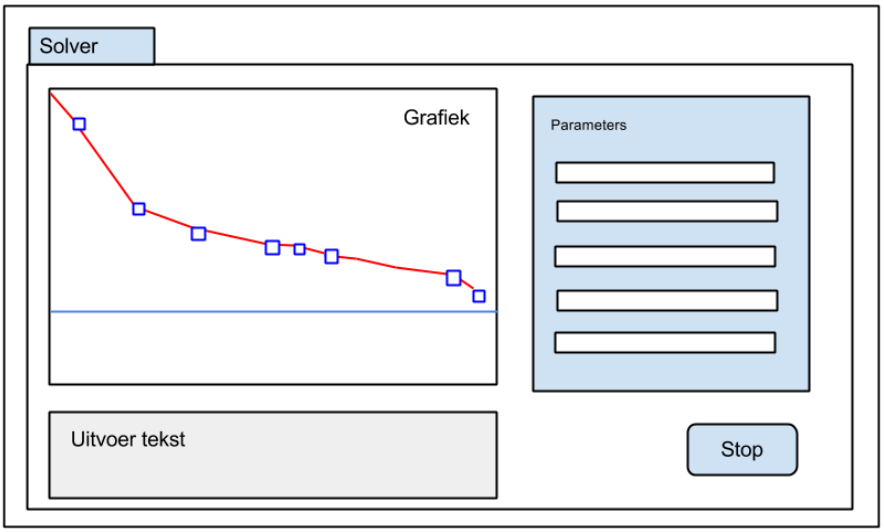
Onder en buiten de tabbladen is er een ruimte waar een samenvatting van de probleeminstantie wordt gegeven. Op deze manier heeft de gebruiker altijd inzicht over de algemene punten van zijn invoer. Voorbeelden van deze punten zijn welke hub er is gekozen, welk kostenmodel wordt nageleefd en hoe het bestand heet dat als flow marix wordt gebruikt.

4.1.6 Run Knop

De Run knop zal zich ook buiten de tabbladen bevinden zodat de gebruiker de applicatie op elk moment kan laten starten met verwerken van de probleeminstantie.

4.2 Proces

Tijdens het runnen van de solver worden verschillende gegevens weergegeven die de gebruiker informeren over de voortgang van de solver. Hier bespreken we hoe de interface er voor deze gegevens uit zal zien.



Figuur 4.2: Schets van het processcherm

4.2.1 Grafiek

Om de voortgang van de solver duidelijk weer te geven worden de kosten van gevonden oplossingen geplot in een grafiek in de interface. Ook de lower bound wordt geplot, waardoor het eenvoudig te zien is hoe de gevonden oplossingen zich hiermee verhouden.

4.2.2 Solver parameters

De diverse parameters die zijn ingesteld voor de solver zullen zichtbaar zijn, hieronder valt het algoritme type, instantie naam, maximum runtijd, huidige runtijd en gevonden/lower bound kosten.

4.2.3 Tekst output

Naast de grafiek zal de tekstuele output van de solver ook zichtbaar zijn zoals op dit moment ook het geval is. Hierin worden o.a. de best gevonden kosten, runtijd en verschil met lower bound aangegeven.

4.2.4 Stop knop

Het solven kan tussentijds gestopt worden met de stop knop. Later kan de gebruiker verder gaan met de laatst gevonden oplossing, zodat er niet helemaal opnieuw begonnen hoeft te worden.

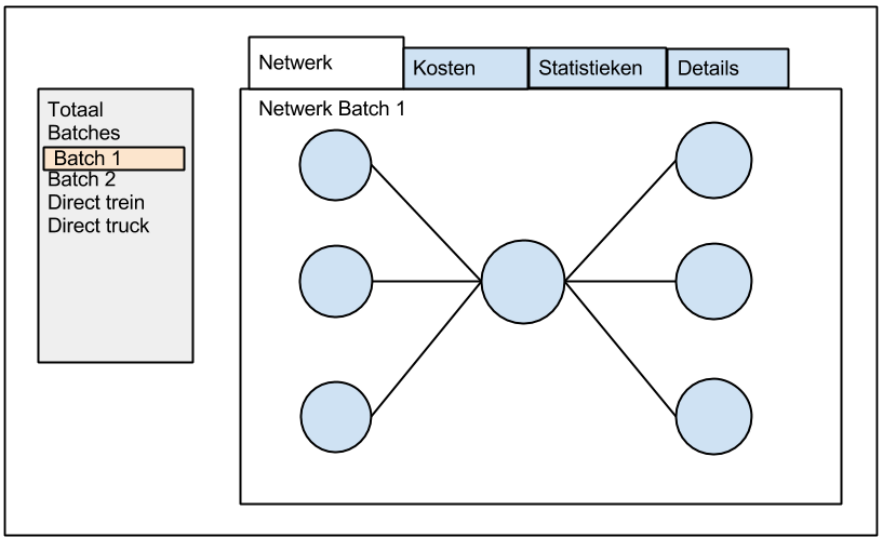
4.2.5 Andere oplossingen weergeven

De gebruiker kan een oplossing uit deze grafiek aanklikken en vervolgens kiezen hiervan de informatie te tonen. Op dat moment verandert de gehele input van het scherm, ofwel de informatie van de laatst gekozen oplossing (standaard de optimale oplossing) wordt in het scherm getoond.

