TECHNISCHE UNIVERSITEIT DELFT

BACHELORPROJECT

NedTrain Planner

Construeren van Flexibele Roosters

Auteurs:
K.C. Bakker
P.A. Bouter
M. den Hoedt

Commissie:

M.A. Larson	Coördinator Bachelorproject
Prof. Dr. C. Witteveen	TU Delft Coach
Ir. B. Huisman	NedTrain

7 juli 2014

TU Delft Delft University of Technology

NedTrain

Voorwoord

Dit verslag is het eindresultaat van het Bachelorproject, dat dient ter afsluiting van de Bachelor Technische Informatica aan de Technische Universiteit Delft. Het project is uitgevoerd in opdracht van het bedrijf NedTrain B.V. en de TU Delft.

De opdracht bestond uit het implementeren van nieuwe algoritmen, die voort zijn gekomen uit recent onderzoek, en het verbeteren van de NedTrain Planner, een visuele tool voor het oplossen van roosterproblemen. Dit verslag geeft inzicht in de werkzaamheden die door ons uitgevoerd zijn in de tien weken waarin het project plaatsvond. Daarnaast beschrijft het de keuzes die tijdens het proces gemaakt zijn, geeft het de resultaten die behaald zijn en biedt het ondersteuning voor een eventueel vervolgproject.

Wij willen hierbij onze begeleiders ir. Bob Huisman, manager van de Maintenance Development afdeling van NedTrain, en prof. dr. Cees Witteveen, werkzaam bij de Algoritmiek Groep op EWI aan de TU Delft, bedanken voor de feedback op zowel het product als het verslag en het geven van ideeën en inzichten voor nieuwe functionaliteit.

7 juli 2014 te Delft

Chris Bakker
Anton Bouter
Martijn den Hoedt

Inhoudsopgave

Samenvatting

Het bedrijf NedTrain beschikt over software om roosterproblemen op te lossen. De wens van NedTrain en het doel van dit project is om deze software uit te breiden met de functionaliteit om flexibele schema's te berekenen. Deze roosterproblemen hebben te maken met het probleem dat ook wel bekend staat als het *Resource Constrained Project Scheduling Problem*. Hierbij wordt er naar een schema gezocht voor bijvoorbeeld het onderhoud aan treinen. Doordat elke trein binnen een bepaalde tijd gerepareerd moet worden en er ook rekening gehouden moet worden met de beschikbare resources, is dit probleem moeilijk om op te lossen.

De bestaande software van NedTrain bestaat uit een interface, genaamd de NedTrain Planner, en een solver, die er voor zorgt dat instanties opgelost worden. Deze software kan echter alleen vaste schema's genereren. Op het moment dat er activiteiten verplaatst worden, kunnen er conflicten ontstaan, waardoor het schema niet meer geldig is. Dit wordt opgelost door het implementeren van een nieuwe solver. Deze solver maakt gebruik van het chaining algoritme en de *COIN Linear Programming* library. Het chaining algoritme is nodig om te zorgen dat bij het verplaatsen van activiteiten het schema consistent blijft qua resources. De COIN LP library is gebruikt om flexibiliteitsintervallen te kunnen berekenen, die vervolgens worden weergegeven in de interface.

Tijdens dit project is er gebruik gemaakt van Scrum met zes wekelijkse sprints. Als versiebeheersysteem is Git gebruikt op een private repository gehost door Bitbucket.com. Als continuous integration systeem is Jenkins gebruikt, maar dit was helaas geen succes. De codekwaliteit is gemeten door de Software Improvement Group, die vervolgens feedback heeft gegeven over verschillende aspecten van de code.

Er is een aantal aanbevelingen voor ontwikkelaars die de NedTrain Planner gaan doorontwikkelen. Er wordt aanbevolen om een undo-functionaliteit te implementeren, de communicatie tussen interface en solver sneller te maken, de kwaliteit van de code te verbeteren en activiteiten beter te laten verdelen over alle beschikbare chains.

Van alle wensen van de opdrachtgever zijn alle features met een hoge prioriteit geïmplementeerd en is ook een groot aantal features met middelmatige of lage prioriteit geïmplementeerd. Er kan dus geconcludeerd worden dat er aan de eisen van de opdrachtgever voldaan is en dat de doelen bereikt zijn.

1 Inleiding

Dit verslag is het eindresultaat van het Bachelorproject in opdracht van NedTrain B.V. en de Technische Universiteit Delft. Het project was gericht op het uitbreiden en verbeteren van software voor een applicatie, de NedTrain Planner, die roosterproblemen kan oplossen en de uitkomsten hiervan op een overzichtelijk manier visualiseert. De applicatie kan voor verschillende doeleinden gebruikt worden door zowel NedTrain als de TU Delft. Er moet dus tijdens het project rekening worden gehouden met verschillende belangen en wensen van beide opdrachtgevers.

In de komende hoofdstukken zal de opdracht en het proces eromheen beschreven worden. Allereerst zal er gedefiniëerd worden wat het probleem van de opdrachtgever is. Vervolgens zal het hoofdstuk probleemanalyse gericht zijn op het analyseren en oplossen van het gedefiniëerde probleem van de opdrachtgevers. Daarna zullen in het hoofdstuk ontwerp de ontwerpkeuzes genoemd en toegelicht worden en komen in het daarop volgende hoofdstuk alle opgeleverde producten en diensten aan bod. Omdat de applicatie gericht is op het oplossen van moeilijke problemen zal er ook een hoofdstuk gewijd zijn aan de performance van de opgeleverde producten. Vervolgens zal besproken worden hoe het ontwikkelproces gelopen is en wat voor hulpmiddelen er gebruikt zijn. Tenslotte zullen er functionaliteiten aanbevolen worden die eventueel later nog geïmplementeerd kunnen worden en wordt in de conclusie teruggekeken op het project en een conclusie getrokken met betrekking tot de eisen van het project. Als voorbereiding op dit project is in de eerste weken een aantal documenten opgesteld, namelijk een Plan van Aanpak *(zie bijlage B)*, een Oriëntatieverslag *(zie bijlage C)* en een Requirementsanalyse *(zie Bijlage D)*.

De volgende paragrafen zullen een korte inleiding geven over het bedrijf NedTrain en waarom de NedTrain Planner relevant is voor hen en de TU Delft.

1.1 NedTrain

Het bedrijf NedTrain is een dochterbedrijf van de Nederlandse Spoorwegen (NS), net als bijvoorbeeld NS Reizigers B.V., de verantwoordelijke voor het personenvervoer. NedTrain is het dochterbedrijf dat zorgt voor het onderhoud aan alle treinen aangeleverd door NS Reizigers. Dat onderhoud bestaat niet alleen uit het repareren, maar ook uit het reinigen en moderniseren van treinen. Er moet hierbij gezorgd worden dat van de grofweg 3000 bestaande treinstellen bij NS er op elk moment genoeg beschikbaar zijn om gebruikt te kunnen worden door NS Reizigers. Om dit te bereiken staan er op elk moment van de dag ongeveer 250 treinstellen tegelijk voor onderhoud op meer dan 30 locaties door heel Nederland in de werkplaatsen van NedTrain. Hier werken in totaal ongeveer 3500 mensen 24 uur per dag, 7 dagen per week aan de opgenoemde taken. Het hoofdkantoor van NedTrain staat in Utrecht.

1.2 NedTrain Planner

Vanwege de omvang van de activiteiten die NedTrain dagelijks moet afhandelen, worden er schema's gemaakt die een weergave van de dag geven en er voor moeten zorgen dat deze activiteiten op tijd uitgevoerd worden. Dit is een uitputtende taak en alhoewel NedTrain in het bezit is van software moet er nog steeds veel met de hand worden gedaan. Om NedTrain te helpen bij het inzicht geven van de nieuwste ontwikkelingen rondom het oplossen van moeilijke roosterproblemen, is er in samenwerking met de TU Delft de NedTrain Planner ontwikkeld, dat het resultaat is van meerdere projecten van studenten aan de TU Delft. Roostermakers bij NedTrain zouden deze applicatie kunnen gebruiken om hun werkverrichtingen te vergemakkelijken en hun denkwijze te verbeteren. Daarnaast biedt het voor de TU Delft een educatief demonstratiemiddel voor onderzoek, doordat de applicatie visueel goed kan laten zien hoe de oplossing door een algoritme tot stand komt en hoe dus een dergelijk algoritme werkt.

2 Probleemstelling

De NedTrain Planner is een uitgebreide grafische applicatie waarin roosterproblemen kunnen worden aangemaakt, geïmporteerd, gevisualiseerd en aangepast. Door middel van een zogenaamde solver, die onafhankelijk is van de gebruikersinterface en die ingeladen moet worden in de applicatie, kunnen deze roosterproblemen opgelost worden. Vervolgens kan er informatie verkregen en opgevraagd worden over de gevonden oplossing en kan er stapsgewijs door het proces van de solver heen gelopen worden, om zo een beter beeld te krijgen van hoe de solver werkt. Daarnaast geeft de NedTrain Planner ook de mogelijk om na het oplossen aanpassingen te maken en de instantie vervolgens opnieuw te laten oplossen. Zo kunnen veranderingen in het resultaat makkelijk weergegeven worden.

Eén van de mogelijkheden die de NedTrain Planner biedt na het oplossen, is het verschuiven van activiteiten over hun uitvoerbare intervallen. Deze intervallen tonen aan in hoeverre activiteiten verschoven kunnen worden, zonder dat dit leidt tot een inconsistente oplossing. De huidige solver biedt echter alleen de garantie dat het gegenereerde schema waarbij elke activiteit op zijn vroegste starttijd begint een consistente oplossing van het probleem is. Het is dus mogelijk dat er bij het verschuiven van een activiteit de capaciteit van een resource overschreden wordt, waardoor het schema inconsistent wordt.

De opdrachtgevers hebben, als beschrijving van dit Bachelorproject, een document *(zie Bijlage A)* opgesteld waarin wordt beschreven wat de opdracht van het project is. Na nader overleg tussen het ontwikkelteam en de opdrachtgevers kwam daar een concreet probleem uit. Dit probleem is op te delen in drie gedeeltes.

Het eerste gedeelte van het probleem is het implementeren van een algoritme dat garandeert dat voor elke tijdsbepaling van de activiteiten een consistent schema bestaat. Er moet dus uiteindelijk met de activiteiten over hun uitvoerbare intervallen verschoven kunnen worden, zonder dat een resource zijn capaciteit overschrijdt of dat er een andere beperking geschonden wordt.

Het tweede gedeelte van het probleem is om de NedTrain Planner uit te breiden met functionaliteit die er voor zorgt dat er flexibele schema's geconstrueerd en gevisualiseerd kunnen worden. Dit betekent dat activiteiten vrij over een interval moeten kunnen bewegen, zonder dat daarbij de starttijd of vorm van een andere activiteit veranderd wordt. Dit geeft NedTrain de mogelijkheid om bij het samenstellen van een schema makkelijk aanpassingen kunnen maken, zonder daarbij het hele rooster te hoeven veranderen of opnieuw te berekenen.

Het laatste gedeelte van het probleem is om informatie te geven over de flexibiliteit en het berekenen hiervan. Daarnaast moeten veranderingen in deze informatie geëvalueerd kunnen worden na het geven van gebruikersfeedback. Dit kan door de TU Delft gebruikt worden om beter te begrijpen hoe de algoritmes zich gedragen en kan het helpen om verder onderzoek naar het oplossen van roosterproblemen te stimuleren.

3 Probleemanalyse

Het roosterprobleem waarmee NedTrain te maken heeft, zal in dit hoofdstuk formeel gedefiniëerd worden. Vervolgens zal een aantal oplosmethoden besproken worden voor de verschillende onderdelen van dit probleem.

3.1 Resource Constrained Project Scheduling Problem

Het probleem waar NedTrain mee te maken heeft, staat ook wel bekend als het *Resource Constrained Project Scheduling Problem (RCPSP)*. Dit probleem kan in verschillende variaties en uitbreidingen voorkomen, maar het globale probleem voor elke variatie is het vinden van een rooster bestaande uit activiteiten, zodanig dat de resources die de activiteiten nodig hebben niet de maximale capaciteit van de resources overschrijdt en dat de opgelegde beperkingen niet geschonden worden [4]. Het is bekend dat het RCPSP een NP-moeilijk probleem is [2]. Dat wil zeggen dat er geen polynomiaal algoritme voor dit probleem bekend is, waardoor alleen kleine instanties van dit probleem in een redelijke tijd opgelost kunnen worden.

De instanties van NedTrain bestaan uit projecten (treinen) $P = \{p_1, \ldots, p_n\}$ die uit één of meer subactiviteiten $v_{a,b}$ bestaan. De verzameling van alle activiteiten, $\bigcup_{\{p_a \in P\}} V_a$, wordt gedefinieerd als de verzameling $V = \{v_{1,1}, \ldots, v_{n,m}\}$. Een project p_a heeft een starttijd *(release time)* rs_a en een deadline dl_a. De subactiviteiten van project p_a mogen pas vanaf rs_a uitgevoerd worden, maar moeten wel uiterlijk op dl_a klaar zijn. De b^e activiteit van project p_a wordt weergegeven als $v_{a,b}$. Deze activiteit heeft een tijdsduur van $d_{a,b}$ tijdseenheden.

Er zijn drie soorten voorwaarden *(constraints)* waar rekening mee gehouden moet worden bij het oplossen van een RCPSP. Dit zijn tijdsconstraints, voorrangsrelaties *(precedence constraints)* en resource constraints. De tijdsconstraints zijn de hierboven genoemde starttijden en deadlines van de projecten. Een eventuele voorrangsrelatie $v_{i,j} \prec v_{u,v}$ kan bestaan tussen twee activiteiten $v_{i,j}$ en $v_{u,v}$, waardoor activiteit $v_{u,v}$ pas uigevoerd mag worden nadat activiteit $v_{i,j}$ voltooid is. Tenslotte zijn oplossingen ook gelimiteerd door de resource constraints. Elke activiteit kan namelijk van verschillende soorten resources r_k een bepaald aantal units nodig hebben. Het aantal units dat activiteit $v_{i,j}$ van resource r_k nodig heeft, wordt genoteerd als $req(v_{i,j}, r_k)$.

Een formele definitie van RCPSP kan nu als volgt opgesteld worden:

Gegeven:
Een verzameling projecten $P = \{p_1, \ldots, p_n\}$ met voor elk project p_a een starttijd rs_a, een deadline dl_a en een verzameling activiteiten $V_a = \{v_{a,1}, \ldots, v_{a,m}\}$ met voor elke activiteit $v_{a,b}$ een tijdsduur $d_{a,b}$. Daarnaast een verzameling van voorrangsrelaties $E \subseteq V \times V = \{(v_{a,b}, v_{c,d}) | v_{a,b} \in V_a \wedge v_{c,d} \in V_c \wedge p_a, p_c \in P\}$, een verzameling van w resources $R = \{r_1, \ldots, r_w\}$ met voor elke resource r_i een eindige capaciteit $cap(r_i)$, en voor elke activiteit $v_{a,b} \in V$ een hoeveelheid van resource $r_k \in R$ die nodig is, $req(v_{a,b}, r_k)$.

Vind:
Een schema $S = \{s_1, \ldots, s_m\}$ bestaande uit starttijden $s_{a,b}$ voor elke activiteit $v_{a,b} \in V$, zodat $rs_a \leq s_{a,b} \wedge s_{a,b} + d_{a,b} \leq dl_a$, waarbij voor elke voorrangsrelatie $(v_{a,b}, v_{c,d}) \in E$ geldt dat $s_{a,b} + d_{a,b} \leq s_{c,d}$. Bovendien mag op geen enkel tijdstip de capaciteit van een resource overschreden worden, wat inhoudt dat $\forall r_i \in R$ en $\forall t \in \mathbb{R}$ geldt dat $\Sigma_{\{v_{a,b} \in V | s_{a,b} \leq t \leq s_{a,b} + d_{a,b}\}} req(v_{a,b}, r_i) \leq cap(r_i)$.

3.2 Flexibiliteit

Om er voor te zorgen dat de flexibiliteit van een schema geoptimaliseerd kan worden, moet de flexibiliteit van een schema eerst goed gedefiniëerd worden. Een mogelijke (naïeve) maat voor de flexibiliteit van een activiteit is simpelweg de grootte van het toegelaten interval van deze activiteit. Deze grootte wordt berekend door het verschil te nemen tussen de eerste starttijd (EST) en de laatste starttijd (LST). De totale flexibiliteit van een schema kan vervolgens berekend worden met Formule 1. Deze flexibiliteit wordt echter niet beïnvloed als er voorrangsrelaties tussen activiteiten worden toegevoegd, terwijl dit wel degelijk een negatieve invloed heeft op de flexibiliteit van het schema. Een aantal andere flexibiliteitsmaten, besproken in [10], houdt ook geen rekening met onderlinge afhankelijkheden van activiteiten bij het vaststellen van de flexibiliteit.

$$flex_{totaal} = \sum_{v \in V} (EST(v) - LST(v)) \tag{1}$$

De flexibiliteitsmaat van Wilson *et al.* [10] houdt echter wel rekening mee dat een schema minder flexibel wordt door het toevoegen van voorrangsrelaties. Deze maat zal dus in het vervolg gebruikt worden om de flexibiliteit van een schema te maximaliseren. Voor het vaststellen hiervan moet voor elke activiteit $v_{a,b}$ een interval $[l_v, u_v]$ berekend worden, zodat elke activiteit in zijn eigen interval verschoven kan worden, zonder dat daardoor andere activiteiten verplaatst hoeven te worden. Dit houdt dus in dat de verzameling van intervallen $I_S = \{I_v = [l_v, u_v] | v \in V\}$ voldoet desda voor elke $v \in V$ en voor elke $t_v \in [l_v, u, v]$, een toewijzing van starttijd t_v aan activiteit v een consistent schema oplevert. Vervolgens kan de flexibiliteit van het schema vastgesteld worden door de lengtes van de intervallen bij elkaar op te tellen, zoals in formule 2:

$$flex_{totaal} = \sum_{v \in V} (u_v - l_v) \tag{2}$$

3.3 Oplossing

In de bestaande software was al een algoritme geïmplementeerd dat een consistent schema kan vinden door middel van het ESTA$^+$ algoritme [3]. Dit algoritme lost het RCPSP op door het vinden en oplossen van *contention peaks*, plekken in het resource profiel waar er meer resource units gebruikt worden dan er beschikbaar zijn. Deze worden veroorzaakt doordat er te veel activiteiten die dezelfde resource gebruiken tegelijk uitgevoerd worden. Deze contention peaks kunnen geëlimineerd worden door tussen ten minste 1 paar van de betreffende activiteiten een voorrangsrelatie toe te voegen. Het $ESTA^+$ algoritme levert als oplossing voor elke activiteit een starttijd, zodat het gehele schema consistent is. Als er echter met activiteiten geschoven wordt, kan het schema nog wel inconsistent worden.

3.3.1 Chaining

Om onafhankelijke flexibiliteitsintervallen te kunnen bepalen, moeten activiteiten vrij verschoven kunnen worden, zonder dat hierdoor het schema inconsistent wordt. Hier kan het chaining algoritme voor zorgen [5]. Dit algoritme splitst elke resource op in zogenaamde units van elk een capaciteit van 1. Vervolgens wordt aan elke resource unit een zogenaamde chain, een gesorteerde lijst van activiteiten, toegekend. De activiteiten in deze chain zullen de enige activiteiten zijn die gebruik mogen maken van de betreffende resource unit. In een chain geldt voor elk paar opeenvolgende activiteiten, $v_{a,b}$ en $v_{c,d}$, dat er een voorrangsrelatie $v_{a,b} \prec v_{c,d}$ bestaat. Hierdoor

kunnen twee activiteiten uit dezelfde chain nooit tegelijk uitgevoerd worden. Bovendien wordt een activiteit die meerdere resource units nodig heeft, voor elke resource unit aan een aparte chain toegevoegd. Als een bepaalde activiteit dus bijvoorbeeld drie resource units nodig heeft, is deze activiteit dus aanwezig in precies drie verschillende chains. Op deze manier kan het aantal benodigde resource units nooit groter zijn dan de capaciteit, omdat twee activiteiten uit dezelfde chain nooit tegelijk uitgevoerd kunnen worden en het aantal chains van een resource gelijk is aan de capaciteit daarvan.

