NTNU
Kunnskap for en bedre verden

BACHELOROPPGAVE:

OPTAROUTE – FORDELINGSALGORITME FOR KOSTNADSREDUKSJON HOS TRANSPORTBASERT NÆRING

FORFATTERE:

Helge Eriksen
Sigurd Molnes Harkjerr

Dato:

17.05.2016

SAMMENDRAG

Tittel:	OptaRoute – fordelingsalgoritme for kostnadsreduksjon hos transportbasert næring	Dato : 17.05.2016
Deltakere	Helge Eriksen Sigurd Molnes Harkjerr	
Veileder:	Ivar Farup	
Oppdragsgiver:	Electric Time Car AS	
Nøkkelord	Optimalisering, CSP, Optaplanner	

Antall sider: 77	Antall vedlegg: 7	Publiseringsavtale inngått: ja

Denne rapporten beskriver prosessen rundt utviklingen av OptaRoute - en optimaliseringsalgoritme utviklet for å redusere kostnader hos transportbaserte næringer. Programmet fungerer ved at data om en organisasjon bestående av avdelings-, ansatt-, kjøretøy- og oppdragsinformasjon mottas sammen med en definisjon av hva som skal prioriteres under optimaliseringen. Algoritmen forsøker så, basert på denne definisjonen å komme frem til den mest gunstige løsningen ved først å fordele oppdragene mellom bedriftens avdelinger. Deretter knyttes ansatte opp til kjøretøy og ruter dannes som viser rekkefølgen for hvordan oppdrag skal gjennomføres. Dette gjøres ved å forsøke ulike kombinasjoner og gradvis bevege seg mot bedre løsninger. Ved utviklingen av algoritmen ble optimaliseringsrammeverket Optaplanner benyttet. For å teste algoritmen ble det i tilegg utviklet et program som genererer tilfeldige slike datasett og en web-applikasjon som kan vise frem resultatene fra algoritmen.

ABSTRACT

Title:	OptaRoute - Distribution algorithm for cost reduction in transportation based industries	Date : 17.05.2016
Participants	Helge Eriksen Sigurd Molnes Harkjerr	
Supervisor	Ivar Farup	
Employer:	Electric Time Car AS	
Keywords:	Optimization, CSP, Optaplanner	

Number of pages: 77	Number of appendix: 7	Availability: open

This thesis describes the development process of OptaRoute - an optimization algorithm designed to reduce costs in transportation-based industries. The program works by using data about an organization consisting of department, employee, vehicle and assignment information, together with a definition of what should be given priority during the optimization. The algorithm then tries - based on this definition, to arrive at the most favorable solution by first allocating assignments to company departments. Employees are then assigned to vehicles and routes are generated showing the order in which the assignments will be conducted. This is done by trying different combinations and gradually move towards better solutions. For developing the algorithm Optaplanner, an optimization framework was used. To test the algorithm, a program that generates random organization data and a web application that can display the results of the algorithm was also developed.

Forord

Vi vil gjerne takke ETC for å ha gitt oss muligheten til å arbeide med denne oppgaven. Spesielt vil vi takke Dag L. Solhaug og Øyvind Flatval for råd, veiledning, konstruktiv kritikk, og tett oppfølging igjennom hele prosjektperioden. Vi vil også takke vår veileder Ivar Farup som på pålitelig vis gjorde seg tilgjengelig både gjennom ukentlige møter og snarvisitter og bistod med prosessveiledning og teknisk ekspertise. Monica Kristiansen fra hjemmetjenesten i Gjøvik fortjener også takk for å ha gitt gruppen nødvendig innsikt i hjemmetjenestens arbeidsmetoder slik at gjennomføringen av dette prosjektet ble mulig. Tilslutt vil vi takke familie og venner for all støtte og bistand de har bidratt med i denne prosjektperioden.

Terminologi

Constraint Satisfaction Problem – matematisk problem definert som et sett med variabler med gitte domener der variablenes tilstand må overholde fastsatte føringer. Også kalt CSP.

Optaplanner – rammeverk for løsning av CSPer.

Planentitet – begrep for objekter som får en eller flere av sine verdier endret under en optimalisering. Definert i Optaplanner som Planning Entities.