4.3 Uitvoer

Wanneer de Solver de optimale oplossing heeft gevonden of de gebruiker de Solver heeft stopgezet en de oplossing wil bekijken, opent zich het volgende scherm van de applicatie: De weergave van de Uitvoer. Net als bij de invoer willen we de verschillende aspecten van de uitvoer kunnen laten weergeven. Hieronder proberen we te schetsen wat ons globale idee is; hierbij gaan we voornamelijk in op keuzes rondom probleem 3, 4 en 5 uit het vorige hoofdstuk.

4.3.1 De verdeling van het scherm



Figuur 4.3: Schets van het uitvoerscherm

Net als bij de invoer zullen we werken met een systeem met tabbladen, waardoor de gebruiker gemakkelijk kan wisselen tussen de verschillende onderdelen van de uitvoer.

Het uitvoerscherm zal bestaan uit twee delen. Links staat het overzicht van de oplossing, met de algemene informatie en de keuzemogelijkheid de verschillende batches / deeloplossingen aan te klikken. In het rechtergedeelte van het scherm bevinden zich de verschillende tabbladen met gedetailleerde informatie over de aangeklikte keuze, ofwel de totale oplossing of verschillende batches. Het idee is dat deze keuze hetzelfde blijft als de gebruiker wisselt van tabblad om de gebruiksvriendelijkheid te vergroten.

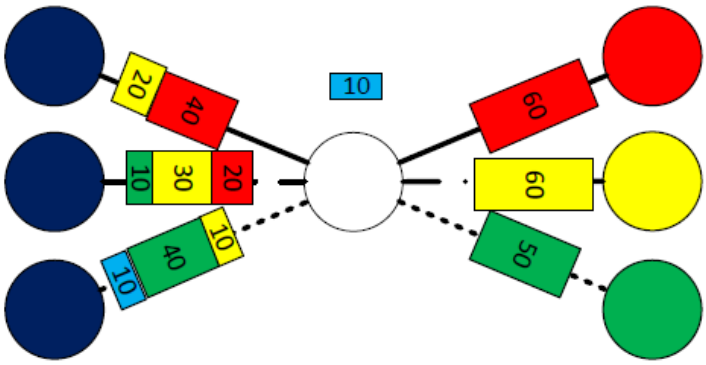
De verschillende tabbladen met uitgebreide(re) informatie zijn: De netwerkweergave, kostenweergave, statisieken en gedetailleerde kostenweergave. We zullen meer uitleggen over de eerste drie schermen. De gedetailleerde kostenweergave is hetzelfde als de gedetailleerde batchweergave die op dit moment via de interface van de Solver op te vragen is. Deze zullen wij in de applicatie laten.

Naast deze schermen is er de optie andere oplossingen weer te geven. Dit is een apart tabblad waarin opnieuw de voortgangsgrafiek te zien is en waar de gebruiker een oplossing kan aanklikken en kan kiezen hier de informatie over te tonen.

4.3.2 Netwerkweergave

In de netwerkweergave moet het voor de gebruiker mogelijk zijn om meer te weten te komen over hoe het transport volgens de oplossing plaats zal vinden, dit op een grafische manier. Hierbij maken we gebruik van het graafnetwerk dat ook wordt gebruikt bij de invoer. Echter, waar daar het netwerk ertoe diende om de invoer te controleren, kan de gebruiker hier verschillende acties ondernemen om meer informatie over het verloop van het transport te verkrijgen.

Allereerst geeft het netwerk inzicht in welke bestemmingen er betrokken zijn bij de oplossing of gekozen batch. Bij een batch zullen alleen de betrokken bestemmingen zichtbaar zijn. Daarnaast zal d.m.v. de figuurlijke uitbeelding en click and show (oplossing 4 en 5 van Probleem 4 van het vorige hoofdstuk) meer informatie gegeven worden over de verdeling van het transport. Hierbij heeft de gebruiker de optie om te kiezen om vanuit de bron of de bestemming te kijken. Vervolgens kan de gebruiker de verschillende bronnen / bestemmingen aanklikken en dan verschijnen er op of langs de kant blokken die duidelijk maken waar het transport naar toe gaat of vandaan komt. De verschillende locaties zullen met kleur aangegeven worden. We zullen dit illustreren met een voorbeeld, gebruikmakend van figuur 4.4.