De output van het chaining algoritme bestaat uit een zogenaamd *Partial Order Schedule (POS)*, wat een schema is waarvoor geldt dat elk schema dat qua tijdsconstraints consistent is, ook consistent is met de resource constraints. Het exacte chaining algoritme is weergegeven in Algoritme 1.

Algorithm 1 Chaining [5]

Input: Een probleem P en daarvan een (fixed-time) oplossing S
Output: Een Partial Order Schedule POS^{ch}

1: **function** CHAINING(P, S)
2: $POS^{ch} \leftarrow P$
3: Sorteer alle activiteiten op hun starttijd in S
4: Initialiseer alle chains leeg
5: **for each** resource r_j **do**
6: **for each** activity v_i **do**
7: **for** 1 **to** req_{ij} **do**
8: $k \leftarrow SelectChain(v_i, r_j)$
9: $v_k \leftarrow last(k)$
10: $AddConstraint(POS^{ch}, v_k \prec v_i)$
11: $last(k) \leftarrow v_i$
12: **return** POS^{ch}

De functie $AddConstraint(P, v_k \prec v_i)$ zorgt ervoor dat de voorrangsrelatie $v_k \prec v_i$ aan het probleem toegevoegd wordt en werkt vervolgens de starttijden van de betrokken activiteiten bij. In de functie $SelectChain(v_i, r_j)$ wordt een chain gekozen waar activiteit v_i aan toegevoegd zal worden. Een simpele, voor de hand liggende methode is om een activiteit gelijk aan een chain toe te voegen als er een geschikte chain gevonden is. Een chain is geschikt voor een activiteit v_i als de eindtijd van het laatste element van de chain, v_k, eerder is dan de starttijd van v_i, dus $s_k + d_k \leq s_i$. Er kan ook gekozen worden om een willekeurige chain te selecteren. Hiervoor moeten eerst alle geschikte chains gevonden worden en wordt er vervolgens uit deze verzameling een willekeurig element gekozen. Tenslotte is er ook in [6] een heuristiek voorgesteld die probeert om het aantal voorrangsrelaties tussen activiteiten uit verschillende chains te minimaliseren. Deze heuristiek, weergegeven in 2, probeert eerst een chain k te vinden waarvoor geldt dat er al een voorrangsrelatie $last(k) \prec v_i$ bestaat. Als een dergelijke chain bestaat, wordt de voorrangsrelatie $last(k) \prec v_i$ toegevoegd en anders wordt er voor v_i een willekeurige chain uitgekozen.

Als v_i nog meer units van dezelfde resource nodig heeft, wordt er eerst geprobeerd om v_i aan chains toe te voegen die ook $last(k)$ als laatste element hebben, omdat er dan geen nieuwe voorrangsrelatie toegevoegd hoeft te worden. De voorrangsrelatie $v_i \prec last(k)$ bestaat immers al. Hierna komen pas andere chains, die $last(k)$ niet als laatste element hebben, in aanmerking voor activiteit v_i.

Algorithm 2 SelectChain Heuristiek [6]

Input: Een probleeminstantie P, een verzameling voorrangsrelaties E, een activiteit v_i en een resource r_j.

function SELECTCHAIN(P, v_i, r_j)
  $C_a \leftarrow \{\text{chain } c | last(c) \in E\}$
  if $C_a \neq \emptyset$ **then**
    $k \leftarrow$ een willekeurige chain uit C_a
  else
    $k \leftarrow$ een willekeurige geschikte chain
  $AddConstraint(P, v_k \prec v_i)$
  if $req(v_i, r_j) > 0$ **then**
    $C_k \leftarrow \{\text{chain } c | last(c) == last(k)\}$
    $\overline{C_k} \leftarrow \{\text{chain } c | last(c) \neq last(k)\}$
    for $req = req(v_i, r_j) - 1 \ldots 0$ **do**
      if $C_k \neq \emptyset$ **then**
        $c_k \leftarrow$ willekeurig element in C_k
        $AddConstraint(P, last(c_k) \prec v_i)$
        $C_k \leftarrow C_k - c_k$
      else
        $c_k \leftarrow$ willekeurig element in $\overline{C_k}$
        $AddConstraint(P, last(c_k) \prec v_i)$
        $\overline{C_k} \leftarrow \overline{C_k} - c_k$

3.3.2 Linear Programming

Nu alle voorrangsrelaties door het chaining algoritme aan het model zijn toegevoegd, kan de flexibiliteit van het schema worden geoptimaliseerd, door de probleeminstantie om te zetten naar een Linear Programming *(LP)* probleem. Dit probleem kan in polynomiale tijd opgelost worden door middel van een LP-solver.

De flexibiliteit van een activiteit kan berekend worden door middel van $flex = t^+ - t^-$, waarbij t^+ de laatste starttijd en t^- de vroegste starttijd van een activiteit t is. De flexibiliteit van het hele schema wordt, zoals in paragraaf 3.2, gemeten door de som van de flexibiliteit van alle activiteiten $t \in T$ te nemen, zoals in Formule 3 is weergegeven.

$$flex_{totaal} = \sum_{t \in T} (t^+ - t^-) \tag{3}$$

Het doel is de flexibiliteit te maximaliseren en ervoor te zorgen dat er tegelijkertijd wordt voldaan aan de voorwaarden $(t - t' \leq c) \in C$. Ook mag de EST niet na de LST zijn ($t^- \leq t^+ \Rightarrow 0 \leq t^+ - t^-$). Hieruit volgt het LP-probleem zoals in formule 4 weergegeven is.

$$\begin{aligned} \text{max:} \quad & \sum_{t \in T} (t^+ - t^-) & \\ \text{met voorwaarden:} \quad & 0 \leq t^+ - t^- & \forall t \in T \\ & t^+ - t^- \leq c & \forall (t^+ - t^{'-} \leq c) \in C \end{aligned} \tag{4}$$

Het nadeel van de formulering in formule 4, is dat de flexibiliteit niet netjes verdeeld hoeft te worden over de taken. Bij het testen is zelfs gebleken dat de 'Simplex' methode meestal de flexibiliteit juist heel slecht verdeelt. Met de 'Interior Point' methode bleek de flexibiliteit wel

beter verdeeld te worden. Beide methoden zijn beschikbaar in de CLP library en komen wel uit op hetzelfde antwoord van de geoptimaliseerde variabele, maar verschillen qua waarden van de andere variabelen. Het exacte verschil tussen deze twee methoden ligt buiten het kader van dit verslag.

Met het LP-probleem dat weergegeven is in Formule 5 wordt de minimale flexibiliteit van elke activiteit min gemaximaliseerd [9]. Vervolgens wordt van het LP-probleem, zoals in Formule 6, de totale flexibiliteit gemaximaliseerd. Dit is hetzelfde stelsel als in Formule 4, maar nu met als nieuwe voorwaarde dat elke activiteit minimaal een flexibiliteit van min moet hebben ($min \leq t^+ - t^-$).

$$\begin{aligned} \text{max:} \quad & min \\ \text{met voorwaarden:} \quad & min \leq t^+ - t^- && \forall t \in T \\ & t^+ - t'^- \leq c && \forall (t - t' \leq c) \in C \end{aligned} \tag{5}$$

$$\begin{aligned} \text{max:} \quad & \sum_{t \in T} (t^+ - t^-) \\ \text{met voorwaarden:} \quad & min \leq t^+ - t^- && \forall t \in T \\ & t^+ - t'^- \leq c && \forall (t - t' \leq c) \in C \end{aligned} \tag{6}$$

Door het toevoegen van de nieuwe voorwaarde $min \leq t^+ - t^-$, kan het voorkomen dat de totale flexibiliteit $flex_{totaal}$ lager is dan bij het gebruik van formule 4, maar in dit geval is de flexibiliteit wel beter over alle activiteiten verdeeld.

4 Ontwerp

In dit hoofdstuk zal het ontwerp van de NedTrain Planner besproken worden. Aangezien de applicatie al voor een groot gedeelte bestond, moet er voortgebouwd worden op het bestaande ontwerp van de vorige projecten. Allereerst zal het ontwerp van de interface besproken worden. Vervolgens zal het ontwerp van de solver gegeven worden en zal er toegelicht worden hoe de communicatie tussen de nieuwe solver en de interface werkt.

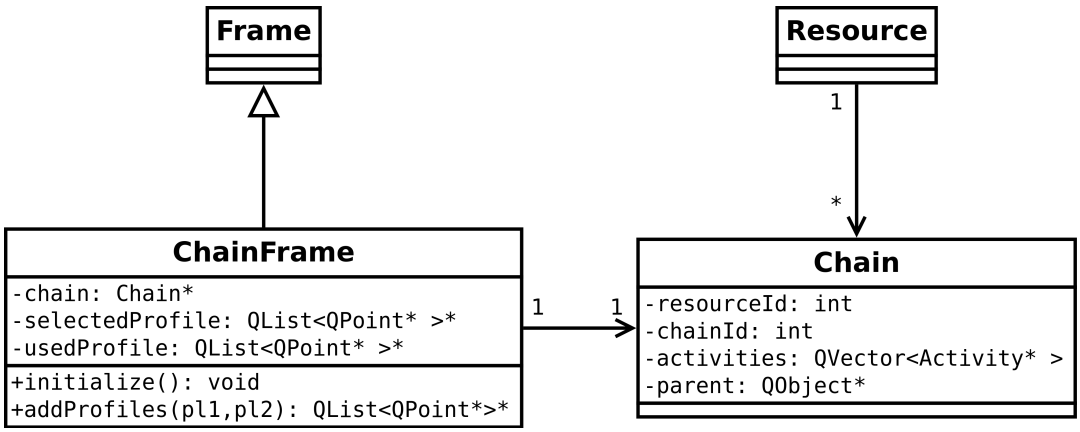
4.1 Interface

Voor de implementatie van nieuwe features aan de interface is voor het grootste deel gebruik gemaakt van de al bestaande klassen. Klassediagrammen van de aangeleverde interface zijn terug te vinden in [1].

De volgende klassen zijn wel toegevoegd aan het bestaande model:

- `Chain` - In deze klasse wordt de lijst van activiteiten opgeslagen die bij een bepaalde chain hoort. Een chain is te identificeren door middel van twee gehele getallen: de resourceId en de chainId. De resourceId identificeert de bijbehorende resource en de chainId geeft aan bij welke unit van de resource deze chain hoort.
- `ChainFrame` - Deze klasse is een extensie van de klasse `Frame` en wordt gebruikt om in de interface alle voorrangsrelaties van de geselecteerde chain te laten zien. Daarnaast kan uit deze klasse het resource profiel van de betreffende chain berekend worden.
- `JobInfoDialog` - Dit is een `QDialog` venster waarin informatie, zoals de starttijd en deadline, van een project wordt weergegeven.

In Figuur 1 is het klassediagram weergegeven waarin `Chain` en `ChainFrame` toegevoegd zijn. De verdere relaties van `Frame` en `Resource` zijn voor de duidelijkheid weggelaten, omdat deze in dit geval irrelevant zijn. Bovendien zijn in het klassediagram `get`- en `set`-methoden weggelaten.

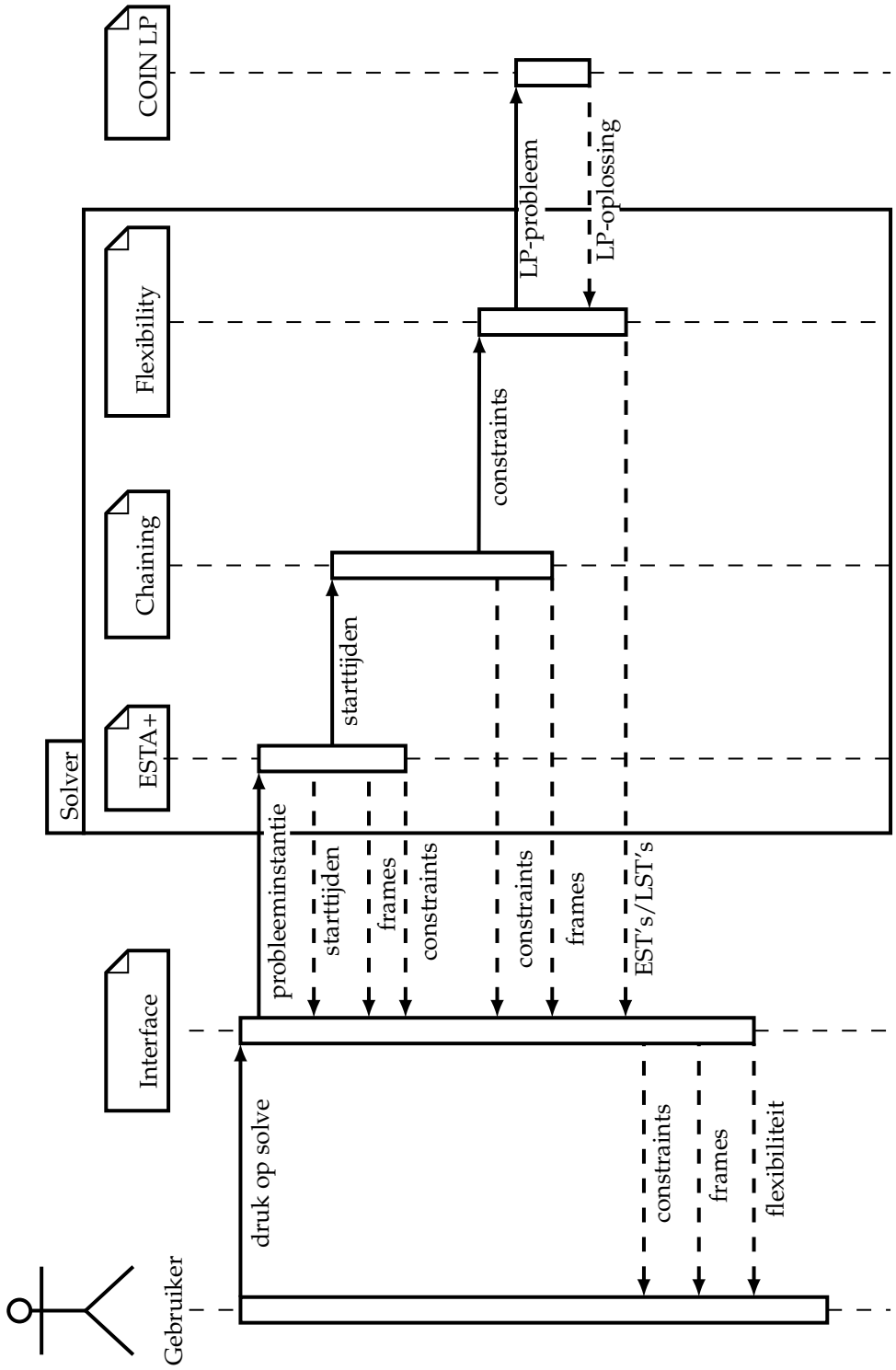


Figuur 1: Klassediagram van toegevoegde klassen.

4.2 Solver

Als de gebruiker op de knop 'solve' drukt, wordt de solver aangeroepen. Deze leest de probleeminstantie in en verwerkt deze. Uiteindelijk geeft de solver weer informatie terug aan de applicatie, die de informatie weergeeft op het scherm. Deze informatie bestaat onder andere uit de chains en de flexibiliteitsintervallen.

De solver bestaat grofweg uit drie componenten, namelijk het ESTA$^+$ algoritme, het chaining algoritme en een algoritme voor het berekenen van de flexibiliteitsintervallen. Het ESTA$^+$ algoritme was al geïmplementeerd door onze voorgangers en is niet veranderd. Het chaining algoritme, uitgelegd in paragraaf 3.3.1, is wel in dit project geïmplementeerd. De code voor het berekenen van de flexibiliteitsintervallen bouwt een LP-probleem op en lost dit vervolgens op met de COIN LP (CLP) library. Hoe de solver werkt en wat voor aanroepen er gedaan worden, is weergegeven in het sequencediagram in Figuur 2.



Figuur 2: Sequencediagram van het proces dat uitgevoerd wordt bij het aanroepen van de solver.

4.3 Communicatie

In de bestaande versie van de NedTrain Planner bestond al een communicatieprotocol voor communicatie tussen de NedTrain Planner en de bestaande solver. Deze communicatieregels worden ook gebruikt door de nieuwe solver [1] en zijn als volgt:

- `PROGRESS`: geeft een indicatie (in procenten) van de voortgang van het solveproces. De syntax is `'PROGRESS: <percentage>'`.
- `ERROR`: geeft bericht van een interne fout in de solver (een invariant tijdens het solve proces blijkt niet waar te zijn). Dit zou niet mogen optreden.
- Een bericht of de instantie succesvol is opgelost. Dit is een regel `'Instance solved.'` of `'Instance not solved.'`.
- `PC`: duidt aan dat een voorrangsrelatie toegevoegd wordt. De syntax is als volgt: `'PC: <i1> <j1> <i2> <j2>'`, waarbij `<i1> <j1>` de voorgaangde activiteit aanduidt, en `<i2> <j2>` de opvolgende activiteit.
- `STATE`: Geeft een beschrijving van de huidige verzameling groepen (partitie in groepen van de taken) in de solver en de huidige EST en LST van elke groep. De syntax is `'STATE: <groepen> -1'`, waarbij `-1` als scheidingsteken wordt gebruikt. Het `<groepen>` blok bevat de specificaties van groepen. Specificatie per groep begint met `<treinID> <EST> <LST> <#taken>`. Daarna volgen `#taken` taken, aangeduid met `<projectID> <activiteitID>`.
- `PEAK`: Deze regel wordt geprint als het solven niet gelukt is, omdat er aan het einde een contention peak was die niet geresolved kon worden. Het formaat is `'PEAK:' <time> <resource> <capacity> <lijst taken en groepen> '-1'`. Eerst wordt aangegeven waar de peak zich bevindt en de capaciteit. Elke activiteit of groep wordt gerepresenteerd met `<treinID> <activiteitID>`. Voor groepen geldt dat een van de taken die in de groep zitten wordt gegeven als representant. Vanuit de laatste `STATE` kan worden afgeleid welke groep bedoeld wordt. Deze peak kan door het planprogramma worden gevisualiseerd.
- `MUTEX`: Deze regel wordt geprint als het solven niet gelukt is, omdat er niet kon worden voldaan aan de eis dat wederzijdse uitsluiting moet gelden. Het formaat is hetzelfde als bij `PEAK`.

De volgende communicatieregels zijn toegevoegd en zijn nodig om de NedTrain Solver te laten communiceren met de nieuwe solver:

- `CHAIN`: De syntax is `'<resourceID> <chainID> <#activiteiten> <activiteiten> -1'`. Hierbij is `<activiteiten>` een lijst van `#activiteiten` activiteiten met elke activiteit in de vorm `<projectID> <activiteitID>`. Dit wordt gebruikt om in de GUI de voorrangsrelaties te laten zien die door chaining toegevoegd zijn. Als op deze manier een chain wordt geprint, wordt deze aan het laatste frame toegevoegd.
- `CLEARSOFTPREC`: Als op een regel enkel `'CLEARSOFTPREC'` geprint wordt, worden alle zogenaamde 'soft precedence constraints' verwijderd. Dit zijn voorrangsrelaties die tijdens het oplossingsproces worden toegevoegd en niet bij de instantie horen. Deze aanroep wordt door het chaining algoritme gebruikt, omdat de voorrangsrelaties van het ESTA$^+$-algoritme overbodig zijn voor chaining. Door het weghalen hiervan, vermindert dus het totaal aantal voorrangsrelaties.
- `FLEX`: Geeft een lijst van EST's en LST's waarmee de flexibiliteitsintervallen bepaald kunnen worden. De syntax is `'FLEX: <minflex> <flextotaal> <variabelen> -1'`. Hierbij geeft `-1` het einde van de lijst aan en is `<variabelen>` een lijst is van EST's en LST's met de syntax `'<projectID> <activiteitID> - <tijd>'` voor de EST en `'<projectID> <activiteitID> + <tijd>'` voor de LST.