Planvariabel – variabel i en planentitet som får sin verdi endret under en optimalisering. Definert i Optaplanner som Planning Variables.

Løsningsplan – en mulig løsning på et CSP-problem i Optaplanner. Kalles her Planning Solution.

Problemfaktum – Objekt som ikke endres i løpet av en optimalisering, men som allikevel har påvirkning under poengberegning av løsningsplaner algoritmen produserer. Kalles i Optaplanner Problem Facts.

Harde føringer – begrensninger satt på planvariabler som ikke kan brytes. Tilsvarer Hard Constraints i Optaplanner.

Myke føringer – begrensninger satt på planvariabler som det kun er ønskelig om ikke brytes. Tilsvarer Soft Constraints i Optaplanner.

Skyggevariabel – variabel som utledes ved bruk av verdien til en planvariabel. Kalles i Optaplanner Shadow Variable.

Verdileverandør – Metode definert i Optaplanner som angir domenet for planvariabler.

Anker – skyggevariabel som alltid peker til proglemfaktumet som er roten av en lenket kjede av planentiteter.

Toveisskyggevariabel – skyggesiden i et variabelpar bestående av en genuin planvariabel og en skyggevariabel.

Innholdsfortegnelse

1 Innledning

Enhver organisasjon står ovenfor problemer knyttet til planlegging. Bedrifter ønsker å oppnå størst mulig profitt gjennom å tilby produkter og tjenester. Det som tilbys leveres imidlertid med et begrenset antall ressurser som råvarer, tid, ansatte og penger. Det hele kompliseres av at ytterligere forbehold må tas. Spesifikke oppgaver kan kreve ansatte med spesiell kompetanse, logistiske utfordringer må løses og de ansatte er kun tilgjengelige i oppsatte arbeidstider. Dårlig planlegging leder til nedsatt produktivitet som igjen leder til reduserte inntekter og økte kostnader. Mye kan derfor spares gjennom å effektivisere bruken av organisasjonens ressurser.

I forbindelse med bacheloroppgavene som ble gjennomført våren 2016, utlyste ETC et oppdrag bestående av å utvikle en fordelingsalgoritme for oppdragsbaserte transportnæringer. Et eksempel på en slik næring er hjemmetjenesten, der tjenester som hjelp til medisinering, sårbehandling, kosthold, veiledning og generell praktisk bistand tilbys til kunder på deres hjemmeadresse.

ETC, Electric Time Car AS er et IT-selskap lokalisert i Gjøvik. Selskapet ble etablert i 2003 og leverer løsninger relatert til administrasjon av fellesbiler for bedrifter. Deres hovedprodukt, CarAdmin lar brukere holde daglig oppfølging av kjøretøy for hele bedriftens bilpark. Daglig leder for ETC, Dag L. Solhaug, ønsker med utlysningen av oppgaven å se på muligheter for å kunne tilby en tjeneste som kan hjelpe transportbaserte næringer å kutte kostnader gjennom optimalisering av ressursbruk.

1.1 Problemområde

Oppdragsbasert transportnæring opererer innenfor et geografisk område og kan være inndelt i ulike avdelinger. Hver avdeling kan så ha ansvar for mindre delområder. Disse delområdene er ofte rigide slik at tjenester som skal leveres innenfor området alltid gjøres av dette områdets avdeling, uavhengig av om ressurser kunne ha blitt brukt bedre ved å ta i bruk arbeidskraft fra andre avdelinger. To nabohus som ligger i skillet mellom slike delområder kan derfor i verste fall bli dekket av to ulike avdelinger. Avdelingene har som regel varierende antall biler og ansatte tilgjengelige.

På toppen av dette kommer tilleggskrav som kan være gjeldene for ulike tjenesteytere. Hjemmetjenesten opererer for eksempel med vedtak som tildeles den hjelpetrengende. Vedtakene inkluderer hvilken type hjelp som skal utføres, en liste over konkrete oppgaver og kompetansekrav satt til den ansatte som utfører oppdraget. Vedtaket inneholder også et tidsrom oppgaven skal utføres i, hvor lenge hjemmebesøket skal vare og antall personer som trengs for å utføre oppgaven. Det blir også opprettet en primærkontakt for hver hjelpetrengende, der det er ønskelig at besøket forekommer av den ansatte som har blitt tildelt denne rollen.