Figuur 4.4: Grafische weergave van het verkeer

De figuur stelt een weer te geven batch voor. Stel, de gebruiker heeft gekozen het transport te bekijken vanuit de bron. Bij deze optie is het mogelijk de bronnen aan te klikken, waarna er zichtbaar is naar welke verschillende bestemmingen dit transport gaat. Zoals in de figuur zullen in deze optie de eindbestemmingen verschillende kleuren hebben en zijn de bronbestemmingen zwart (indien aangeklikt een andere kleur). Indien er geen bestemmingen zijn aangeklikt, zijn er geen

blokken zichtbaar. Klikt de gebruiker op de node linksboven, dan zullen de blokken op de edge tussen deze node en de hub zichtbaar worden. Of de blokken aan de rechterkant ook zichtbaar zullen worden in deze optie zal gekozen worden aan de hand van tests.

Uit het voorbeeld volgt dat wanneer de gebruiker het transport vanuit de bestemming bekijkt het precies andersom zal werken. Dan krijgen de beginbestemmingen verschillende kleuren.

De gebruiker heeft ook nog de optie 'alles aan' of 'alles uit' te kiezen. Deze optie is ook te vinden in het scherm. Ook zal er met de vorm van de blokken aangegeven worden of het transport per trein of vrachtwagen is. Het scherm zal voorzien zijn van legenda om alles duidelijk te maken aan de gebruiker.

4.3.3 Kostenweergave

In dit tabblad zullen we per gekozen onderdeel een samenvatting geven van de kosten hiervan. Dit zal uitgebreider zijn dan het basisscherm van de uitvoer van de Solver nu nog aangeeft, maar minder uitgebreid zijn als de gedetailleerde informatie. Doel is alle informatie kort, bondig en overzichtelijk weer te geven. Welke informatie er gegeven zal worden, moet nog besloten worden.

4.3.4 Statistieken

De Solver geeft op dit moment veel statistische informatie over de totale oplossingen en verschillen tussen de batches. Op dit moment worden alleen de getallen gegeven (minimum, maximum, mean en standaarddeviatie). In het Statistieken tabblad wordt er een grafische weergave van deze informatie gegeven.

De informatie op dit tabblad is altijd hetzelfde, onafhankelijk welke keuze in het linkerveld van het scherm is gemaakt.

4.4 Matrix

Bij het invoeren van de inputgegevens van de solver wordt op dit moment gebruik gemaakt van twee verschillende matrices. De eerste matrix bevat de verschillende afstanden tussen de nodes (sources, destinations en hubs). De tweede matrix bevat de flows tussen deze nodes. Deze worden op dit moment ingeladen via CSV bestanden. Een van onze taken is de invoer van deze gegevens te vereenvoudigen, we hebben hiervoor verschillende oplossingen bedacht.

4.5 Distance matrix

Allereerst kijken we naar de opties voor het invoeren van de distance matrix.

4.5.1 Vaste distance matrix binnen de tool

Ervan uitgaand dat alle sources, nodes en destinations vaste afstanden tot elkaar hebben en zelf ook constant zijn, zou het mogelijk moeten zijn de distance matrix al van te voren binnen de tool te definieren. Hierdoor hoeft de gebruiker alleen nog de flows in te voeren. Dit is echter afhankelijk van de wens van de opdrachtgever of dit flexibel moet zijn.

4.5.2 Geografische informatie

Voor het representeren van de nodes op een graaf met een geografisch logische layout, moeten er geografische gegevens beschikbaar zijn voor de nodes. Deze zouden ook ingevoerd moeten worden in de distance matrix. Hier kan een speciaal bron/bestemming editor scherm voor gemaakt worden, waarbij parameters ingesteld kunnen worden voor elke bron/bestemming (waaronder geografische ligging).

4.5.3 Import van distance matrix

Een andere oplossing is het direct importeren van een bestand dat de distance matrix bevat zoals nu ook het geval is. Het is dan echter wenselijk de input matrix rechthoekig te laten zijn en verder conversie binnen de tool te laten plaatsvinden.

4.6 Flow matrix

Dit zijn de oplossingen voor het invoeren van de flow matrix.

4.6.1 Inputtabel in de tool

De flowmatrix invoer wordt ook verplaatst naar de tool. Hier is het mogelijk het aantal rijen en kolommen te kiezen en per rij en kolom de source/destination in te stellen. Hierna kunnen de flows ingevuld worden. Deze matrix kan vervolgens ook geëxporteerd worden naar een bestand in CSV of Excel formaat.

4.6.2 Import van flow matrix

Ook hier is een oplossing om de flow matrix direct te importeren uit een bestand mogelijk. Dit kan ook weer gecombineerd worden met een inputtabel.