5 Opgeleverde Producten en Diensten

In dit hoofdstuk worden de opgeleverde producten en diensten beschreven. Er zal gekeken worden naar nieuwe functionaliteit die toegevoegd is aan zowel de interface als de solver. Daarnaast zullen ook de uitgevoerde diensten, die niet direct een bijdrage leveren aan de functionaliteit van de applicatie, worden toegelicht.

5.1 Solver

De nieuwe functionaliteit van de solver kan grotendeels worden onderverdeeld in het chaining algoritme en het vinden van flexibiliteitsintervals. Van deze onderdelen zal in deze paragraaf toegelicht worden hoe deze geïmplementeerd zijn.

5.1.1 Chaining

Een nieuwe functionaliteit die is toegevoegd aan de solver is het chaining algoritme, zoals geïntroduceerd in paragraaf 3.3.1. Er zijn voor dit algoritme drie verschillende methoden voor het selecteren van een chain geïmplementeerd, namelijk:

- Selecteer een willekeurige geschikte chain.
- Selecteer de eerst gevonden geschikte chain.
- Gebruik de heuristiek zoals beschreven in paragraaf 3.3.1.

Om alle drie methoden te kunnen analyseren, zijn deze allen geïmplementeerd en kan er gemakkelijk gewisseld worden van methode door een kleine aanpassing in de source code te maken. Uiteindelijk is door middel van de analyse van de flexibiliteit en het aantal voorrangsrelaties dat door de algoritmes toegevoegd wordt besloten om de heuristiek zoals beschreven in paragraaf 3.3.1 te gebruiken voor de NedTrain Planner. De analyse van de drie mogelijke heuristieken voor het selecteren van een chain is weergegeven in paragraaf 6.2.1.

Het chaining algoritme is een vereiste voor het berekenen van flexibiliteitsintervallen, omdat dit algoritme ervoor zorgt dat elke oplossing die consistent is met de tijdsconstraints, ook consistent is met de resource constraints. Hierdoor wordt het mogelijk om activiteiten te verschuiven, zonder dat er resource conflicten kunnen ontstaan.

5.1.2 Linear Programming

Om voor elke activiteit een flexibiliteitsinterval te vinden moet er, zoals beschreven in paragraaf 3.3, een LP-probleem opgelost worden. Om dit gemakkelijk te kunnen doen, wordt de CLP[1] (COIN-OR Linear Programming) open source library gebruikt. Deze library verwacht als input een stelsel van lineaire vergelijkingen en een variabele om te optimaliseren en geeft als output de geoptimaliseerde waarde en voor elke variabele de toekenning.

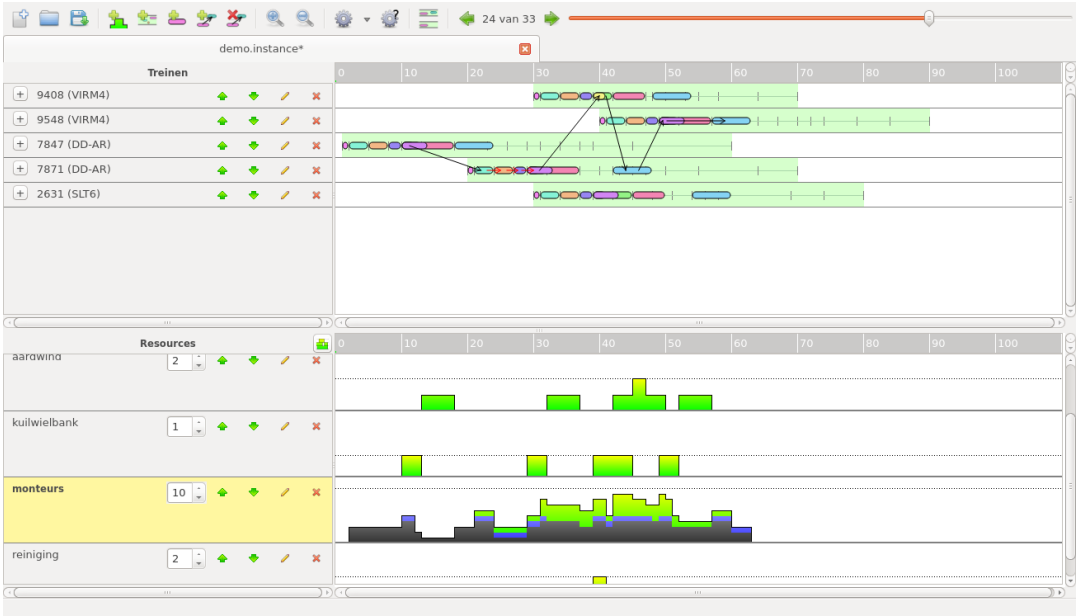
De documentatie van CLP is helaas niet heel erg uitgebreid, maar het is desondanks toch gelukt om het LP-probleem door CLP te laten oplossen. Er moet hiervoor een instantie van de klasse `ClpModel` worden opgebouwd door het aanmaken en bepalen van variabelen en limieten. Er bestaan verschillende instanties van de `ClpModel` klasse. Zo is er door ons gebruik gemaakt van de modellen `ClpSimplex` en `ClpInterior`. Het grote verschil tussen `ClpSimplex` en `ClpInterior` is de manier van oplossen, deze gebruiken namelijk respectievelijk de 'Simplex' en de 'Interior Point' methode. Er is zelf nog een `Constraints` klasse gemaakt om elke variabele een nummer te geven. De variabelen $t_0^+, t_0^-, \ldots, t_n^+$ en t_n^- krijgen respectievelijk de nummers

[1] projects.coin-or.org/Clp

$0, 1, \ldots, 2 \cdot n$ en $2 \cdot n + 1$. Aan het model moet voor elke variabele een onder- en bovengrens worden gesteld. Er moet een objective functie worden opgegeven en ingesteld dat deze gemaximaliseerd moet worden. Vervolgens kan er aan het model constraints worden toegevoegd. Door een solve methode aan te roepen op het model kan er vervolgens met behulp van de variabele nummers de toegekende waarde van de variabele uit het model worden opgevraagd.

5.2 Interface

Er zijn grote en kleine veranderingen aan de interface toegebracht. Kleine veranderingen zijn onder andere het oplossen van bugs die zijn achtergelaten door de vorige projecten die ook aan de NedTrain Planner hebben gewerkt. De grootste veranderingen aan de interface zijn het weergeven van de chains en de flexibiliteitsintervallen. Een voorbeeld van de nieuwe interface kan worden gezien in Figuur 3.



Figuur 3: De gebruikersinterface.

In Figuur 3 is in de rechterbovenhoek een schuifbalk te zien, waarmee de gebruiker door de verschillende stappen van het proces van de solver kan lopen. Voor elke stap is er een frame opgeslagen waarin de toestand van alle activiteiten staan voor dat moment in het proces van de solver. Als de gebruiker de stappen doorloopt, wordt de corresponderende frame getoond met de opgeslagen informatie in de interface. De volgende frames worden opgeslagen bij het uitvoeren van een solver:

- Een frame met de begintoestand van de instantie.
- Voor elke voorrangsrelatie die tijdens het ESTA$^+$ algoritme toegevoegd wordt, een nieuwe frame met daarin alleen de nieuwe voorrangsrelatie en eventuele veranderingen in starttijden van de activiteiten.
- Een frame zonder voorrangsrelaties, waarin geen resource conflicten meer aanwezig zijn. Op dit moment worden alle eerder toegevoegde voorrangsrelaties verwijderd.
- Voor elke resource een frame waarin alle bij deze resource horende chains getoond worden. Vervolgens een aparte frame voor elke chain, die elk bij één resource unit hoort. Voor lege

chains zal er geen frame aangemaakt worden, maar voor chains met maar één activiteit wel.

- Tenslotte een frame waarin voor elke activiteit in het blauw een flexibiliteitsinterval te zien is. Activiteiten worden ook verplaatst, zodat ze zich volledig in het blauwe interval bevinden.

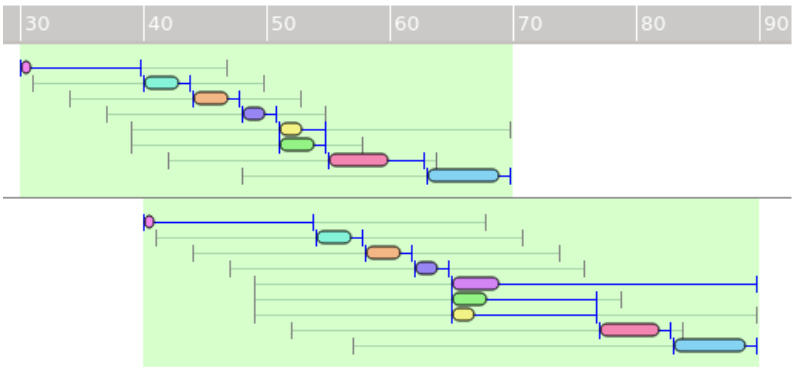
Als een solver geen chains en/of flexibiliteitsintervallen berekent, zullen voor deze stappen geen frames worden toegevoegd en zullen alleen de frames van het ESTA$^+$-algoritme getoond worden.

In het frame dat in Figuur 3 zichtbaar is, wordt een chain van activiteiten, behorende bij de resource 'monteurs', in het bovenste deelvenster getoond. Hierin zijn de rode pijlen voorrangsrelaties die al in de instantie aanwezig waren en zijn de zwarte pijlen later toegevoegd tijdens het oplossen van de instantie.

Zoals eerder genoemd, worden van een resource eerst alle chains tegelijk getoond op een frame en vervolgens elke chain op een apart frame. Tijdens het één voor één laten zien van de chains, wordt het profiel van de gebruikte resources stap voor stap opgebouwd. Op het moment dat er nog geen resources gebruikt worden, is het gehele profiel groen. Bij het tonen van een chain worden alle resource units van een resource die door de activiteiten in deze chain gebruikt worden, in het blauw getoond. Dit betekent dat deze resource units niet meer door andere activiteiten gebruikt kunnen worden. Deze gebruikte resources worden daarom in de daarop volgende frames grijs gekleurd. Zo wordt in elke frame het grijs gekleurde gebied groter, totdat de gehele resource profiel opgevuld is.

5.2.1 Flexibiliteitsintervallen

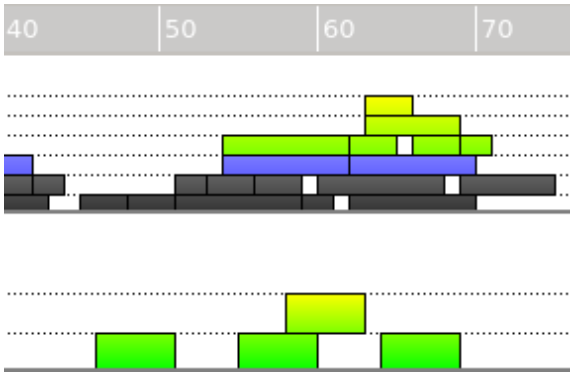
Zoals in Figuur 4 is te zien, is het ook mogelijk om in de interface het flexibiliteitsinterval van elke activiteit weer te geven. Activiteiten kunnen vrij bewegen binnen het flexibiliteitsinterval zonder dat er conflicten ontstaan met andere activiteiten en zonder dat de starttijden van andere activiteiten veranderen. Als de intervallen worden weergegeven, is het niet mogelijk om de activiteiten buiten het bijbehorende interval te verplaatsen. Ook is het niet mogelijk activiteiten zodanig langer te maken dat ze buiten hun flexibiliteitsinterval vallen. Die twee zaken zijn niet mogelijk, om ervoor te zorgen dat het schema flexibel blijft. Als het weergeven van de flexibiliteitsintervallen wordt uitgezet, kunnen de activiteiten wel verplaatst en vergroot worden buiten het flexibiliteitsinterval.



Figuur 4: Flexibiliteitsintervallen

5.2.2 Rooster

Naast het resource profiel dat weergegeven kan worden in de onderste helft van de interface *(zie Figuur 3)*, is er nu ook een nieuwe manier van het weergeven van de resources die worden gebruikt. Deze nieuwe feature maakt het mogelijk de chains, die worden gegenereerd door het chaining algoritme, op een eenvoudig te begrijpen manier weer te geven. De chains van resource units resulteren in een rooster, dat in de onderste helft van de interface kan worden weergegeven *(zie Figuur 5)* met behulp van een nieuwe knop. Vooral als de resource units bijvoorbeeld monteurs, schoonmakers of ander personeel zijn, is deze feature erg handig, omdat de werknemers zelf kunnen zien wie op welk moment aan welke activiteit werkt of pauze heeft.



Figuur 5: Rooster feature

5.2.3 Informatie Trein en Activiteit

Het was al mogelijk om van een activiteit de informatie weer te geven in een klein overzicht. Als er van deze activiteit flexibiliteitsintervallen bekend zijn, worden deze aan het overzicht toegevoegd *(zie Figuur 6)*. Als de instantie nog niet opgelost is of als de instantie is opgelost met een solver die geen flexibiliteitsintervallen ondersteunt, dan worden deze niet in het overzicht weergegeven.

Nu is er ook een overzicht van informatie over een trein beschikbaar. Als de gebruiker met de rechtermuisknop op de tijdlijn van een trein klikt, maar niet op een activiteit, dan verschijnt er een keuzemenu, waarin de optie 'Treininformatie' gekozen kan worden. Daarmee wordt er een nieuw scherm *(zie Figuur 7)* geopend met daarop informatie zoals het aantal activiteiten en de hoeveelheid resources die minimaal voor deze trein nodig zijn. Vanwege de andere voorwaarden aan het probleem betekent het voldoen aan deze minimumwaarden niet altijd dat de instantie oplosbaar is. Andersom geldt wel dat de instantie nooit oplosbaar is als niet aan de minimale benodigde resources voldaan wordt.

Figuur 6: Taakinformatie venster

Figuur 7: Treininformatie venster

5.2.4 Informatie Solver en Oplossing

Na het oplossen van een instantie door de solver worden de uitkomsten en gegevens van de oplossing gecommuniceerd naar de grafische interface. Zo werd er bijvoorbeeld al doorgegeven hoeveel voorrangsrelaties er aan de instantie zijn toegevoegd door de solver. Nu wordt er ook doorgegeven, als de solver gebruik maakt van flexibiliteitsintervallen, wat de totale flexibiliteit van de oplossing is. Als de gebruiker vervolgens aanpassingen maakt en de instantie vervolgens opnieuw laat oplossen, kan er ook meteen gezien worden wat voor effect dit heeft op het aantal voorrangsrelaties en de totale flexibiliteit. Dit wordt gedaan door het kleuren van de tekst en het geven van de nieuwe en de oude gegevens *(zie Figuur 8)*. Een groene kleur van de flexibiliteit tekst geeft aan dat de nieuwe oplossing meer of gelijke flexibiliteit heeft en een rode kleur aangeeft dat de flexibiliteit juist omlaag is gegaan. Hetzelfde geldt voor de tekst van het aantal voorrangsrelaties, waarbij rood aangeeft dat er meer voorrangsrelaties toegevoegd zijn en groen minder of gelijk.

Het is nu ook mogelijk om de tijdsverdeling van de solver na het oplossen te zien. Op deze manier is het meteen duidelijk welk gedeelte van de solver het meest heeft toegedragen aan de totale tijd die de solver nodig had om de instantie op te lossen.

Figuur 8: Venster na oplossen van instantie

5.2.5 Gebruiksgemak

Om het gebruiksgemak te vergroten, is er een aantal sneltoetsen toegevoegd. Ook is het sluiten van instanties veranderd, want waar eerst instanties gesloten konden worden met een grote rode kruis in de menubalk, is er nu op elke tab van een instantie een klein kruisje geplaatst, waarmee de instantie gesloten kan worden. Dit ontwerp ziet er intuïtief uit en wordt ook gebruikt door veel bekende browsers zoals Google Chrome en Mozilla FireFox.

Er zijn ook sneltoetsen toegevoegd voor het in- en uitzoomen en voor het horizontaal scrollen. Er kan in- en uitgezoomd worden door de ctrl-toets ingedrukt te houden en het scrollwiel van de muis te bewegen. Horizontaal scrollen kan in het venster waarin de muis zich op dat moment bevindt door de shift-toets ingedrukt te houden en het scrollwiel van de muis te bewegen. Deze sneltoetsen zijn ingesteld zodat de gebruiker makkelijker door een probleeminstantie kan navigeren, zonder daarvoor steeds zijn muis naar de menubalk te hoeven verplaatsen.

Nog een feature die voor een verbeterd gebruiksgemak moet zorgen bestaat uit de knopjes voor het van plek verwisselen van taken en resources. Dit is vooral handig als er een nieuwe activiteit of resource wordt toegevoegd, want deze wordt altijd onderaan gezet.

Tijdens het doorlopen van de stappen van de solver worden de treinen en resources opvallend gekleurd als hier een aanpassing in gedaan wordt door de solver. Zo is het bijvoorbeeld gemakkelijk bij de frames van het chaining algoritme te zien voor welke resource het getoonde chain geldt. Daarnaast verspringt het beeld ook naar de resource zodat de gebruiker niet door de lijst van resources hoeft te zoeken.

Tenslotte nog een feature die ook prettig is voor gebruikers, maar niet visueel is. Het is mogelijk de applicatie zo te compileren, zodat een gebruiker met hetzelfde besturingssysteem de applicatie meteen kan gebruiken zonder extra software te installeren. Deze manier van compileren wordt ook wel *static builden* genoemd. In de *static build* zitten alle afbeeldingen, icoontjes, fonts en externe libraries meegeleverd in de executable. Dit maakt het installeren van de applicatie voor de gebruiker gemakkelijker, aagezien de gebruiker geen zorgen hoeft te maken over extra benodigde software en libraries.

5.3 Codekwaliteit

Om de codekwaliteit te verbeteren zijn er een aantal aanpassingen aan de code gemaakt. Deze veranderingen zorgen dus niet voor nieuwe features, maar voor een betere onderhoudbaarheid van de code, zodat ook ontwikkelaars na dit project makkelijk nieuwe features kunnen toevoegen. Zo zijn alle resources (zoals icoontjes, fonts en vertalingen) aan een speciaal bestand, genaamd `Resources.qrc`, toegevoegd, dat bijhoudt van welke resources de interface gebruik maakt. Door het gebruik van deze file hoeft er geen zorgen gemaakt te worden over waar de bestanden staan op het systeem. Ze worden mee gecompileerd in de code, wat ook nodig is voor het static builden van de applicatie, eerder besproken in paragraaf 5.2.5.

Daarnaast is er ook nog veel code verbeterd en verschoond. Zo was er nog veel typische C code aanwezig, wat niet gebruikelijk is in C++ en vaak waarschuwingen gaf tijdens het compileren. In deze gevallen is zo veel mogelijk geprobeerd om de code te laten voldoen aan C++ conventies. Daarbij zijn het gebruik van een `boolean` in plaats van een `integer` en een `string` in plaats van een `char*` goede voorbeelden. Ook is er een betere scheiding gemaakt tussen de header en de source files door het includen van libraries op de goede plaats te zetten.

5.4 Port naar Windows 7

Om de toegankelijkheid van de applicatie te vergroten, is er voor gezorgd dat deze ook op het besturingssysteem Windows 7 werkt. Hierbij is het ook nog steeds mogelijk om deze op Unix-based systemen te draaien, zoals Ubuntu. Alle functies van het programma moeten dus zowel op Windows 7 als op Ubuntu correct werken.