Oppdragsbasert transportnæring, slik som hjemmesykepleien, benytter seg i stor grad av manuell ruteplanlegging i dag. Faktorer slik som beskrevet ovenfor påvirker ruteplanleggingen. Dette medfører ofte mye arbeid fordi planleggingen fort kan bli kompleks og tidkrevende. I tillegg sitter ruteplanleggerne med informasjon som ikke registreres i datasystemet men som er avgjørende for planleggingen. Dette kan blant annet være hvilke ansatte som best håndterer vanskelige pasienter og derfor burde utføre oppdrag knyttet til dem.

1.2 Oppgavedefinisjon

Oppgaven gikk ut på å utvikle en automatisert algoritme som kalkulerer de beste kjørerutene ved å fordele oppdragene på en gunstig måte mellom de tilgjengelige kjøretøyene, ansatte og avdelingene. Dette skulle gjøres ved å følge et sett av kriterier som skulle optimaliseres. Eksempler på slike kriterier inkluderer minst mulig kjørte kilometer per bil per ansatt eller kortest mulig ventetid imellom oppdrag.

Algoritmen skulle utvikles som en selvstendig modul som tar imot og eksporterer data om et problem på et bestemt format. Input til algoritmen kommer fra et eksternt system, som for eksempel et journal- eller bestillingssystem. Resultatet eksporteres deretter tilbake til dette tredjepartsprogrammet gjennom et API.

Algoritmen kjøres typisk før en arbeidsdag, men skulle helst også ta hensyn til unntakstilfeller som for eksempel at en ansatt blir syk i løpet av en dag. Algoritmen ville i et slikt tilfelle kjøres på nytt, men da også ta hensyn til de oppdaterte dataene.

Det var viktig at algoritmen dekket kravene satt av hjemmetjenesten i Gjøvik. En slik tjeneste inneholder et komplisert sett med krav. Ved å dekke Hjemmetjenestens behov ble det antatt at algoritmen vil kunne generaliseres og brukes av en rekke andre transportbaserte tjenester, slik som post- eller matleveranse. Kravene ble delt inn i to kategorier. Harde og myke føringer. Navnendringen fra krav til føring ble foretatt for å samsvare med navneterminologi brukt innenfor fagfeltet. Harde føringer definerer føringer der løsningen ikke kan brukes dersom disse brytes, for eksempel vil en løsning som bryter føringen om kompetanse hos ansatte som utfører oppdrag ikke kunne benyttes i virkeligheten, disse føringene optimaliseres derfor først. Harde og myke føringer vektes avhengig av ønsket utfall for bedriften ved bruk av algoritmen. Bedriften skulle selv kunne velge hvordan de vil vekte disse. Alle føringene måtte også kunne utelukkes fullstendig dersom de ikke var relevante.

Det skulle i tillegg utvikles et program som benytter seg av data fra algoritmen til å generere tidslinjer med oppdrag per bil, timelister per ansatt og rutevisning på kart. Dette programmet skulle utvikles kun for eget bruk for å teste resultatene fra algoritmen.

1.3 Målgruppe

Målgruppen for denne bachelorrapporten har vært IT-studenter og profesjonelle med interesse for optimaliseringsproblemer og hvordan bruk av eksisterende rammeverk kan benyttes for å løse slike problemer. Oppgaven vil også fungere som dokumentasjon for løsningen som overleveres oppdragsgiver etter prosjektets slutt.

Selve algoritmens målgruppe var oppdragsgiver. ETC ønsker å benytte gruppens løsning som en prototype som kan vise hva optimaliseringsrammeverk kan benyttes til. De ønsker også å benytte kildekoden gruppen har skrevet for videreutvikling av løsningen, eller som grunnlag for en ny løsning som kan selges til bedrifter med behov for optimaliseringsprogramvare.

1.4 Formål

Resultatmål

Gruppen definerte følgende resultatmål for systemet:

- Det finnes en enkel modul for å fore algoritmen med data.
- Algoritmen beregner tilnærmede optimale kjøreruter for ansatte og kjøretøy.
- Systemet kan eksportere sluttresultatet av optimaliseringen på en fornuftig måte slik at tredjeparts programmer kan benytte seg av dataen.
- De optimaliserte oppdragsrutene skal kunne vises frem på en hensiktsmessig måte.