5.5 Upgrade naar Qt 5.2

De bestaande applicatie was ontwikkeld in versie 4.8 van het Qt framework, maar om de applicatie zo up-to-date mogelijk te houden, is deze geport naar Qt versie 5.2. Hierdoor hoeft een eventuele groep, die de applicatie verder gaat ontwikkelen, niet met een oude versie van het Qt framework te werken. Deze upgrade betekent echter ook dat de applicatie niet meer werkt met Qt 4, maar alleen met versie 5.

5.6 Behouden van bestaande functionaliteit

Bij het ontwikkelen van de nieuwe solver is er rekening mee gehouden worden dat alle bestaande functionaliteit behouden blijft. De interface moet dus blijven werken met de bestaande solver en alle functionaliteiten die in de interface geïmplementeerd zijn, moeten kunnen functioneren op flexibele schema's. Denk hierbij in het bijzonder aan het kunnen vergrendelen van taken, het kunnen instellen van de flexibiliteit en het kunnen verlagen van de resource capaciteit op bepaalde intervallen.

6 Performance

Om de gebruikte oplossingsmethoden te verantwoorden, zullen deze in dit hoofdstuk vergeleken worden met andere mogelijke methoden. Zowel de looptijd van het oplossen van verschillende probleeminstanties als de flexibiliteit die met verschillende methoden bereikt wordt zullen geanalyseerd worden.

6.1 Looptijd

In dit hoofdstuk zal een analyse van de looptijd van de solver uitgevoerd worden. Aangezien de opdrachtgever de applicatie voornamelijk gebruikt voor onderzoek en educatie zullen de probleeminstanties kleiner zijn dan in de realiteit. Bij deze analyse wordt de solver opgedeeld in vijf belangrijke delen: parsen van input, STJN, ESTA$^+$, chaining en LP. Om de looptijd te bepalen wordt er gebruik gemaakt van een grote probleeminstantie genaamd 'xxl-instance'. Deze probleeminstantie bestaat uit:

- 10 resources, met elk 1 resource unit
- 180 projecten, met elk tussen de 4 en de 6 activiteiten
- 912 activiteiten, die elk gebruik maken van tussen de 1 en 5 resources

In Tabel 1 en Tabel 2 is de duur van de verschillende onderdelen van de solver te vinden. De totale looptijd van de solver zou nog verlaagd kunnen worden door de hoeveelheid informatie die naar de buffer `stdout` wordt gestuurd te verlagen. Dit zou voor de xxl-instantie rond de 10 seconden aan tijdwinst op kunnen leveren. Een mogelijkheid om deze hoeveelheid informatie te verlagen is om niet telkens alle informatie van een nieuwe frame te printen, maar alleen het verschil met het vorige frame. Een andere mogelijkheid is om niet met een aparte solver te werken die wordt aangeroepen door de NedTrain Planner, maar de solver te integreren in de NedTrain Planner. Hierdoor hoeft er geen communicatie te gebeuren tussen de solver en interface via een buffer, maar dit heeft wel als nadeel dat er dan niet gemakkelijk een nieuwe solver toegevoegd kan worden aan de NedTrain Planner.

Het onderdeel Linear Programming, die de flexibiliteitsintervallen berekent, heeft het grootste aandeel in de totale looptijd. In de huidige solver wordt de Interior Point methode gebruikt voor het oplossen van LP-problemen. Deze is langzamer dan de Simplex methode, maar verdeelt de flexibiliteit wel beter over alle activiteiten. Uit metingen is gebleken dat de Simplex methode nagenoeg geen tijd kost, dus dit zou een grote snelheidswinst kunnen opleveren. De consequenties van het gebruik van de Simplex methode worden besproken in paragraaf 6.2.2 Linear Programming.

	Looptijd (s)	Percentage (%)
Parsing	$0,132556$	$0,3$
STJN	$0,003002$	$0,0$
ESTA$^+$	$11,447062$	$23,3$
Chaining	$1,109195$	$2,3$
LP	$36,464763$	$74,2$
Totaal	$49,156718$	$100,0$

Tabel 1: xxl-instance met output

	Looptijd (s)	Percentage (%)
Parsing	$0,009821$	$0,0$
STJN	$0,002505$	$0,0$
ESTA$^+$	$1,533199$	$3,9$
Chaining	$0,843122$	$2,6$
LP	$36,445771$	$93,8$
Totaal	$38,834843$	$100,0$

Tabel 2: xxl-instance zonder output

6.2 Flexibiliteit

Om ervoor te zorgen dat de schema's die door de solver berekend worden zo flexibel mogelijk zijn, worden er verschillende methoden geanalyseerd voor de onderdelen van het proces dat de solver doorloopt. De nieuwe onderdelen hiervan zijn het chaining algoritme en het berekenen van flexibiliteitsintervallen en zullen dus in deze paragraaf geanalyseerd worden.

6.2.1 Chaining

Een belangrijk deel van het chaining algoritme, dat is geïntroduceerd in paragraaf 3.3.1, is de methode $SelectChain(v_i, r_j)$. Voor deze methode zijn ook in paragraaf 3.3.1 drie verschillende oplossingen genoemd: het kiezen van een willekeurige chain, het kiezen van de eerste geschikte chain en het gebruiken van de heuristiek beschreven in paragraaf 3.3.1. Deze drie methoden zullen nu vergeleken worden op het gebied van flexibiliteit en het aantal toegevoegde voorrangsrelaties. Het is daarbij het best om het aantal voorrangsrelaties te minimaliseren en de flexibiliteit te maximaliseren.

Er is gebruik gemaakt van vier verschillende verzamelingen instanties: 480 J30-instanties, 480 J60-instanties, 480 J90-instanties en 600 J120-instanties. Deze instanties bevatten respectievelijk 30, 60, 90 en 120 projecten met in totaal 64, 124, 184 en 244 taken. De resultaten van deze analyse zijn te zien in Tabel 6.2.1, waarbij cellen onder V het gemiddeld aantal toegevoegde voorrangsrelaties bevatten en cellen onder F de gemiddelde flexibiliteit.

	J30		J60		J90		J120	
	V	F	V	F	V	F	V	F
Willekeurige chain	92,2	527,5	257,4	1829,5	453,1	3744,7	585,4	4002,4
Eerste chain	47,5	543,2	119,8	1898,3	208,7	3789,6	300,2	4051,6
Heuristiek	47,6	542,8	120,0	1900,2	199,1	3894,0	300,4	4211,4

Tabel 3: Resultaten van de analyse van de drie verschillende $SelectChain$ methoden.

Het is duidelijk te zien dat het kiezen van een willekeurige chain slechter is dan de twee andere methoden, omdat hierbij voor elke verzameling instanties het aantal voorrangsrelaties groter is en de flexibiliteit kleiner. Om de resterende twee methoden te vergelijken, wordt er gekeken naar de procentuele verbetering ten opzichte van het kiezen van een willekeurige chain. De resultaten hiervan zijn weergegeven in Tabel 4.

	J30		J60		J90		J120	
	V	F	V	F	V	F	V	F
Eerste chain	48,5 %	3,0 %	53,5 %	3,8 %	53,9 %	1,2 %	48,7 %	1,2 %
Heuristiek	48,4 %	2,9 %	53,4 %	3,9 %	56,1 %	4,0 %	48,7 %	5,2 %

Tabel 4: Procentuele verbetering van de bovenstaande methoden ten opzichte van het selecteren van een willekeurige chain.

Het blijkt uit Tabel 4 dat de 'Eerste chain'-methode en de heuristiek voor kleine instanties, J30 en J60, nauwelijks in resultaat verschillen. Het verschil tussen beide methoden is voor deze instanties maar $\pm 0.1\%$ ten opzichte van de 'Willekeurige chain'-methode. Bij grotere instanties,

J90 en J120, wordt er echter wel een duidelijk verschil in de flexibiliteit merkbaar. Hierbij is de flexibiliteit van de heuristiek bij J90-instanties gemiddeld 2.8% en bij J120- instanties 4.0% beter. Het aantal voorrangsrelaties dat toegevoegd wordt, is bij de J120-instanties echter wel praktisch gelijk.

Hoewel het verschil tussen de 'Eerste chain'-methode en de heuristiek erg klein is, wordt de flexibiliteit van de heuristische methode wel groter naarmate de grootte van de instantie toeneemt. Aangezien NedTrain veel te maken heeft met instanties die zeker zo groot zijn als de J20-instanties, is dit dus een positieve eigenschap van de heuristiek. Er is uiteindelijk dus gekozen om deze heuristiek te gebruiken, maar ook de andere methoden zijn geïmplementeerd en hierop kan gemakkelijk overgeschakeld worden door een simpele aanpassing in de source code.

6.2.2 Linear Programming

De verdeling van de flexibiliteit over alle activiteiten kan gemeten worden met de *Mean Squared Error* (MSE) van de flexibiliteit $MSE(flex)$ die gedefiniëerd staat in Formule 6.2.2. Hoe lager deze MSE is, hoe beter de flexibiliteit is verdeeld over alle activiteiten.

$$\begin{aligned} flex_{gem} &= \sum_{t \in T} \frac{t^+ - t^-}{|T|} && \text{met } |T| \text{ het aantal elementen in } T \\ MSE(flex) &= \sum_{t \in T} (flex_t - flex_{gem})^2 \\ &= \sum_{t \in T} (t^+ - t^- - flex_{gem})^2 \end{aligned} \tag{7}$$

Er is met 2040 probleeminstanties getest welke van de vier methoden de beste verdeling van flexibiliteit levert en welke de grootste totale flexibiliteit heeft. In Tabel 5 wordt in elke cel weergegeven in hoeveel procent van de gevallen de 'rij' beter is dan de 'kolom'. Hierbij wordt de Interior Point methode afgekort tot IP en de Simplex methode tot S. Met methode 1 wordt het gebruik van het LP-probleem in Formule 4 aangeduid en met methode 2 het gebruik van het LP-probleem in de Formules 5 en 6.

	IP_1	IP_2	S_1	S_2
IP_1	×	$0,0\%$	$88,5\%$	$0,5\%$
IP_2	$99,9\%$	×	$100,0\%$	$88,9\%$
S_1	$10,5\%$	$0,0\%$	×	$0,1\%$
S_2	$99,4\%$	$10,7\%$	$99,9\%$	×

Tabel 5: Vergelijking verdeling flexibiliteit

Met behulp van Tabel 5, waar de verdeling van flexibiliteit gemeten is met Formule , kan worden bepaald welke methode de beste verdeling van flexibiliteit heeft. Hieruit volgt dat IP_2 de beste verdeling van flexibiliteit heeft, gevolgd door van hoog naar laag S_2, IP_1 en S_1.

De totale flexibiliteit is ook een belangrijk aspect waarmee de prestatie van de solver gemeten kan worden. Bovendien vindt de opdrachtgever het belangrijk dat de totale flexibiliteit niet lijdt onder de verdeling hiervan. Aangezien de totale flexibiliteit van methode 1 bij elke probleeminstantie groter of gelijk aan de totale flexibiliteit van methode 2 is, wordt er gekozen voor methode 1 in combinatie met de Interior Point methode (IP_1). Hiermee wordt niet de beste verdeling van flexibiliteit bereikt, maar wel een maximale totale flexibiliteit.

7 Ontwikkelproces

Dit hoofdstuk beschrijft hoe het ontwikkelproces van het project is verlopen. Er wordt verteld welke ontwikkelmethoden en wat voor hulpmiddelen er gebruikt zijn tijdens dit project en wat onze ervaringen daarbij waren. Het hoofdstuk wordt afgesloten met feedback van de Software Improvement Group en wat er met die feedback gedaan is.

7.1 Ontwikkelmethode

Gedurende het hele project is er zo veel mogelijk gebruik gemaakt van de Scrum principes, zoals beschreven in *The Scrum Guide* [7]. De implementatiefase is in 6 sprints verdeeld van elk een week. Voorafgaand aan elke week is er vastgesteld wat de taken en doelen voor die week zijn. Meestal waren features aan het eind van een sprint geschikt voor een mogelijke demo, maar zat er nog wel een aantal bugs in. Het meest voorkomende probleem was dat een feature aan het einde van de sprint nog niet werkte op een Windows systeem. Voor elke sprint zal een korte beschrijving gegeven worden en een kleine terugblik op hoe de sprint is verlopen en wat onze ervaringen waren.

7.1.1 Sprint 1

Het doel van de eerste sprint was om het chaining algoritme te implementeren en bekend te raken met de bestaande software. Daarnaast was een begin maken van het porten van de NedTrain Planner naar Windows 7 ook een doel in de eerste sprint, aangezien tijdens de onderzoeksfase gebleken was dat dit in een redelijke tijd uitgevoerd zou moeten kunnen.

Terugblik
We zijn erg tevreden over het verloop van de eerste sprint. Het porten naar Windows 7 liep met een paar kleine tegenvallers redelijk vlot. Naast het porten naar Windows 7 is het ook in deze sprint gelukt om de versie van Qt te upgraden naar versie 5.2.1. De basis van ons chaining algoritme werkt, maar de samenwerking tussen de GUI en de solver loopt nog niet zoals gewenst. Er is zelfs ook al een begin gemaakt aan de LP solver, een doel dat voor de tweede sprint gepland stond.

7.1.2 Sprint 2

Het voornaamste doel van de tweede sprint is de COIN-OR LP solver te laten samenwerken met onze applicatie. Daarnaast was het laten zien van aparte chains met daarbij welke resources er door deze chain gebruikt worden ook een doel voor deze sprint.

Terugblik
In deze sprint hebben we een aantal tegenvallers gehad. Zo hadden we last van een memory leak, wat redelijk veel tijd kostte om op te lossen. Bovendien moest de bestaande architectuur voor het laten zien van chains grotendeels aangepast worden. Een tweede tegenvaller was het updaten van de software op de Jenkins-server. Aangezien de tools geüpgrade zijn naar Qt versie 5.x, moet op de server Qt ook worden geüpgraded. Dit ging niet meteen de eerste keer goed, waarbij uiteindelijk is overgegaan tot een volledige herinstallatie. Dit heeft als gevolg dat een deel van de eerste commits mist in de continuous integration grafiek. In deze sprint is er ook nog verder gewerkt aan het porten van de applicatie naar Windows. We merkten dat kleine aanpassingen in de code om features te laten werken voor Linux, zorgde voor nieuwe Windows gespecificeerde bugs. Hierdoor duurde het implementeren van nieuwe features langer dan normaal, omdat de applicatie voor beide systemen moet werken.

7.1.3 Sprint 3

In deze sprint stond een demo geven aan de opdrachtgevers op het programma. Hiermee kon er gecontroleerd worden of de tot dan toe geïmplementeerde features naar wens waren volgens de opdrachtgevers. Hierdoor was het mogelijk om nog het één en ander aan te passen in deze en volgende sprints. Verder was deze week beschikbaar gemaakt voor uitloop en het verfijnen van de gemaakte features.

Terugblik
Het is in deze sprint gelukt om per chain te laten zien welke resources er door de activiteiten in deze chain gebruikt worden. Elke chain kan apart op een frame getoond worden, waarbij de resource units die door deze chain gebruikt zijn blauw gekleurd worden. De resource units die door eerdere chains van deze resource gebruikt zijn, zijn zwart gekleurd. Dit is tijdens de demo gedemonstreerd. De feedback die hierover gegeven was, was dat er niet heel makkelijk gezien kon worden welke taken bij een bepaalde resource unit hoorde. Om dit duidelijker te maken, zou een aparte view gemaakt kunnen worden. Hierin is echter minder makkelijk te zien hoeveel resources er op een bepaald tijdstip gebruikt worden.

De CLP solver is in deze sprint ook veel verder gevorderd. Er bestaat niet meer een aparte binary voor de CLP solver, maar deze is geïntegreerd in de binary van de chaining solver. Er kan een willekeurige LP instantie als input gegeven worden en deze kan door de solver opgelost worden. Wat nu nog overblijft is het transformeren van een RCPSP instantie naar een LP instantie, zodat deze opgelost kan worden.

7.1.4 Sprint 4

In deze sprint was het doel om de flexibiliteitsintervallen, die worden berekend met behulp van de LP-solver, door te geven aan de gebruikersinterface. Ook was het belangrijk om dit op een goede manier weer te geven, zodat de gebruiker deze informatie op een juiste manier kan interpreteren. Tenslotte wilden we de tool zo compileren, dat de gebruiker het gemakkelijk kan installeren. Hiervoor moest de tool statisch gelinkt worden met de gebruikte libraries. Op deze manier kan de applicatie uitgevoerd worden, zonder dat er nog andere software geïnstalleerd hoeft te worden.

Terugblik
Het is ons gelukt om in deze sprint de flexibiliteitsintervallen weer te geven op het scherm, maar de gebruiker kan dit nog niet aan- en uitzetten. Het verder doorontwikkelen van de features rondom de flexibiliteitsintervallen krijgt een hoge prioriteit in de volgende sprint. De NedTrain Planner kan nu zeer gemakkelijk geïnstalleerd worden op Windows 7, maar voor systemen met Linux werkt dit helaas nog niet. Het statisch linken van de applicatie is tijdrovend, aangezien de applicatie van veel dingen afhankelijk is.

7.1.5 Sprint 5

In deze sprint was het doel om de feature van het weergeven van de flexibiliteit af te maken en te testen. Aan het einde van deze sprint moest de code worden opgestuurd naar SIG en moest er een eerste versie van het eindverslag af zijn.

Terugblik
We hebben in deze sprint veel tijd moeten besteden aan het verslag, maar we hebben gelukkig ook tijd kunnen besteden aan het verder door ontwikkelen van de NedTrain Planner. In deze sprint hebben we de code kwaliteit verbeterd en veel bugs ontdekt die de vorige projecten hebben achter gelaten. De meeste bugs zijn in deze sprint verholpen.

7.1.6 Sprint 6

Aan het begin van deze sprint werd er weer een demo gegeven, zodat de opdrachtgever kon zien hoe het er met de product voor staat. De feedback die verkregen waren tijdens de demo zijn als doelen opgesteld voor deze sprint. Daarnaast moest er in deze sprint ook gekeken worden naar de efficiëntie van de geïmplementeerde algoritmes, aangezien bij grote instanties de interface vastliep.

Terugblik
Na de demo is er besloten om nog één laatste feature te implementeren, omdat dit voor de opdrachtgever een vrij hoge prioriteit had. Dit is de rooster feature, die de werking van het chaining algoritme duidelijker maakt. Daarnaast is er duidelijker gemaakt welke resources aangepast worden in een stap van de solver, door de desbetreffende resources te highlighten. Het vastlopen van de interface is ook verholpen in deze sprint. De algoritmes en de communicatie tussen de solver en de interface zijn efficiënter gemaakt. Verder zijn er weer behoorlijk wat bugs opgelost en is het verslag afgemaakt.

7.2 Hulpmiddelen

Zoals beschreven in het Plan van Aanpak (Bijlage B) en het Oriëntatieverslag (Bijlage C) is er voor een aantal hulpmiddelen gekozen om te gebruiken tijdens het project. In deze paragraaf wordt onze ervaring van het gebruik van deze hulpmiddelen besproken, maar ook een aantal resultaten daarvan.

7.2.1 Versiebeheer

Als versiebeheersysteem is voor dit project een Git repository gebruikt op de website Bitbucket.com. Er was hiervoor gekozen omdat de projectleden het meest bekend waren met Git. Ondanks dat is er nog veel bijgeleerd en gaat het gebruik ons steeds makkelijker af. Het is gelukt om altijd een werkende master branch te hebben. Dit heeft als grote voordeel dat er op elk moment tijdens het ontwikkelproces een demo gehouden kan worden. Ook is dit belangrijk om de tests op de continuous intregration server te laten slagen. Doordat de tool moest blijven werken op zowel Windows 7 als Linux, werden nieuwe features relatief laat aan de master branch toegevoegd. Vaak werd een nieuwe feature op een systeem met Linux geïmplementeerd en pas als het af was op een systeem met Windows 7 getest, om daar vervolgens de problemen te verhelpen.

7.2.2 Continuous Integration

Tijdens het project was het plan om continuous integration toe te passen door gebruik te maken van een Jenkins server. Deze server werd zelfstandig beheerd en stond fysiek bij één van de projectleden thuis. Het kostte erg veel tijd om alle software, die nodig was voor het compileren van dit project, te installeren. Zoals vermeld staat in paragraaf 'Versiebeheer', is er relatief laat nieuwe features toegevoegd aan de master branch. Dit heeft als gevolg dat de master branch weinig verandert en het daardoor minder nuttig is om elke dag de applicatie te compileren en te testen. Ook zijn er problemen ontstaan bij het upgraden naar Qt versie 5. Het upgraden van de software op de server ging gepaard met veel problemen en er is uiteindelijk overgegaan tot herinstallatie van de server. In sprint 5 zijn er wederom problemen ontstaan toen de server een keer opnieuw werd opgestart, waardoor de build geschiedenis kwijt is geraakt. Al met al heeft het toepassen van continuous integration veel tijd gekost en weinig opgeleverd.

7.2.3 Google Test

Het project beschikte aan het begin van het project al over een flinke hoeveelheid tests, maar er waren nog veel klassen waar helemaal geen tests voor waren gemaakt door eerdere projecten. Voor deze klassen en voor nieuwe features zijn door ons tests gemaakt. De solver beschikte over helemaal geen tests. Een uitgebreide testsuite is ook belangrijk voor de opdrachtgever, zodat volgende ontwikkelaars gemakkelijk op dit project kunnen voortbouwen zonder ongemerkt onderdelen kapot te maken.

Alle tests die gemaakt zijn, maken gebruiken van het Google Test framework. Dit was erg prettig in het gebruik en het uitbreiden van de testsuite ging redelijk eenvoudig. Er is besloten om voor de solver een aparte testsuite te maken en deze niet te integreren met de bestaande testsuite voor de interface. Dit heeft als voordeel dat de testsuites net als de interface en solver zelf apart gebruikt kunnen worden. Voor de solver zijn er 42 tests gemaakt.

Om de grootte van de testsuite te controleren, is er gebruik gemaakt van de *Code Coverage Tool* `gcovr`. Deze tool geeft per bestand aan hoeveel regels code en branches er zijn uitgevoerd door de testsuite. Deze tool stimuleerde ons om meer testcases te schrijven, om daarmee een hogere dekking te bereiken. De resultaten van de analyse van deze tool zijn te vinden in Tabel 6. Het percentage branches dat wordt getest door de testsuite is met een gemiddelde van $56,1\%$ aan de lage kant. Het percentage van regels code dat wordt getest door de testsuite is met een gemiddelde van $76,8\%$ wel redelijk.

Bestand	Regels (%)	Branches (%)
`chaining.cpp`	87,2	58,1
`constraints.cpp`	100,0	56,6
`esta_plus.cpp`	69,3	56,0
`exceptions.cpp`	62,5	50,0
`flexibility.cpp`	99,0	53,2
`heap.cpp`	54,7	58,3
`output.cpp`	68,8	50,0
`solve.cpp`	65,2	50,0
`stjn.cpp`	68,9	57,8
`timing.cpp`	79,6	63,6
`tmsp.cpp`	95,5	55,9
Gemiddelde	76,8	56,1

Tabel 6: Code coverage van Solver

Ook voor de NedTrain Planner is er een code coverage analyse gedaan. De resultaten daarvan staan in Tabel 7. Vooral de mappen *controller*, *widgets* en *dialogs* bevatten veel code met een lage dekking. Dit komt doordat het testen van interfaces niet gemakkelijk is en veel tijd kost, zo stelt ook de vorige groep ontwikkelaars [1]. Doordat de solver nog geen testsuite had en daar ook veel code bij is gemaakt tijdens dit project, is er vooral gefocust op een goede testsuite voor de solver, dit ten nadele van de testsuite voor de interface.

Bestand	Regels (%)	Methoden (%)
`controller`	29,4	40,2
`data`	37,3	47,7
`model`	63,1	65,7
`solve`	45,1	54,3
`util`	60,2	62,2
`widgets`	46,3	38,1
`widgets/dialogs`	6,9	11,4
Gemiddelde	41,8	41,0

Tabel 7: Code coverage van NedTrain Planner

7.2.4 C++ en Qt

Voor de ontwikkeling van de applicatie werd er gebruik gemaakt van C++ in combinatie met het Qt framework. Wij hadden vrij weinig ervaring met C++ maar doordat deze programmeertaal gebruikt moest worden tijdens dit project is er veel bijgeleerd over C++Het debuggen van C++-code werd wel lastiger bevonden dan bijvoorbeeld bij een programmeertaal als JAVA. Vooral *Segmentation faults* kwamen vaak voor, waarbij er weinig informatie werd gegeven over wat er precies fout ging. Ondanks de moeilijkheden zouden wij C++ wel willen gebruiken bij volgende projecten, vooral als een goede performance een eis van de opdrachtgever is.

Niemand van de projectleden had ooit eerder met Qt gewerkt, maar dit zouden we zeker vaker willen gaan gebruiken. Het is prettig werken met Qt en het levert een grafisch mooie gebruikersinterface op. Het was ook nuttig dat de applicatie voor een gedeelte was geïmplementeerd, omdat dan sneller duidelijk was hoe bepaalde functies werken in Qt. Aan de andere kant was het ontdekken en verhelpen van bugs vervelend, omdat de code al een redelijke omvang had en voor ons onbekend was.

7.3 Software Improvement Group

Aan het eind van sprint 5 is al onze code opgestuurd naar de Software Improvement Group, die op de code feedback en een beoordeling van 1 tot 5 sterren geeft. De code wordt op de volgende onderdelen beoordeeld [8]:

- **Volume** - Hoe meer regels code een applicatie bevat, hoe moeilijker het wordt om deze te onderhouden.
- **Duplication** - Het kopiëren van code moet zo veel mogelijk vermeden worden.
- **Unit size** - Het grootste deel van de applicatie moet bestaan uit units van maximaal 20 regels.
- **Unit complexity** - De cyclomatische complexiteit, gemeten d.m.v. het *McCabe complexity number*, moet voor het grootste deel van de code niet hoger zijn dan 5.
- **Unit interfacing** - Het meegeven van meer dan 2 parameters moet zo veel mogelijk vermeden worden.
- **Module coupling** - Het sterk gekoppeld zijn van verschillende groepen units moet vermeden worden, omdat dit de software moeilijker te onderhouden maakt.
- **Component balance** - Alle componenten moeten ongeveer even groot zijn. Dit onderdeel wordt beoordeeld d.m.v. de *adjusted Gini coefficient*.
- **Component independence** - Er moet gestreefd worden naar zo veel mogelijk componenten die niet aangeroepen worden door componenten van andere modules. Alleen interface

klassen zouden de communicatie tussen verschillende modules moeten regelen.

In de volgende paragrafen zal het ontvangen feedback van SIG en de handelingen die zijn uitgevoerd door ons op basis van de feedback gegeven worden.

7.3.1 Feedback SIG

Over de code die op vrijdag 13 juni 2014 is opgestuurd, is de volgende feedback gekregen.

> De code van het systeem scoort bijna 4 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Module Coupling, Unit Size en Unit Complexity.
>
> Voor Module Coupling wordt er gekeken naar het percentage van de code wat relatief vaak wordt aangeroepen. Normaal gesproken zorgt code die vaak aangeroepen wordt voor een minder stabiel systeem omdat veranderingen binnen dit type code kan leiden tot aanpassingen op veel verschillende plaatsen. In dit systeem wordt bijvoorbeeld het relatief grote bestand 'instance_misc' op ruim 160 verschillende plaatsen gebruikt. Een ander voorbeeld is 'instance_manip.cpp', 1 van de grootste files in het systeem welke op ruim 60 plaatsen gebruikt wordt. Het lijkt erop dat deze files verschillende functionaliteiten bevatten, daarnaast is het uit de naamgeving niet duidelijk welke functionaliteit waar geïmplementeerd zou moeten zijn. Om zowel de grootte als het aantal aanroepen te verminderen zouden deze functionaliteiten gescheiden kunnen worden, wat er ook toe zou leiden dat de afzonderlijke functionaliteiten makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden worden.
>
> Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methoden in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methoden in dit systeem, zoals bijvoorbeeld de 'esta_plus'-methode, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methoden. De body van condities of de 'cleanup' code zijn hier voorbeelden van. Het is aan te raden kritisch te kijken naar de langere methoden binnen dit systeem en deze waar mogelijk op te splitsen.
>
> Voor Unit Complexity wordt er gekeken naar het percentage code dat bovengemiddeld complex is. Ook hier geldt dat het opsplitsen van dit soort methoden in kleinere stukken ervoor zorgt dat elk onderdeel makkelijker te begrijpen, makkelijker te testen en daardoor eenvoudiger te onderhouden wordt. In dit geval komen de meest complexe methoden ook naar voren als de langste methoden, waardoor het oplossen van het eerste probleem ook dit probleem zal verhelpen.
>
> Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase. De aanwezigheid van test-code is in ieder geval veelbelovend, hopelijk zal het volume van de test-code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.
>
> Software Improvement Group

7.3.2 Verbeteringen

Aan de hand van de feedback van SIG zijn er vooral aanpassingen gedaan aan code van vorige projecten. Zo zijn onder andere de lange en complexe methoden in `esta_plus.cpp` en `solver.cpp` opgegedeeld in meerdere methoden.

In de code van de applicatie zijn ook meerdere methoden opgesplitst. Zo is er nu voor elke klasse die een grafisch venster aanmaakt een methode toegevoegd die de layout van het venster opbouwt, in plaats van dit allemaal in de al behoorlijk lange constructor te doen. Alhoewel deze methode nog steeds bovengemiddeld lang is voor sommige klassen, is deze niet verder opgesplitst, aangezien dit niet de leesbaarheid van de code ten goede zou komen. Deze code is goed uitgelijnd en het is meteen duidelijk wat het doet.

Aan 'Module Coupling' is vrijwel niets gedaan, omdat voor het verlagen van de 'Module Coupling' grootschalige aanpassingen gedaan moesten worden aan het project en hiervoor niet meer genoeg tijd beschikbaar was. De methoden in `intance_manip.cpp` en `instance_misc.cpp` worden veel aangeroepen, wat wordt veroorzaakt doordat deze klasse alle informatie bewaart van een instantie. Als in de interface een instantie geopend wordt, moet alle informatie over die instantie worden bijgehouden. Aangezien een instantie één geheel is, is het ook niet logisch om deze nog verder op te delen. Het is al opgedeeld in twee bestanden, om de coupling te verlagen.

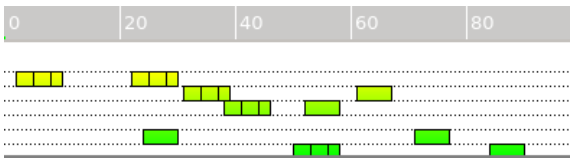
8 Aanbevelingen

De vorige bacheloreindprojectgroep beveelde een undo-functionaliteit aan voor de applicatie [1]. Wij zien ook zeker het voordeel van een undo-functionaliteit in, omdat bij het gebruik wel eens iets per ongeluk verwijderd of veranderd wordt. Aangezien de NedTrain Planner nu meer wordt gebruikt voor onderzoek en educatie, wordt de focus toch gelegd op andere belangrijke features. Als deze tool gebruikt gaat worden door NedTrain, is een undo-functionaliteit een must-have. We hebben onderzocht hoe deze feature gemaakt zou kunnen worden. Een voordeel is dat Qt een Undo Framework heeft, maar om dit te kunnen gebruiken moet er veel aan de code veranderd worden.

Vervolgens raden we ook het verbeteren van de kwaliteit van de code aan. Bij het implementeren van nieuwe functionaliteiten hebben we namelijk soms gemerkt dat de huidige structuur van de applicatie niet ideaal was. Voor de bestaande functionaliteiten was de structuur wel geschikt, maar dus niet altijd voor de nieuwe features. Voor het toevoegen van sommige nieuwe functionaliteiten zou het beter zijn om de structuur van het gehele programma te veranderen, maar vanwege beperkte tijd is dit niet uitgevoerd.

Als derde aanbeveling zou het goed zijn om de communicatie tussen de solver en de interface te verbeteren. De grootste snelheidswinst is nu te behalen in de communicatie tussen de NedTrain Planner en de solver. Er kan worden gekeken naar de uitvoer van de solver tijdelijk naar een bestand te schrijven en deze daarna in te lezen door de NedTrain planner. Ook kan de hoeveelheid informatie die wordt verstuurd van de solver naar de NedTrain Planner worden gereduceerd. Van elk frame dat wordt getoond door de interface, worden alle starttijden van de activiteiten door de solver uitgeprint, ook al verandert er maar één activiteit. Door steeds het verschil tussen het huidige en het vorige frame door te geven, valt een relatief grote snelheidswinst te behalen.

Een vierde aanbeveling is een feature toe te voegen waarbij de resources zo eerlijk mogelijk verdeeld worden over de resource units. Dit is vooral handig voor personeel in de werkplaatsen van NedTrain. Zo kan elk personeelslid voor elk moment van de dag zien aan welk project hij of zij werkt en wanneer zijn of haar pauzes zijn. Belangrijk is ook om er rekening mee te houden dat deze resources mensen zijn. Ze kunnen niet te lang achter elkaar werken en hebben na een bepaalde tijd pauze nodig. Het werkschema kan visueel gemaakt kunnen worden door in plaats van het resource profiel een schema zoals in Figuur 9 te laten zien in de interface. Elke rij hoort bij een werknemer/resource unit en elke kolom bij een tijdseenheid. Een groen ingekleurd vlak betekent dat de werknemer aan het werk is.



Figuur 9: Resource unit schema

9 Conclusie

Achteraf gezien kunnen we een aantal conclusies trekken over dit project. In het Plan van Aanpak (Bijlage B) wordt een aantal op te leveren features genoemd, onderverdeeld in drie categorieën van prioriteit. Alle taken met hoge prioriteit zijn volbracht en in de categorie met taken van gemiddelde prioriteit stond de undo-functionaliteit, die wij niet hebben kunnen implementeren door een gebrek aan tijd. We hebben onderzocht wat er nodig zou zijn om deze functionaliteit te ïmplementeren en kwamen tot de conlusie dat dit erg veel tijd zou gaan kosten en niet in verhouding zou staan met wat het zou opleveren. In de categorie lage prioriteit hebben we alles kunnen doen, behalve *"Het verduidelijken van het vergelijken van verschillende oplossingen."* We zijn erg tevreden nu we kunnen concluderen dat we zoveel features hebben kunnen maken.

We hebben allen veel geleerd van de programmeertaal C++ en het Qt framework. Deze waren beide voor ons erg onbekend en vooral C++ leverde soms aardig wat hobbels op in de weg naar succes.

De samenwerking liep gesmeerd, dit mede doordat wij ook al aan een eerder project hebben samengewerkt. De feedback van onze begeleiders heeft ook zeker bijgedragen aan een beter eindproduct. Het feit dat andere groepen al aan dit project hebben gewerkt en zeer waarschijnlijk ook wij opgevolgd zullen worden door enthousiaste informatici maakt dit een erg realistisch project. Wij vonden het een zeer leerzaam en geslaagd Bachelorproject en zijn tevreden met de resultaten.

Referenties

[1] E. Ammerlaan, J. Elfers, E. Walraven, and W. Wisse. Visualisatie en ondersteuning bij planning en scheduling. 2012.

[2] J. Blazewicz, J. Lenstra, and A. R. Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, 1983.

[3] R. Evers. Algorithms for scheduling of train maintenance. 2010.

[4] M. Lombardi and M. Milano. Constraint based scheduling to deal with uncertain durations and self-timed execution. In *Proceedings of the 16th International Conference on Principles and Practice of Constraint Programming*, CP'10, pages 383–397, Berlin, Heidelberg, 2010. Springer-Verlag.

[5] N. Policella, A. Cesta, A. Oddi, and S. F. Smith. From precedence constraint posting to partial order schedules: A CSP approach to robust scheduling. *AI Commun.*, 20(3):163–180, 2007.

[6] N. Policella, A. Oddi, S. F. Smith, and A. Cesta. Generating robust partial order schedules. In *Principles and Practice of Constraint Programming–CP 2004*, pages 496–511. Springer, 2004.

[7] K. Schwaber and J. Sutherland. *The Scrum Guide*. 2011.

[8] J. Visser. Sig/tüvit evaluation criteria trusted product maintainability: Guidance for producers version 6.1. *Software Improvement Group*, 2014.

[9] D. Wilmer. Investigation of ënhancing flexibility and robustness in multi-agent task scheduling". *CoRR*, abs/1307.0024, 2013.

[10] M. Wilson, T. Klos, C. Witteveen, and B. Huisman. Flexibility and decoupling in the simple temporal problem. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI'13, pages 2422–2428. AAAI Press, 2013.

Bijlage A Opdrachtomschrijving

Project description

NedTrain has developed a scheduling tool for operational maintenance. This tool consists of a scheduler and an extensive user interface. The proposed project consists in extending the current solver to construct flexible schedules for maintenance jobs. The proposed activities are.

1. The addition of constraint posting techniques that provide a set of temporal constraints ensuring the satisfaction of all resource constraints.
2. The use of an LP solver to measure the flexibility of the resulting constraint system.
3. The extension of the user interface to provide the user with flexibility information and to evaluate the changes in flexibility after the users feedback.

Company description

NedTrain is a part of NS Group, which is the largest railway operator in the Netherlands. NS Group can be decomposed in three important parts: railway station exploitation, train maintenance and of course passenger transportation. NedTrain, which performs rolling stock maintenance, is currently contracted to perform maintenance for the rolling stock used by nsr and NS HiSpeed. The fleet used by these two divisions has a size of around 3000 passenger carriages, of which some 250 are under some form of maintenance at any one time. Maintenance operations are spread out over 37 locations throughout the Netherlands. The core task of NedTrain is guaranteeing fleet availability in the broadest sense of the word. Firstly, this means that trains should have as few breakdowns during service as possible. To ensure this, NedTrain has to focus on the quality of their maintenance work.

Auxiliary information

This information has been provided by me (Cees Witteveen) after consulting Bob Huisman MSc, the director of research at NedTrain Company: bob.huisman@nedtrain.nl

Bijlage B Plan van Aanpak

TI3800 BACHELOR EINDPROJECT

Plan van Aanpak

Chris Bakker
Anton Bouter
Martijn den Hoedt

Mei 2014

Inhoudsopgave

1 Inleiding

Dit project zal gaan over het implementeren van een nieuw algoritme voor een bestaande grafische scheduling-tool en het uitbreiden en verbeteren hiervan voor het bedrijf *NedTrain*. Met deze tool kunnen gebruikers complexe schedulingsproblemen oplossen en op een intuïtieve manier stapsgewijs onderzoeken. Om het project zo gestructureerd mogelijk te laten verlopen wordt er van tevoren een Plan van Aanpak geschreven. Een Plan van Aanpak bevat alle gemaakte afspraken tussen de opdrachtgever en de projectleden en beschrijft op welke manier het project aangepakt gaat worden. Het bevat onder andere een precieze beschrijving van de opdracht, de gestelde eisen, de op te leveren producten, beperkingen en de voorwaarden. Daarnaast geven wij een tijdsplanning en beschrijven we hoe de kwaliteit van de gemaakte code en het project gewaarborgd wordt. Dit zorgt ervoor dat het gehele project en de uitvoering daarvan helder is voor alle betrokkenen. Wijzigingen en toevoegingen aan het Plan van Aanpak kunnen eventueel gedurende het project besproken worden, zolang de opdrachtgever alsmede de projectleden hiervoor goedkeuring hebben gegeven. Deze wijzigingen zullen opgenomen worden in de voortgangsrapportages.