Effektmål

Ved overtagelse av kildekoden og dokumentasjonen av arbeidet var målet at prosjektet skulle bidra til at ETC enklere kunne utvikle egne optimaliseringsalgoritmer eller videreutvikle gruppens løsning slik at ETCs kunder i fremtiden kan nyte godt av et utvidet tilbud. Ved bruk av algoritmen er målet at følgene fordeler skal kunne være gjeldene for disse kundene. De faktiske målene vil variere fra kunde til kunde.

- Erstatte behovet for manuell ruteplanlegging.
- Redusere antall kilometer kjørt per bil per ansatt.
- Redusere svinn av ressurser i form av at ansatte må vente på tilgjengelig bil.
- Redusere antall biler som trengs for å utføre oppdragene.
- Redusere behovet for arbeidskraft ved å luke bort overflødige arbeidstimer brukt under transport.
- Øke samværtiden med kunde.
- Redusere stress hos de ansatte.

Læringsmål

Ved å utføre prosjektet ønsket gruppen å få erfaring med, eller tilegne seg kunnskap om:

- Å jobbe strukturert og systematisk ved hjelp av god prosjektstyring og arbeidsmetodikk på et større prosjekt.
- Å jobbe tett med en reell oppdragsgiver.
- Å arbeide med kartdata.
- Å utvikle optimaliseringsalgoritmer.

1.5 Rammer

For å gjøre ruteberegninger skulle algoritmen benytte seg av en ferdig utviklet kart- og routingtjeneste. Disse tjenestene måtte kunne skiftes ut for å gi fleksibilitet dersom problemer med tjenestene skulle oppstå. Tjenestene måtte også fungere uten tilgang til nett, da algoritmen var nødt til å være rask. Flere spørringer over internett ville ført til for lang responstid.

Kildekoden til prosjekter kunne heller ikke være åpent tilgjengelig i sin helhet. Det kunne derfor ikke benyttes kildekode under lisenser hvor dette var ett krav. Dersom en slik restriksjon fant seg gjeldende ved valg av kart- eller routingtjeneste meddelte oppdragsgiver at de var villige til å stille midler disponible for kjøp av nødvendig lisens som tillot lukket bruk. Ved utviklingen av prosjektet

gjaldt prosjektavtalen mellom NTNU ved avdeling for informatikk og medieteknikk, ETC og gruppens medlemmer. Avtalen fastsetter blant annet at studentene ikke kunne publisere kode uten samtykke fra ETC, foruten den publiseringen som gjøres i regi av NTNU i Gjøvik.

På grunn av gruppens størrelse ble det bestemt sammen med oppdragsgiver at API-et som skulle knytte seg opp til algoritmen ikke skulle utvikles av gruppen og at fokuset i hovedsak skulle ligge på optimaliseringen. Organisasjonsdataen som brukes av algoritmen blir derfor lest direkte fra filsystemet og mottas ikke gjennom et slikt API.

1.6 Gruppens bakgrunn

Bachelorgruppen som utførte oppdraget utgjorde to studenter fra NTNU i Gjøvik som studerer til dataingeniør. Studentene har liten erfaring utenom den erfaringen de har fått gjennom studier og har kun arbeidet på mindre prosjekter før dette. Prosjektstyring har tidligere utelukkende blitt gjort på teoretisk nivå. Studentene har fulgt standard utdanningsløp for dataingeniører på Høgskolen i Gjøvik/ NTNU i Gjøvik. Femte semester valgte de begge å følge emnene programvareutvikling og mobil programvareutvikling, som tredje valgemne valgte Sigurd Molnes Harkjerr Matematikk 3 og Helge Eriksen valgte Programvaresikkerhet.

1.7 Avgrensninger

Gruppen ble i løpet av prosjektets gjennomføring nødt til å introdusere flere nye avgrensninger. Det ble bestemt at algoritmen ikke skulle ta seg av geokoding – oversettelse fra adresseinformasjon til kartkoordinater. Algoritmen opererer derfor kun med lengde- og breddegrad koordinater. Adressene som sendes til algoritmen må altså på forhånd være geokodet når algoritmen skal arbeide med disse.

I arbeidet med å sette seg inn i og forstå optimaliseringsalgoritmer ble gruppen etter hvert overbevist om at det ikke var i oppdragsgivers eller gruppens interesse å utvikle en algoritme helt fra bunnen av. Isteden ønsket gruppen å benytte seg av ett optimaliseringsrammeverk for å utvikle en mer komplett løsning i tidsperioden som var tilgjengelig.

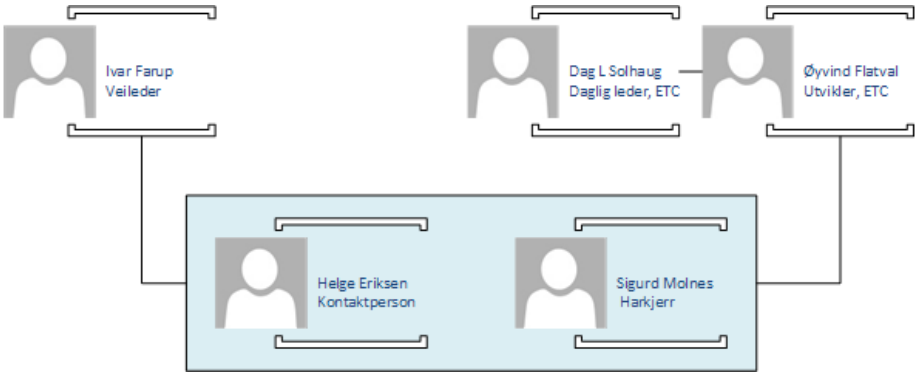
Det ble i tillegg utover i prosjektet innført nye avgrensninger ettersom gruppen fikk et klarere overblikk av oppgavens omfang. For en beskrivelse av disse avgrensningene se 8.1 Diskusjon, Avgrensninger som følge av tidsbegrensninger.

1.8 Statusmøter og Beslutningspunkter

På grunn av oppgavens kompleksitet ble det bestemt at gruppen skulle ha ukentlige møter med veileder og oppdragsgiver. Ukentlige møter ble bestemt for at veileder og oppdragsgiver tidlig skulle ha mulighet til å korrigere kursen til gruppen hvis de oppdager problemer som vil vanskeliggjøre eller hindre ett godt sluttprodukt. Møtetidspunktet ble satt til tirsdag morgen med veileder og tirsdag ettermiddag med oppdragsgiver.

Viktige beslutninger knyttet til prosjektet, som for eksempel valg av karttjeneste, skulle dokumenteres med begrunnelse for valget som ble tatt. Beslutninger knyttet til hvilke oppgaver som ble gjennomført under utviklingsdelen av prosjektet ble tatt av gruppemedlemmene selv, med innspill fra oppdragsgiver.

1.9 Ansvarsforhold og Roller



FIGUR 1 : ROLLEDIAGRAM

Prosjektet ble organisert med en flat ledelsesstruktur som skulle fremme selvstendig arbeid og beslutningstagning fra hvert enkelt gruppemedlem som vist i figur 1. Helge Eriksen fungerte likevel som prosjektleder og kontaktperson for gruppen utad. Sigurd Molnes Harkjerr var ansvarlig for nettsiden og for å notere under møter.

Dag L. Solhaug og Øyvind Flatval representerte Electric Time Car AS som var oppdragsgiver, og bisto gruppen med teknisk og organisatorisk hjelp for å nå frem til ett best mulig sluttresultat. Dette gjorde de gjennom ukentlige møter og ved å være tilgjengelig for spørsmål over e-post.

Monica Kristiansen fra Hjemmetjenesten i Gjøvik bistod med generell informasjon om deres arbeidsmetoder gjennom et innledende møte og bidro med data om hjemmetjenesten for kvalitetssikring ved ett avsluttende møte.

Ivar Farup fra NTNU i Gjøvik var veileder for gruppen og bistod med prosessveiledning og teknisk ekspertise. Dette ble primært gjort ved ukentlige møter, men han var også tilgjengelig på epost og behjelpelig ved flere dropp-inn besøk.