1.1 Het Bedrijf

NedTrain BV is een dochterbedrijf van de *Nederlandse Spoorwegen (NS)* dat zorgt voor het onderhouden en repareren van de treinen die NS gebruikt [1]. Bovendien zijn ze ook verantwoordelijk voor het onderhoud van het materieel van Arriva, Syntus en Veolia. Het hoofdkantoor van NedTrain is gevestigd in Utrecht.

1.2 Aanleiding Project

Voor NedTrain is dit project onderdeel van onderzoek naar de mogelijkheden van complexe algoritmen om hun taken te automatiseren. In het verleden is er door NedTrain, in samenwerking met de TU Delft, een tool ontwikkeld waarin het plannen van taken gevisualiseerd kan worden. Deze tool, de *NedTrain planner*, wordt gebruikt door NedTrain om onderzoek te doen naar schedulingsalgoritmen en het verbeteren daarvan. Voor de TU Delft heeft deze tool ook een educatief doel, namelijk het op een visuele manier duidelijk maken wat er gebeurt bij het uitvoeren van een complex algoritme.

Naar aanleiding van de laatste vooruitgang op het gebied van schedulingsalgoritmen, bijvoorbeeld door Wilson et. al. op het gebied van flexibiliteit [4], wil NedTrain graag dat hun tool verbeterd wordt om deze vooruitgang te implementeren. Aan ons werd gevraagd of wij belangstelling hadden om hieraan te werken voor ons Bachelorproject. Dit leek ons interessant om te doen en dus zijn wij indirect in contact gekomen met NedTrain. Tijdens een eerste kennismakingsgesprek zijn de opdracht en belangen van NedTrain besproken. Naar aanleiding van dat gesprek is dit Plan van Aanpak opgesteld.

2 Projectopdracht

Om een beschrijving te geven van de taken die we gaan doen in dit project, geven we eerst informatie over onze opdrachtgever en wat zijn belang bij het project is. Daarna geven we de probleemstelling, gevolgd door de beschrijving van de opdracht en onze doelstelling. Aan de hand hiervan stellen we de op te leveren producten en diensten op, evenals de beperkingen en voorwaarden die gelden voor de opdracht.

2.1 Opdrachtgever

De opdrachtgever voor dit project bestaat uit NedTrain en de Technische Universiteit Delft. De contactpersoon binnen NedTrain is ir. Bob Huisman (b.huisman@nedtrain.nl). Hij heeft ook vorige projecten die zich bezig hielden met software voor NedTrain begeleid. Onze begeleider van de TU Delft is prof. dr. Cees Witteveen (c.witteveen@tudelft.nl). Hij is werkzaam binnen de Algoritmiek Groep op de faculteit EWI (Elektrotechniek, Wiskunde en Informatica).

2.2 Probleemstelling

De NedTrain planner, ontwikkeld in samenwerking met de TU Delft, is een grafische tool die moeilijke schedulingsproblemen kan visualiseren. Met behulp van zelfgemaakte oplossing-scripts kunnen deze problemen opgelost worden. De tool geeft vervolgens de mogelijkheid om deze oplossingen te bekijken en aanpassingen te maken om zo de veranderingen te onderzoeken. Deze planner wordt gebruikt door NedTrain om onderzoek te doen naar schedulingsalgoritmen en het verbeteren daarvan. Daarnaast heeft het voor de TU Delft ook een educatief doel, doordat deze op een visuele manier duidelijke maakt hoe schedulingsalgoritmen te werk gaan. De huidige versie is een resultaat van meerdere projecten van studenten aan de TU Delft. De applicatie dient als proof of concept om te laten zien wat er mogelijk is met complexe algoritmen op het gebied van het plannen van treinonderhoud. NedTrain zou graag willen dat de laatste ontwikkelingen op het gebied van schedulingsalgoritmen ook in de tool worden geïmplementeerd. Hierdoor kan NedTrain beter inzicht krijgen in wat voor ontwikkelingen de laatste jaren gemaakt zijn en of ze gebruikt kunnen worden voor andere toekomstige systemen.

2.3 Doelstelling Project

Voor het berekenen en inzicht krijgen in de flexibiliteit van schedulingsproblemen bestaat er nog niet zoveel functionaliteit voor de tool. Daarnaast zijn er in de laatste jaren op dit gebied ontwikkelingen gemaakt die goed gebruikt zouden kunnen door NedTrain voor toekomstige systemen. Aangezien er ook op de TU Delft gewerkt wordt aan dit onderwerp is er daar ook interesse voor het implementeren van deze missende functionaliteit, omdat dit tot nieuwe inzichten kan leiden en de tool gebruikt kan worden als educatief hulpmiddel. Op dit moment kunnen er bij het verschuiven van de starttijden van taken resource conflicten ontstaan wat leidt tot een niet mogelijke rooster. Om dit op te lossen zou de NedTrain planner uitgebreid kunnen worden met een algoritme dat de chaining methode implementeert zodat de taken verschoven kunnen worden zonder dat er een resource conflict ontstaat [2]. Er worden hier namelijk zogenaamde chains, oftewel kettingen, van taken gemaakt waardoor er geforceerd wordt dat taken in dezelfde volgorde uitgevoerd blijven worden en dus niet de capaciteit van de resources overschrijden. Het verschuiven van taken kan echter wel het gevolg hebben dat ook ander taken verschoven worden. Om dit tegen te gaan, wil NedTrain ook dat er door middel van een *Linear Programming (LP)* solver een schema wordt gevonden, waarbij elke taak een interval heeft waarover deze verschoven kan worden, zonder dat dit invloed heeft op de andere taken. De mate waarin dit kan

gebeuren, wordt gezien als de flexibiliteit van een schema. Visueel zullen er daardoor dus ook aanpassingen gemaakt moeten worden, omdat de huidige tool nog geen manier heeft om de maat van de flexibiliteit van een schema te tonen. Hierbij moet er wel rekening worden gehouden dat oude solvers die niet de flexibiliteit van schemas berekenen nog steeds werken op de tool. Ook moet de NedTrain planner voor educatieve doeleinden gebruikt kunnen worden en dus inzicht kunnen bieden in hoe deze solvers aan het schema gekomen is. Tenslotte moet er gekeken worden of de tool, die op dit moment alleen onder Linux werkt, geport kan worden naar Windows 7 om de bruikbaarheid te vergroten.

2.4 Opdrachtformulering

De volgende hoofdzaken zullen behoren tot de verantwoordelijkheden van de projectgroep:

1. Het implementeren van een solver die gebruik maakt van de chaining methode.
2. Het uitbreiden van de solver met de functionaliteit om aan elke taak een onafhankelijk interval toewijst waarover het zonder andere gevolgen verschoven kan worden, door middel van een LP solver.
3. Het visualiseren van de manier waarop de solver tot zijn antwoord komt.
4. Onderzoek doen naar de hoeveelheid tijd die nodig is om de NedTrain planner ook beschikbaar te maken voor Windows 7 en de nieuwst Qt versie. Als verwacht wordt dat het porten in redelijke tijd uitgevoerd kan worden, zal dit ook gedaan worden.
5. De gebruikerservaring verbeteren door een aantal opties en shortcuts toe te voegen en door een aantal functies te veranderen.

2.5 Op te leveren Producten en Diensten

De volgende features worden verwacht om geïmplementeerd te worden. Deze zijn onderverdeeld in verschillende maten van prioriteit en horen bij één van de hoofdzaken genoemd in 2.4. De nummers van de onderstaande producten en diensten komen overeen met een nummer van één van de hoofdzaken in 2.4

Hoge Prioriteit

1. Het maken van een solver die er voor zorgt dat er bij het verslepen van een taak geen resource conflicten ontstaan, door middel van de chaining methode.
2. Het toevoegen van functionaliteit aan de solver, zodat deze aan elke taak een onafhankelijk interval toewijst waarover deze taak verschoven kan worden, door middel van een LP solver.
3. Het visualiseren van chains in de interface van taken, zodat duidelijk is tot welke chain elke taak behoort en hoe het algoritme hiertoe gekomen is.
3. Het visualiseren van de flexibiliteit van elke taak en van het schema in het geheel door het tonen van intervallen en het geven van de maat van de flexibiliteit.
4. De tool voor zowel Linux als Windows 7 beschikbaar maken. Op dit moment is deze alleen voor Linux systemen beschikbaar.

Gemiddelde Prioriteit

3. Nieuwe view voor resources, waarbij weergegeven wordt tot welke chains de gebruikte resources behoren.
4. Het porten van de applicatie van Qt versie 4.8 naar versie 5.2.
5. Undo en redo functies om het verslepen en veranderen van taken ongedaan te maken. De laatste tien acties worden opgeslagen, zodat die ongedaan gemaakt kunnen worden. Deze acties zijn niet meer ongedaan te maken na het afsluiten van de tool.

Lage Prioriteit

3. Het verduidelijken van het vergelijken van verschillende oplossingen.
5. Horizontaal scrollen moet mogelijk worden met shift + scroll. Deze functie heeft verder hetzelfde gedrag als wanneer er met de muis een scrollbalk verschoven wordt. De muis moet in het gebied staan dat horizontaal gescrolld moet worden.
5. Zoomfunctie moet ook mogelijk zijn met ctrl + scroll. Dit werkt verder hetzelfde als de knoppen 'inzoomen' en 'uitzoomen'.
5. Sluitknopje op de tab van elke instantie die geopend is in de tool.

2.6 Eisen en Beperkingen

Er wordt geëisd dat alle in 2.5 genoemde features met hoge prioriteit, met uitzondering van het porten naar Windows 7, worden uitgevoerd. Het porten naar Windows 7 zal onderzocht worden en worden uitgevoerd als verwacht wordt dat dit in redelijke tijd kan gebeuren. Eventuele veranderingen aan de eisen zullen met de opdrachtgever gecommuniceerd worden. Daarnaast is kwaliteit van code en documentatie ook een belangrijke eis van de opdrachtgever, zodat het uiteindelijke product makkelijk gebruikt, uitgebreid en overgedragen kan worden.

2.7 Voorwaarden

Van de projectleden wordt verwacht om duidelijk en op tijd te communiceren met de opdrachtgever, bijvoorbeeld in het geval van tegenvallers bij de implementatie of bij het maken van een afspraak voor feedback. Bij eventuele wijzigingen aan eisen en dergelijke moet dit zo spoedig mogelijk met de projectleden besproken worden. Van de opdrachtgever wordt verwacht dat deze duidelijk de eisen van het product communiceert en feedback geeft op de geleverde documentatie.

3 Aanpak en Tijdsplanning

De aanpak van het hele project zal in dit hoofdstuk beschreven worden samen met een tijdsplanning. Op deze manier wordt er duidelijkheid gecreëerd over hoe wij de eerder gestelde eisen aan zullen pakken en in welke fasering van het project.

3.1 Aanpak

Het hele project zal opgedeeld worden in drie fasen, namelijk een oriëntatie- en ontwerpfase, een implementatie- en testfase en uiteindelijk een afrondingsfase. Voor elk van deze fasen zullen we bespreken wat er gedaan gaat worden, welke methode we gebruiken en waarom we hiervoor gekozen hebben.

3.1.1 Oriëntatie en Ontwerp

In de eerste twee weken houden wij ons bezig met de oriëntatie en het ontwerp van de opdracht. Tijdens de oriëntatie moeten we bepalen wat de exacte opdracht is, wie er betrokken zijn, welke middelen we tot onze beschikking hebben en onder welke condities de opdracht gedaan moet worden. Dit soort informatie kunnen wij verkrijgen door een gesprek te voeren met de opdrachtgever en de aangeleverde documenten en informatie te lezen. Dit alles zal resulteren in een *Oriëntatieverslag*.

Als het fundamentele gedeelte van de oriëntatiefase bekend is, kan de focus gelegd worden op het ontwerp en de omvang van het te maken systeem. Dit kan gedaan worden door een *requirement analysis* toe te passen. Hierbij wordt onderzocht aan welke eisen het systeem moet voldoen en welke functionaliteiten aanwezig moeten zijn. Bovendien kan onderscheid gemaakt worden tussen functionele en non-functionele requirements. De functionele requirements kunnen gevonden worden door scenario's te schrijven. De non-functionele requirements moeten voornamelijk gecommuniceerd worden door de opdrachtgever. Dit zal beschreven worden in het document *Requirementsanalyse*.

3.1.2 Implementatie en Testen

Aan de hand van de eisen en ontwerpen die beslist zijn in de oriëntatie- en ontwerpfase wordt het systeem geïmplementeerd. Deze fase duurt zes weken en hierin wordt de techniek Scrum gebruikt, beschreven in *The Scrum Guide* [3]. Scrum is een framework voor het ontwikkelen van software dat ervoor zorgt dat de opdrachtgever en andere belanghebbenden betrokken zijn bij de implementatie, de gemaakte voortgang en de tot dan toe gemaakte software. Hierdoor kan er ook betere terugkoppeling gegeven worden aan de projectleden, wat zorgt voor een betere samenwerking en een betere vervulling van de wensen van de opdrachtgever. Het ontwikkelingsproces bestaat uit meerdere sprints, die in dit geval elk een week duren. Van tevoren is er besloten op welke taak gefocust zal worden in welke week. Aan het begin van elke sprint wordt er besproken hoe het met de planning staat, wat er gedaan is en wat er in die sprint geïmplementeerd gaat worden. Deze besluiten worden in elke sprint nog dagelijks bekeken door het houden van scrummeetings van ongeveer een kwartier. Aan het einde van de sprint zal de gemaakte functionaliteit voorgelegd worden aan de opdrachtgever, zodat de feedback gebruikt kan worden voor de volgende sprint.

In deze fase zal de gemaakte code ook een keer opgestuurd moeten worden naar de *Software Improvement Group (SIG)*. SIG beoordeelt de kwaliteit van de code handmatig en met behulp van geautomatiseerde tools. Het opsturen van de code naar SIG moet gebeuren op 13 juni 2014. Ongeveer een week later ontvangen wij hun feedback.

3.1.3 Afronding

De laatste fase zal bestaan uit het afronden van het project. De feedback die gekregen is van SIG zal dan toegepast moeten worden op de code. Daarnaast moet de code zo afgeleverd worden dat het herbruikbaar is voor toekomstige projecten. De definitieve code moet opnieuw, uiterlijk 5 werkdagen voor de datum van de eindpresentatie, ingeleverd worden. Het afronden van het eindverslag en het voorbereiden van de eindpresentatie zal ook in deze fase gedaan worden.

3.2 Tijdsplanning

Een tijdsplanning voor de beschreven aanpak kan gevonden worden in Bijlage A.

4 Projectinrichting

Het doel van de projectinrichting is om inzicht te geven in de manier waarop het project georganiseerd is en hoe er voor gezorgd wordt dat het project op een gestructureerde manier verloopt.

4.1 Organisatie

Hoewel alle projectleden aan elk onderdeel van het project bijdragen, definiëren we zes onderdelen met voor elk onderdeel een projectlid dat daarvoor eindverantwoordelijk is. Dit projectlid is verantwoordelijk voor de kwaliteit van dit onderdeel en controleert of aan alle voorwaarden is voldaan. Ook herinnert hij de andere projectleden aan deadlines van dit onderdeel en spoort hij ze aan om hieraan te werken als dit nodig is. De verdeling van verantwoordelijkheden is weergegeven in Tabel 1.

Projectlid	**Verantwoordelijkheid**
Anton	Het controleren van de voortgang en de kwaliteit van de LP solver en de communicatie tussen LP solver en GUI.
	Het controleren of de verslagen aan de gestelde eisen voldoen en kwaliteitsbewaking over bijvoorbeeld spelling en stijl van de verslagen.
Chris	Het porten van de applicatie naar Windows 7 en het onderzoeken van de mogelijkheden op dit gebied.
	Ervoor zorgen dat er voldoende en kwalitatief goede tests geschreven worden, maar ook dat falende tests zo snel mogelijk opgelost worden, zodat bugs niet opstapelen.
	Het zo veel mogelijk naleven van de tijdsplanning, zoals het ervoor zorgen dat er niet te lang op één probleem wordt gefocust terwijl andere taken verwaarloosd worden.
Martijn	Het controleren van de voortgang en de kwaliteit van de solver die gebruikt maakt van chaining.
	Het controleren van de voortgang en de kwaliteit van de GUI en de veranderingen die hieraan geïmplementeerd zullen worden.

Tabel 1: De verdeling van verantwoordelijkheden onder de projectleden.

4.2 Tijdsinvestering en vaardigheden

De projectleden zijn in dienst als stagiair bij NedTrain en worden volgens de arbeidsovereenkomst verwacht om gemiddeld minimaal 36 uur per week aan het werk te zijn. Het Bachelor Eindproject is een studieonderdeel van 15 ECTS en heeft dus bij een studielast van 28 uur per ECTS een totale studielast van 420 uur. Over een periode van 10 weken is de verwachte studielast dus gemiddeld 42 uur per week.

Als toelatingseis voor het Bachelor Eindproject moeten de projectleden alle vakken uit het eerste en het tweede jaar van de bachelor Technische Informatica gehaald hebben. De vaardigheden die van de projectleden verwacht worden, bestaan dus uit alle vaardigheden die worden verwacht

bij deze vakken. Deze zijn te vinden in de studiegids van de TU Delft [1].

4.3 Administratieve Procedures

Voor het bijhouden van bugs en taken zal gebruikt gemaakt worden van de ingebouwde issue tracker op Bitbucket [2]. Aangezien de repository ook gehost is op Bitbucket, is het makkelijk om administratie hier ook bij te houden. Met deze tracker kan van een issue geselecteerd worden of het een bug, enhancement, proposal of een task is. Daarnaast kan een prioriteit ingesteld worden en kan de taak toegewezen worden aan een projectlid.

Voor een aantal uitgebreidere trackers, zoals Atlassian Jira [3] of Mantis Bug Tracker [4] moet betaald worden. Wij verwachten dat de Bitbucket tracker ook voldoende mogelijkheden biedt.

4.4 Rapportering

Het team zal communiceren met de opdrachtgever op verschillende manieren. Eén daarvan is het in een gesprek doornemen wat voor code er is gemaakt en wat er in de verslagen opgeschreven is. Er worden vier verslagen gemaakt: het Plan van Aanpak, het Oriëntatieverslag, de Requirementsanalyse en het Eindverslag. Bij de gesprekken maken we notulen, zodat we de afspraken en ideeën die ter sprake komen niet vergeten.

4.5 Financiering en Resources

Voor dit project wordt er zoveel mogelijk gebruik gemaakt van open source software. We verwachten dat we alle doelen kunnen behalen zonder aparte licenties nodig te hebben.

Het team zorgt zelf voor computers waar zowel Windows 7 als een versie van Ubuntu op is geïnstalleerd. De tool die wij gaan verbeteren is gemaakt in C++ met behulp van het framework Qt[5], dat het mogelijk maakt om eenvoudig cross-platform GUI's te ontwikkelen voor C++ software. Door een aantal grote wijzigingen in Qt werkt de tool nog wel in Qt versie 4.8, maar niet met Qt versie 5.x. We maken ook gebruik van een private repository op Bitbucket als versiebeheersysteem in combinatie met de ingebouwde issue tracker. Tenslotte wordt er gebruik gemaakt van de continuous integration service Jenkins, dat wordt gehost op een eigen server op eigen apparatuur om zo kosten voor het huren van een server te besparen.

Op de TU Delft zijn er genoeg werkplekken, maar bij NedTrain is er niet veel ruimte om te werken aan het project. Het maakt voor het team weinig uit waar er gewerkt wordt, thuis werken zou ook een mogelijkheid kunnen zijn. We kiezen er toch voor om zoveel mogelijk op dezelfde locatie te werken, omdat we verwachten dat dit de samenwerking en het product ten goede zal komen.

[1] studiegids.tudelft.nl
[2] bitbucket.org
[3] atlassian.com/software/jira
[4] mantisbt.org
[5] qt-project.org

5 Kwaliteitsbewaking

Het bewaken van de kwaliteit is in elk project een belangrijk onderdeel waarover nagedacht moet worden. Voor ons project is kwaliteit belangrijk omdat de tool na dit project verder gebruikt zal worden binnen NedTrain. Het moet onderhoudbaar zijn en blijven voor toekomstige ontwikkelaars en nieuwe projecten. In dit hoofdstuk zal uitgewerkt worden welke middelen en welke technieken gebruikt zullen worden om deze doelen te bereiken.

5.1 Documentatie

Een belangrijke eis van de opdrachtgever is dat de documentatie kwalitatief goed is en alle aspecten behandeld, zodat het voor anderen die niet betrokken geweest zijn tijdens het project makkelijker is om inzicht te krijgen in de keuzes en implementaties die toegepast zijn tijdens het project. Daarnaast geeft het ook een goed overzicht voor de opdrachtgever welke functionaliteiten toegevoegd zijn en hoe ze werken. Voor de projectleden heeft het ook als doel om de kwaliteit te waarborgen, aangezien het ook voor de projectleden een overzicht geeft en de samenhang beschrijft. Het is daarbij belangrijk voor de projectleden om gedurende het project de documentatie bij te houden en de dingen op te schrijven die uiteindelijk in het eindverslag moeten komen te staan. Voorbeelden hiervan zijn de gesprekken tussen opdrachtgever en projectleden en de gemaakte Scrum sprints.

5.2 Versiebeheer

Er zal gebruik gemaakt worden van een private repository op Bitbucket dat werkt met Git. Subversion behoorde ook tot de mogelijkheden, maar omdat het team beter bekend is met Git en dit ook populairder is onder andere ontwikkelaars hebben we voor een Git repository gekozen. Subversion wordt gratis beschikbaar gesteld voor studenten door de faculteit EWI. Daarentegen kan Git gratis gebruikt worden door iedereen en bestaat ook zo de mogelijkheid om later het project publiekelijk toegankelijk te maken. Er is voldoende documentatie te vinden voor Git als we ergens niet uitkomen. Er is voor de website Bitbucket gekozen en niet voor het bekendere Github.com, omdat op Bitbucket gemakkelijk en gratis een repository aangemaakt kan worden die alleen door het team bekeken kan worden.

5.3 Evaluatie

Het evalueren van de gemaakte code en de tool zal in dit project op drie manier gebeuren. De code zal getest worden met bestaande en nieuwe tests door middel van een continuous integration service. Daarnaast zal de code opgestuurd worden naar de Software Improvement Group (SIG) [1] die feedback zullen geven op de kwaliteit en structuur van de software. Het project zal eindigen met een pilottest.

5.3.1 Code Testen

Om de werking van de code te garanderen zal er tijdens het project tests geschreven worden. Alle tests zullen automatisch uitgevoerd worden door een continuous integration service. Er zal gebruik gemaakt worden van Jenkins [2] die de code en tests op een server uitvoeren en hierover

[1] www.sig.eu/nl
[2] jenkins-ci.org

resultaten en statistieken geeft. Aangezien Jenkins dit meerdere malen op een dag doet krijgen wij gelijk een melding als een testcase niet slaagt.

Zoals eerder genoemd in paragraaf 4.5, wordt er gebruik gemaakt van de issue tracker van Bitbucket. Hierop zullen gevonden bugs geplaatst worden en zal bijgehouden worden welke features al geïmplementeerd zijn en welke niet. Dit geeft een goed overzicht aan de projectleden wat er in de code nog gedaan moet worden.

5.3.2 SIG Evaluatie

Zoals beschreven in paragraaf 3.1.2, beoordeelt SIG de kwaliteit van de code handmatig en met behulp van geautomatiseerde tools. De kwaliteit wordt beoordeeld door het geven van sterren op een schaal van 1 t/m 5. Daarnaast wordt informatie gegeven over hoe deze uitkomst tot stand is gekomen en hoe de code verbeterd kan worden. Het opsturen van de code naar SIG moet gebeuren op 13 juni 2014. Ongeveer een week later ontvangen wij hun feedback.

5.3.3 Pilots

De tool zal getest worden door het uitvoeren van gebruikerstest in de vorm van een pilot. Dit zal tijdens en aan het einde van het project gedaan worden. Een prototype van de tool zal in week 5 gepresenteerd worden aan de opdrachtgever. Hierdoor kan er getest worden of de verwachtingen van de opdrachtgever kloppen met wat er al geïmplementeerd is en of er nog enige veranderingen moeten komen. Aan het einde van het project kan de tool getest worden door een paar studenten om te kijken of de studenten door middel van de tool het nieuwe geïmplementeerde algoritme kunnen begrijpen. Dit kan als maatstaf gebruikt of het project geslaagd is.

Referenties

[1] Nedtrain. `http://www.nedtrain.nl/`. Geraadpleegd: 28 Mei 2014.

[2] C. Bakker, A. Bouter, M. den Hoedt, and W. Meerkerk. *Resource Constrained Project Scheduling*. 2014.

[3] K. Schwaber and J. Sutherland. *The Scrum Guide*. 2011.

[4] M. Wilson, T. Klos, C. Witteveen, and B. Huisman. Flexibility and decoupling in the simple temporal problem. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI'13, pages 2422–2428. AAAI Press, 2013.

Bijlage A Tijdsplanning

Week	Datum	Activiteiten
1	28-04 t/m 04-05	Plan van Aanpak Oriëntatie verslag
2	05-05 t/m 11-05	Inleveren Plan van Aanpak Requirements Analyse verslag
3	12-05 t/m 18-05	*Eerste Scrum Sprint* **Chaining Implementeren**
4	19-05 t/m 25-05	*Tweede Scrum Sprint* **LP Solver**
5	26-05 t/m 01-06	*Derde Scrum Sprint* **Communicatie en Uitloop** 28-5 Prototype Solver 29-5 en 30-5 Vrij (Hemelvaart)
6	02-06 t/m 08-06	*Vierde Scrum Sprint* **Visualisatie van algoritme**
7	09-06 t/m 15-06	*Vijfde Scrum Sprint* **Uitloop of Gemiddelde Prioriteit taken** Draft Eindverslag inleveren 09-06 Vrij (Pinksteren) 13-06: SIG Evaluatie
8	16-06 t/m 22-06	*Laatste Scrum Sprint* **Uitloop of Lage Prioriteit taken** Draft Eindverslag bespreken SIG Feedback
9	23-06 t/m 29-06	Laatste SIG Evaluatie Eindverslag inleveren Voorbereiden Presentatie
10	30-06 t/m 06-07	02-07: Eindpresentatie

Tabel 2: De tijdsplanning van het project.

Bijlage C Oriëntatieverslag

TI3800 BACHELOR EINDPROJECT

Oriëntatieverslag

Chris Bakker
Anton Bouter
Martijn den Hoedt

Mei 2014

Inhoudsopgave

1 Inleiding

Dit project betreft het implementeren van het chaining algoritme in combinatie met een LP solver voor een bestaande grafische scheduling-tool van het bedrijf NedTrain. Deze oplossing moet op een intuïtieve manier gevisualiseerd worden door het uitbreiden en verbeteren van de bestaande software. Hierdoor kunnen gebruikers de flexibiliteit van complexe schedulingsproblemen oplossen en onderzoeken. In het oriëntatieverslag wordt het vooronderzoek in de oriëntatiefase van het project beschreven. Hierbij wordt er niet gekeken naar wat de opdrachtomschrijving is en welke eisen er gesteld worden door de opdrachtgever, aangezien dit beschreven is in het Plan van Aanpak. In dit verslag wordt er gekeken naar de achtergrond van het bedrijf en de software en methoden die voorafgaand aan het project al gebruikt werden. Er zullen beschrijvingen hiervan gegeven worden in combinatie met referenties en een uitleg waarom er voor bepaalde technieken en methoden gekozen is.

2 NedTrain

De Nederlandse Spoorwegen heeft verschillende dochterbedrijven met elk hun eigen taak. Het bekendste dochterbedrijf is NS Reizigers B.V., dat verantwoordelijk is voor het personenvervoer. Onze opdrachtgever is NedTrain B.V., dat in samenwerking met de TU Delft de 'NedTrain Planner' ontwikkelt. NedTrain is het dochterbedrijf dat zorgt voor het onderhoud aan alle treinen aangeleverd door NS Reizigers. Dat onderhoud bestaat niet alleen uit het repareren van treinen, maar ook uit het reinigen en moderniseren van treinen. Er moet hierbij gezorgd worden dat van de grofweg 3000 bestaande treinstellen bij NS er op elk moment genoeg beschikbaar zijn om gebruikt te kunnen worden door NS Reizigers. Om dit te bereiken staan er op elk moment van de dag ongeveer 250 treinstellen tegelijk voor onderhoud op meer dan 30 locaties door heel Nederland bij NedTrain. Hier werken in totaal ongeveer 3500 mensen 24 uur per dag, 7 dagen per week aan de opgenoemde taken. Het hoofdkantoor van NedTrain staat in Utrecht.

3 Bestaande Software

Onze opdracht is het verbeteren en uitbreiden van de 'NedTrain planner', die het resultaat is van meerdere projecten door studenten van de TU Delft. In dit hoofdstuk wordt uiteengezet welke software er al bestaat en wat voor software gebruikt gaat worden gedurende het project. Eerst bespreken we de NedTrain planner zelf, daarna het framework Qt dat gebruikt is voor het implementeren van de tool en tenslotte de Google Test library en de solvers.

3.1 NedTrain Planner

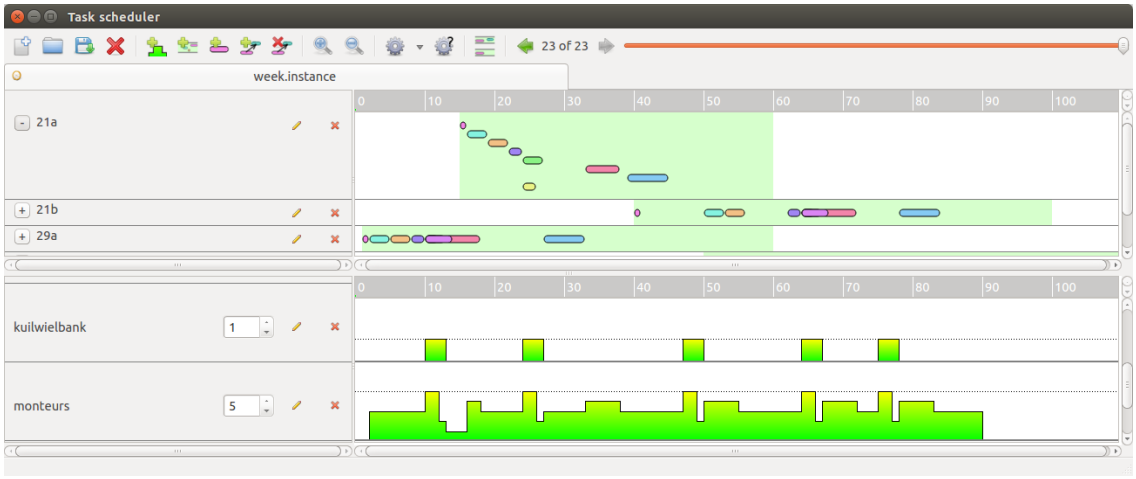
De NedTrain Planner bestaat grofweg uit drie componenten, namelijk de gebruikers-interface, een solver en de communicatie tussen die twee. De gebruikers-interface maakt het mogelijk om een probleeminstantie, dat beschreven zal worden in 4.1, te maken en oplossingen hiervan door een solver te visualiseren.

In vorige bachelorprojecten en mastertheses is deze tool steeds verder uitgebreid. De 'OnTrack Scheduler' is een tool met een eenvoudige GUI en een solver die ontwikkeld is door Ronald Evers. Zijn onderzoek in 2010 voor de masterthesis had als doel om een goede solver te bouwen [5].

Daarna is in 2011 door Edwin Bruurs en Cis van der Louw een verbeterde gebruikers interface

gemaakt [4]. Deze verbeteringen waren gebruiksvriendelijker, maar de kwaliteit van de code was volgens het bachelorproject uit 2012 niet goed [1]. Deze groep, bestaande uit Erik Ammerlaan, Jan Elffers, Erwin Walraven en Wilco Wisse, heeft in 2012 opnieuw een verbeterde gebruikersinterface gemaakt voor de 'OnTrack Scheduler'. Deze hebben zij de NedTrain planner genoemd.

In deze nieuwe versie is onder andere de gebruikersinterface uitgebreid, kan het oplosproces stap voor stap weergegeven worden en kunnen probleeminstanties gemakkelijk geladen en opgeslagen worden. De gebruikersinterface van de NedTrain planner gemaakt in het Bachelor Eindproject van 2012 is te zien in Figuur 3.1.



Figuur 1: Gebruikersinterface NedTrain planner (Bachelor Eindproject 2012)

3.2 Qt Framework

Qt is een gebruikers-interface framework voor C++. De NedTrain planner maakt op dit moment gebruik van versie 4.8, maar de nieuwste versie is nu al versie 5.2. Door een paar belangrijke veranderingen geïntroduceerd in versie 5.0, werkt de NedTrain planner niet meer op versie 5.2. Het updaten van de NedTrain planner naar deze nieuwe versie zou mogelijk moeten zijn. Qt is open source software die voor meerdere besturingssystemen geschikt is. Het moet dus ook mogelijk zijn om de tool beschikbaar te maken voor onder andere Windows, Linux en Mac OS. Er is een IDE genaamd Qt Creator, die het mogelijk maakt om eenvoudig GUI's te ontwikkelen en C++ code te compileren.

3.3 Google Test en QTest

Voor de bestaande software, gemaakt in C++, is er ook C++ testcode geschreven, waarbij gebruik is gemaakt van de Google Test library. [1] Dit maakt het mogelijk om unit testing te doen. Ook ondersteunt de Google Test library fixture, mocks, exceptions, macro's en templates. Er wordt in deze tests gebruik gemaakt van de QTest namespace, waar alle functionaliteit van de QTestLib in zit. De andere test-libraries die we hebben bekeken zijn: QTestLib, CppTest en CppUnit. Deze drie en de eerder genoemde Google Test library worden allemaal, met behulp van een plugin, ondersteund door Jenkins. Dat is belangrijk voor ons, omdat er dan op een relatief eenvoudige manier continuous integration toegepast kan worden.

[1] code.google.com/p/googletest/

QTestLib is bedoeld voor het testen van Qt applicaties en bevat zowel ondersteuning voor unit testing als het testen van GUI's. Dit zou ook een goed alternatief zijn, omdat het echt bedoeld is voor het Qt framework. CppTest en CppUnit verschillen op een aantal punten. Het schijnt dat CppTest in het gebruik wat makkelijker is, maar met CppUnit is er meer mogelijk. Google Test library staat mocks toe, iets wat met CppTest en CppUnit niet mogelijk is. QTestLib staat het ook toe om het toetsenbord en de muis te simuleren. Aangezien de bestaande tests gebruik maken van de Google Test library en de QTest namespace, zijn deze allebei nodig om de bestaande tests te runnen. Daarom zal in ons project op dezelfde manier gebruik gemaakt worden van de Google Test library en de QTest functionaliteit. Dit betekent dat primair de Google Test library gebruikt zal worden en daarnaast de QTest namespace voor de GUI-tests.

3.4 Solver

Een programma dat een instantie van een schedulingsprobleem oplost wordt een solver genoemd. De solver die wij tot onze beschikking hebben, genaamd TMS, is ontwikkeld door Ronald Evers [5] en later verder ontwikkeld door Michel Wilson en de in paragraaf 3.1 genoemde groep voor hun Bachelor Eindproject in 2012. Een solver krijgt als input een instantie van een schedulingsprobleem bestaande uit een verzameling taken, resources en voorrangsrelaties in combinatie met enkele restricties, zoals release- en deadlinetijden en de maximale capaciteit van resources. De parser *Bison*[1] zorgt ervoor dat de data van een dergelijke instantie omgezet wordt in een datastructuur die een solver kan begrijpen. Deze solver maakt gebruik van *Precedence Constraint Posting* [2], waarbij er op basis van een heuristiek voorrangsrelaties aan de taken worden toegevoegd om resource conflicten te verhelpen.

Tijdens het oplossen van een instantie door een solver is er continu communicatie tussen de solver en de visualisatie tool. Zo wordt er onder andere doorgegeven hoe ver de solver is met het oplossen en of er een oplossing bestaat. Als de solver een oplossing gevonden heeft geeft de solver een signaal en wordt de output ingelezen door de visualisatie tool.

Volgens de output die gegeven is door een solver wordt de oplossing van een probleem gevisualiseerd. Zo zorgt de solver die wij tot onze beschikking hebben er voor dat er geen enkel resource conflict meer is. De methode die wordt gebruikt, garandeert echter alleen dat de *Earliest Starting Time* oplossing consistent is. Dit betekent dat elke taak op de vroegst mogelijke starttijd begint. In de tool kan er ook met taken geschoven worden, maar dit kan er toe leiden dat er een resource conflict ontstaat.

[1] `http://www.gnu.org/software/bison/`

4 Oplossingsmethode

In dit hoofdstuk zal eerst het achterliggende schedulingsprobleem van de NedTrain planner gedefiniëerd worden. Vervolgens komt aan bod wat voor methoden er gebruikt gaan worden om de benodigde features te implementeren, dan wat voor werkwijze er gebruikt zal worden en tenslotte wat voor hulpmiddelen hierbij nodig zijn.

4.1 Aanpak

Het probleem dat de NedTrain planner probeert op te lossen, bestaat uit een verzameling taken, een verzameling voorrangsrelaties, een verzameling van verschillende soorten resources en voor elke resource op elk tijdstip een maximale capaciteit. Om dit probleem op te lossen, moet er voor elke taak een starttijd gevonden worden, zodat er aan alle voorrangsrelaties en alle resource-capaciteiten wordt voldaan. Dit probleem wordt ook wel gedefiniëerd als het *Resource-Constrained Scheduling Problem (RCPSP)* [2]. Dit probleem is NP-hard, dus er is geen bekend polynomiaal algoritme om dit probleem om te lossen [3]. Omdat de probleeminstanties van NedTrain te groot zijn om exact op te lossen, moet er dus een niet-exacte methode gebruikt worden. De methode die door de huidige solver gebruikt wordt, is al beschreven in paragraaf 3.4, maar deze methode zal uitgebreid worden met de methoden die beschreven zullen worden in paragraaf 4.2. Meer over RCPSP en de mogelijke oplossingsmethoden ervan is te vinden in het bachelorseminariumverslag dat geschreven is door de huidige projectleden en Willem-Jan Meerkerk [2].

4.2 Oplossing

De grootste hindernissen van het project zijn het implementeren van een nieuwe solver die gebruik maakt van het chaining-algoritme en flexibiliteitsintervallen berekent door middel van een linear programming solver. Het onderzoek dat naar deze problemen gedaan is en de gekozen oplossing zal in deze paragraaf besproken worden.

Chaining

Zoals eerder genoemd in paragraaf 3.4, kan een schema inconsistent worden met de resources als er in de interface taken verschoven worden. Om dit probleem op te lossen, kan de *chaining* methode gebruikt worden [6]. Voor deze methode moet eerst een oplossing voor de instantie gevonden worden die resource consistent is. Daarna wordt er aan elke eenheid van een resource een zogenaamde chain van taken toegekend. Zo'n chain is een lijst van taken $a_1, \ldots a_n$, waarbij voor elke taak a_i geldt dat deze taak afgerond moet zijn, voordat taak a_{i+1} mag beginnen. Deze voorrangsrelatie wordt ook wel genoteerd als $a_i \prec a_{i+1}$. Pseudocode voor het chaining-algoritme is weergegeven in Algorithm 1.

Omdat de chaining methode als input een consistent schema nodig heeft, kan de in paragraaf 3.4 genoemde solver gebruikt worden. Deze solver levert een earliest starting time oplossing, die vervolgens dus door het chaining algoritme gebruikt kan worden.

Linear Programming

Voor het bepalen van de flexibiliteit van een oplossing, zullen we gebruik maken van de methode beschreven in *Flexibility and Decoupling in the Simple Temporal Problem* [8]. Dit paper beschrijft een metriek voor de flexibiliteit op basis van intervalsets. Om deze flexibiliteit te maximaliseren,

Algorithm 1 Chaining [6]

Input: A problem P and one of its fixed-times schedules S
Output: A partial order solution POS^{ch}

function CHAINING(P, S)
  $POS^{ch} \leftarrow P$
  Sort all the activities according to their start times in S
  Initialize all the chains empty
  for each resource r_j **do**
    for each activity a_i **do**
      for 1 **to** req_{ij} **do**
        $k \leftarrow SelectChain(a_i, r_j)$
        $a_k \leftarrow last(k)$
        $AddConstraint(POS^{ch}, a_k \prec a_i)$
        $last(k) \leftarrow a_i$
  return POS^{ch}

moet het gegeven tijdschema getransformeerd worden, waarna het door middel van *Linear Programming* efficiënt opgelost kan worden. Dit geeft als resultaat voor elke taak een onafhankelijk interval waarover deze taak verschoven kan worden, zonder dat dit invloed heeft op de andere taken.

Voor het oplossen van Linear Programming problemen bestaan er al verschillende solvers. Eén daarvan is het Gurobi commercieel softwarepakket, dat de problemen zeer snel kan oplossen. COIN-OR LP[1], afgekort CLP, is een open source Linear Programming solver in C++. We kiezen voor CLP, omdat deze software open source is, wat voor onze opdrachtgever aantrekkelijker is dan Gurobi, omdat daar wel een licentie voor gekocht moet worden.

4.3 Werkwijze

Tijdens de ontwikkelfase gaat het team werken met Scrum in wekelijkse sprints. De techniek Scrum wordt beschreven in *The Scrum Guide* [7]. Scrum is een framework voor het ontwikkelen van software dat ervoor zorgt dat de opdrachtgever en andere belanghebbenden betrokken zijn bij de implementatie, de gemaakte voortgang en de tot dan toe gemaakte software. Hierdoor kan er ook betere terugkoppeling gegeven worden aan de projectleden, wat zorgt voor een betere samenwerking en een betere vervulling van de wensen van de opdrachtgever. Het ontwikkelingsproces bestaat uit meerdere sprints, die in dit geval elk een week duren. Van tevoren is er besloten op welke taak gefocust zal worden in welke week. Aan het begin van elke sprint wordt er besproken hoe het met de planning staat, wat er gedaan is en wat er in die sprint geïmplementeerd gaat worden. Deze besluiten worden in elke sprint nog dagelijks bekeken door het houden van Scrum-meetings van ongeveer een kwartier. Aan het einde van de sprint zal de gemaakte functionaliteit voorgelegd worden aan de opdrachtgever, zodat de feedback gebruikt kan worden voor de volgende sprint.

De taken die nog gedaan moeten worden, worden in de productbacklog bijgehouden. Deze productbacklog is de lijst met taken die in het project nog gedaan moeten worden. Deze houden we bij met de 'issue tracker' van BitBucket[2]. Hierbij kan elke taak in één van de drie categoriën en in één van de vijf prioriteitsgraden ingedeeld worden. De sprintbacklog is de lijst met taken die in de huidige sprint gedaan worden. Ook deze houden wij bij met de 'issue tracker' van BitBucket, door de taak aan iemand toe te wijzen. Als een taak klaar is, kan deze als 'done' worden gemarkeerd. Een taak is pas klaar als deze goed werkt, er voldoende tests voor zijn en

[1] projects.coin-or.org/Clp
[2] bitbucket.org

de tests allemaal slagen.

4.4 Hulpmiddelen

Als versiebeheersysteem gebruiken we BitBucket. Op deze website zijn private Git repositories gratis beschikbaar. Bij de concurrent GitHub zijn ook private Git repositories beschikbaar, maar dan moet het account wel geactiveerd worden met een TU Delft e-mailadres. Op BitBucket is dit dus eenvoudiger in te stellen. We hebben voor Git gekozen en niet voor Subversion, omdat we meer ervaring hebben met Git. BitBucket heeft ook een ingebouwd issue-trackingsysteem waarvan wij gebruik gaan maken. Dit is in Subversion minder makkelijk in te stellen.

Om ervoor te zorgen dat we altijd werkende code hebben op Bitbucket, gaan we continuous integration gebruiken. Daarvoor is een server waarop Jenkins is geïntalleerd erg geschikt. In het geval dat iemand code uploadt die niet werkt, geeft Jenkins daarvan automatisch een melding, zodat wij de code kunnen verbeteren en we weer werkende code hebben op Bitbucket.

Om er voor te zorgen dat onze code goed onderhoudbaar blijft, gaan we ook een stylechecker gebruiken. De stylechecker *cppcheck* controleert statisch de code op fouten en geeft waarschuwingen wanneer er bijvoorbeeld variabelen worden gedeclareerd die nooit gebruikt worden. De stylechecker *cpplint* controleert op de Google code guideline voor C++. Ook *cpplint* is niet perfect, zo zeggen ze zelf, omdat het niet alle fouten ontdekt, maar wij denken dat dit wel een goede toevoeging is. We hebben ook naar de alternatieven *Vera++* en *cxxchecker*, maar deze waren niet gemakkelijk te installeren.

Referenties

[1] E. Ammerlaan, J. Elfers, E. Walraven, and W. Wisse. Visualisatie en ondersteuning bij planning en scheduling. 2012.

[2] C. Bakker, A. Bouter, M. den Hoedt, and W. Meerkerk. *Resource Constrained Project Scheduling*. 2014.

[3] J. Blazewicz, J. Lenstra, and A. R. Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, 1983.

[4] E. Bruurs and C. van de Louw. Onderhoudsplanner nedtrain. 2011.

[5] R. Evers. Algorithms for scheduling of train maintenance. 2010.

[6] N. Policella, A. Cesta, A. Oddi, and S. F. Smith. From precedence constraint posting to partial order schedules: A CSP approach to robust scheduling. *AI Commun.*, 20(3):163–180, 2007.

[7] K. Schwaber and J. Sutherland. *The Scrum Guide*. 2011.

[8] M. Wilson, T. Klos, C. Witteveen, and B. Huisman. Flexibility and decoupling in the simple temporal problem. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI'13, pages 2422–2428. AAAI Press, 2013.

Bijlage D Requirementsanalyse

TI3800 BACHELOR EINDPROJECT

Requirementsanalyse

Chris Bakker
Anton Bouter
Martijn den Hoedt

Mei 2014

Inhoudsopgave

1 Inleiding

In dit document worden de requirements opgesteld voor dit project. Onze opdrachtgevers, NedTrain en TU Delft, hebben een aantal eisen en wensen. We delen de requirements op in twee groepen namelijk de functionele- en de niet-functionele requirements. In de sectie 'Functionele Requirements' wordt besproken wat de tool moet kunnen en in de sectie 'Niet-functionele Requirements' worden de overige eisen aan de software gesteld.

2 Requirements

De hier opgestelde functionele requirements zijn nodig om een goed en volledig product te krijgen, zoals besproken in het Plan van Aanpak. In het Plan van Aanpak wordt een aantal features opgesomd, waarna deze features in drie verschillende categorieën opgedeeld worden. De functionele requirements worden hier weer opgedeeld in diezelfde categorieën. De niet-functionele requirements worden niet opgedeeld, aangezien de tool daar altijd aan moet voldoen. In deze sectie wordt met de tool gerefereerd naar de 'NedTrain planner'.

2.1 Functionele Requirements

Hoge Prioriteit

1. De bestaande functionaliteit moet blijven bestaan en wordt dus slechts uitgebreid.
2. De tool bevat een nieuwe solver die gebruik maakt van de chaining-methode. In oplossingen van deze solver kunnen geen resource conflicten ontstaan door het verschuiven van taken.
3. Deze nieuwe solver berekent ook door middel van een Linear Programming-solver voor elke taak een interval waarover de taak verschoven kan worden zonder enige gevolgen voor de andere taken te hebben.
4. In de gebruikersinterface zijn de flexibiliteitsintervallen, die berekend zijn door de Linear Programming-solver, zichtbaar en taken kunnen hierover verschoven worden.
5. De nieuwe solver lost probleeminstanties op in het bestaande format en geeft de oplossing in een nieuw format, zodat ook de flexibiliteitsintervallen worden meegegeven in de oplossing.

Gemiddelde Prioriteit

6. De tool moet een undo-functionaliteit hebben. Deze undo-functionaliteit maakt het mogelijk om de laatste tien veranderingen aan taken, projecten en resources ongedaan te kunnen maken. Deze acties zijn niet meer ongedaan te maken na het afsluiten van de tool.
7. De tool bevat een optie om weer te geven tot welke chains de taken horen.

Lage Prioriteit

8. De sneltoets ctr + scroll zal zorgen voor inzoomen van de tasks en recources time-line. Zoals daarvoor nu ook al de knoppen 'inzoomen' en 'uitzoomen' voor aanwezig zijn.

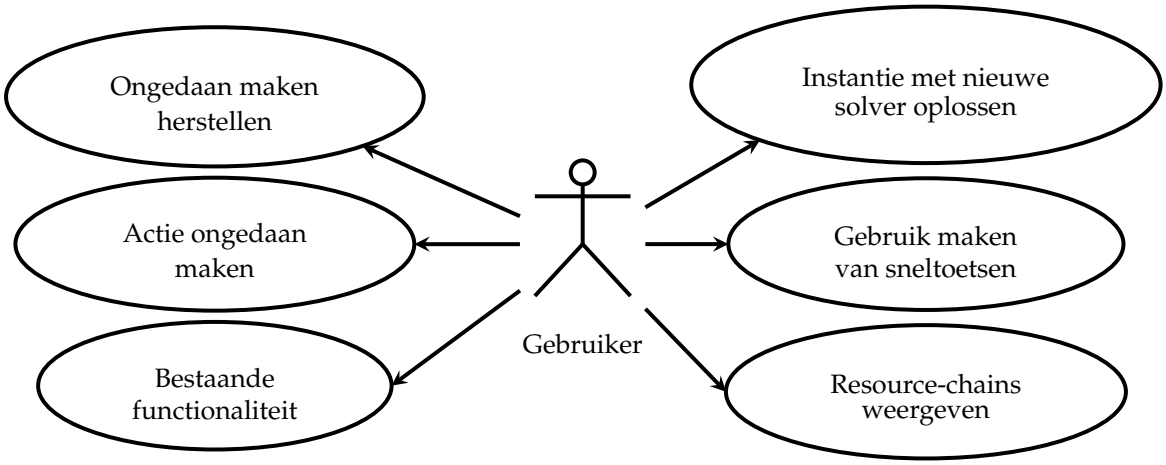
9. De sneltoets shift + scroll zal zorgen voor horizontaal scrollen in de verschillende views, waar horizontaal zoomen mogelijk is.

2.2 Niet-functionele Requirements

1. Het generen van een oplossing mag niet te lang duren. Voor de grootste geleverde probleeminstantie mag dit niet meer dan 5 minuten bedragen.
2. Gehele project moet gemakkelijk overdraagbaar zijn naar een volgende groep van ontwikkelaars. Voldoende documentatie en testcode speelt hierbij een rol.
3. De tool is te gebruiken op Windows 7 computer met minstens 200 MB vrije schijfruimte, 2 GB RAM of meer en een processor met 1 GHz of sneller.
4. De tool is te gebruiken op een computer met een Linux versie, minstens 200 MB vrije schijfruimte, 2 GB RAM of meer en een processor met 1 GHz of sneller. Op deze computer moet het dan wel mogelijk zijn om grafische interfaces te gebruiken, dit is bijvoorbeeld mogelijk op Ubuntu Desktop. De tool is niet geschikt voor bijvoorbeeld Ubuntu Server.
5. De tool moet werken op de nieuwste huidige Qt versie, welke op dit moment versie Qt 5.2.1 is.
6. De LP-solver waar de nieuwe solver gebruik van maakt moet open source software zijn, zodat dit ook voor commercieel gebruik gratis is.

3 Use Cases

Een gebruiker kan een werknemernemer zijn van NedTrain, een onderzoeker van de TU Delft of iemand anders die geïnteresseerd is in de NedTrain planner. Elke gebruiker mag evenveel doen met de tool. Er wordt dus geen onderscheid gemaakt tussen de verschillende soorten gebruikers.



4 User Stories

User Story 1

Als gebruiker **wil** ik op een knop kunnen drukken **om** de resource-chains weer te geven op het scherm. Die chains worden afgebeeld als lijnen die van taak naar taak gaan.

- *Gegeven* dat er een probleeminstantie geladen is.
- *Gegeven* dat de probleeminstantie opgelost is met de solver die chaining gebruikt.

User Story 2

Als gebruiker **wil** ik op een knop kunnen drukken **om** de flexibiliteit van taken zichtbaar te maken. Die flexibiliteit wordt weergegeven met een lijn die begint bij de vroegste begintijd en stopt bij de laatste eindtijd.

- *Gegeven* dat er een probleeminstantie geladen is.
- *Gegeven* dat de probleeminstantie opgelost is met de solver die chaining gebruikt.

User Story 3

Als gebruiker **wil** ik op een knop kunnen drukken **om** de laatste actie ongedaan te maken. **Als** ik, als gebruiker, nog een keer op diezelfde knop druk, **wil ik** dat de actie daarvoor ongedaan wordt gemaakt. Het ongedaan maken van de acties gebeurt dus van meest recent tot minst recent.

- *Gegeven* er is een actie gedaan door de gebruiker nadat de tool is gestart. Het is namelijk niet mogelijk om acties ongedaan te maken in een vorige sessie.
- *Gegeven* de actie betreft niet: het openen of sluiten van een instantie.
- *Gegeven* de actie betreft wel: een aanpassing aan een taak, een project of een resource.

User Story 4

Als gebruiker **wil** ik op een knop kunnen drukken **om** de laatste undo actie te herstellen.

- *Gegeven* ik, als gebruiker, heb een actie ongedaan gemaakt met de 'undo' knop.
- *Gegeven* de undo-actie is in deze sessie uitgevoerd. Met andere woorden de tool is na de undo-actie niet opnieuw opgestart.

User Story 5

Als gebruiker **wil** ik op een knop drukken op een tab van een instantie **om** die instantie te sluiten.

- *Gegeven* er zijn één of meerdere instanties geopend.

User Story 6

Als gebruiker **wil** ik door middel van het indrukken van de 'shift'-toets en het scrollen met de muis **om** horizontaal te scrollen.

- *Gegeven* de muis staat in het gebied wat ik, als gebruiker, horizontaal wil bewegen.
- *Gegeven* het gebied wat ik, als gebruiker, horizontaal wil scrollen, heeft een scrollbalk voor horizontaal scrollen. Met andere woorden moet er normaal dus ook de mogelijkheid zijn om horizontaal te scrollen.

User Story 7

Als gebruiker **wil** ik, de 'ctrl'-toets indrukken en tegelijkertijd scrollen met de muis **om** te kunnen in- en uitzoomen.

- *Gegeven* de muis staat in het gebied wat ik, als gebruiker, wil in- en uitzoomen.
- *Gegeven* het gebied dat in- of uitgezoomd moet worden betreft een timeline met resources en/of taken.

User Story 8

Als gebruiker **wil** ik, twee activiteiten met elkaar kunnen wisselen van plek op het scherm **door** op een knop te drukken.

- *Gegeven* er is bij een activiteit A op de knop naar beneden gedrukt *en* er is op het scherm onder activiteit A nog een andere activiteit B, dan worden activiteiten A en B met elkaar verwisseld. Activiteit A staat nu direct onder activiteit B.
- *Gegeven* er is bij een activiteit A op de knop naar boven gedrukt *en* er is op het scherm boven activiteit A nog een andere activiteit B, dan worden activiteiten A en B met elkaar verwisseld. Activiteit A staat nu direct boven activiteit B.

User Story 9

Als gebruiker **wil** ik, twee resources met elkaar kunnen verwisselen van plek op het scherm **door** op een knop te drukken.

- *Gegeven* er is bij een resource A op de knop naar beneden gedrukt *en* er is op het scherm onder resource A nog een andere resource B, dan worden resources A en B met elkaar verwisseld. Resource A staat nu direct onder resource B.
- *Gegeven* er is bij een resource A op de knop naar boven gedrukt *en* er is op het scherm boven resource A nog een andere resource B, dan worden resource A en B met elkaar verwisseld. Resource A staat nu direct boven resource B.

User Story 10

Als gebruiker **wil** ik de flexibiliteit van de oplossing zien op het scherm. De flexibiliteit wordt weergegeven door voor elke taak een flexibiliteitsinterval weer te geven. Dit flexibiliteitsinterval laat zien wanneer een activiteit *mag* beginnen en wanneer het klaar *moet* zijn. Als elke taak binnen het eigen flexibiliteitsinterval geplaatst is, dan kan elke taak volgens dit schema worden uitgevoerd. Deze flexibiliteitsintervallen worden weergegeven:

- *Gegeven* de optie 'Paint flexibility intervals' aangevinkt staat.
- *Gegeven* de probleeminstantie is opgelost met een solver die deze flexibiliteitsintervallen berekent.

User Story 11

Als gebruiker **wil** ik na het oplossen van een probleeminstantie de duur van een activiteit kunnen aanpassen door met de muis de rechter zijde van een activiteit te verslepen.

- *Als* de optie 'Paint flexibility intervals' staat aangevinkt, dan mag de activiteit niet buiten het flexibiliteitsinterval plaatsvinden.
- *Als* de optie 'Paint flexibility intervals' niet staat aangevinkt, dan mag de activiteit wel buiten het flexibiliteitsinverval, maar niet buiten het feasible interval plaatsvinden.

User Story 12

Als gebruiker **wil** ik na het oplossen van een probleeminstantie de begintijd van een activiteit kunnen aanpassen door een activiteit te verslepen.

- *Als* de optie 'Paint flexibility intervals' staat aangevinkt, dan mag de activiteit niet buiten het flexibiliteitsinterval plaatsvinden.
- *Als* de optie 'Paint flexibility intervals' niet staat aangevinkt, dan mag de activiteit wel buiten het flexibiliteitsinverval, maar niet buiten het feasible interval plaatsvinden.

User Story 13

Als gebruiker **wil** ik na het oplossen van een probleeminstantie de flexibiliteitsintervallen onzichtbaar maken door op een knop te drukken.

- *Gegeven* er worden flexibiliteitsintervallen weergegeven.

User Story 14

Als gebruiker **wil** ik dat activiteiten worden verplaatst, zodat het begin van de activiteit op hetzelfde moment is als het begin van het flexibiliteitsinterval.

- *Als* er op 'solve' geklikt is, waarbij de solver flexibiliteitsintervallen berekent.
- *Als* 'Paint flexibility intervals' wordt aangevinkt en de probleeminstantie is al opgelost met een solver die flexibiliteitsintervallen berekent.

User Story 15

Als gebruiker **wil** ik op een knop kunnen drukken om het rooster te kunnen zien. Een rooster is een schema waarbij voor elke resource unit te zien is wanneer deze nodig is.

- *Gegeven* de probleeminstantie met het chaining algoritme is opgelost.
- *Gegeven* de instantie na het oplossen niet is aangepast. Het verschuiven van taken is wel toegestaan.

User Story 16

Als gebruiker **wil** ik op een knop kunnen drukken om de informatie over een trein weer te geven. Op dit scherm staat onder andere informatie over het aantal activiteiten voor deze trein en de start- en eindtijd.

- *Gegeven* er is een instantie geopend.
- *Gegeven* de huidige instantie bevat treinen.