1.10 Verktøybruk

Under følger en oversikt over verktøyene som ble benyttet både for prosjektstyring og utvikling av prosjektet. Merk at eksterne biblioteker som ble benyttet ikke er listet opp her. For en oversikt over disse se kapittel 6.1 Algoritme og 6.2 Presentasjonsapplikasjon for henholdsvis algoritmen og presentasjonsapplikasjonen.

Google Docs - En web-basert kontorpakke som gjør det enkelt å samskrive dokumenter. Google Docs ble brukt for utarbeiding av alle dokumenter, inkludert møtereferater, prosjektplaner og rapport.

Microsoft Word – Tekstbehandler utviklet av Microsoft. Programmet ble benyttet for arbeid og ferdigstilling av rapporten.

Bitbucket - Et web-basert vertssystem for prosjekter som benytter seg av GIT som system for versjonskontroll. All kildekode tilknyttet prosjektet ble lagret med tilhørende filhistorikk i et lukket repository. BitBucket ble benyttet for å holde oversikt over endringene som ble utført i kildekoden, samt for å gi mulighet til å gå tilbake til tidligere versjoner av koden.

Eclipse IDE - En Open Source IDE administrert av Eclipse Foundation primært brukt for utvikling av javaapplikasjoner. Gruppen tok i bruk tilgjengelige verktøy som kodeeditor og debugger ved at kildekode i Java ble skrevet ved hjelp av Eclipse.

SonarLint - En plattform for inspeksjon av kode for å opprettholde høy kvalitet i koden. Verktøyet ble benyttet for å avdekke mangler i koden, slik som at koden ikke fulgte anbefalte standarder, var problematisk med tanke på sikkerhet eller i noen tilfeller inneholdt feil.

Trello - Et web-basert prosjektstyrings verktøy. Trello ble benyttet for å holde oversikt over arbeidsoppgaver som skulle utføres, vise ansvarsfordelingen i gruppen og å vise fremgang i arbeidet.

Microsoft Project Professional 2016 - Program utviklet av Microsoft som spesialiserer seg på utarbeiding av dokumenter relatert til prosjektstyring og planlegging. Gruppen brukte programmet for å lage og vedlikeholde Gantt-skjema.

Microsoft Visio 2016 - Program for generering av diagrammer og vektorgrafikk. Programmet ble benyttet til å lage generelle diagrammer til bruk i rapport og prosjektplaner.

1.11 Disposisjon

Innledning – Gir en innledende forklaring av prosjektets art, introduserer hvilke parter som var involvert i prosjektet og formålet bak gjennomføringen.

Arbeidsmetode og utviklingsprosess – Begrunner valg av utviklingsmetodikk og definerer endringer gjort for å tilpasse den valgte utviklingsmetodikken for prosjektet.

Kravspesifikasjon – Gir et oversiktsbilde av systemene og hvilke krav som settes til dem og definerer grensesnittet inn mot og ut fra algoritmen.

Teori og valg av teknologi – Gir en oversikt over relevant teori og de mest vesentlige teknologiene som ble brukt i prosjektet. Diskuterer hvorfor teknologiene ble tatt i bruk, hvordan de fungerer og en drøfting rundt alternative valg.

Design – Inneholder designvalg for systemet og drøfting rundt hvorfor disse valgene er tatt med hensyn på teknologier som er tatt i bruk.

Implementasjon – En beskrivelse av hvordan prosjektet ble implementert med oversikt over filstrukturer, hvilke eksterne avhengigheter som finnes i prosjektet og konkrete klassebeskrivelser.

Kvalitetssikring – Diskuterer hvilke verktøy og virkemidler som ble benyttet for kvalitetssikring av prosjektet.

Diskusjon og resultater – Diskuterer valg gjort underveis i prosjektet, drøfter i hvilken grad prosjektmålene ble gjennomført, hva som burde blitt gjort annerledes og forslag til videre arbeid. Kapittelet inneholder også en evaluering av gruppens arbeid.

Konklusjon – Oppsummerer oppgaven, dens utfordringer og sluttresultatet for prosjektet.

2 Arbeidsmetode og utviklingsprosess

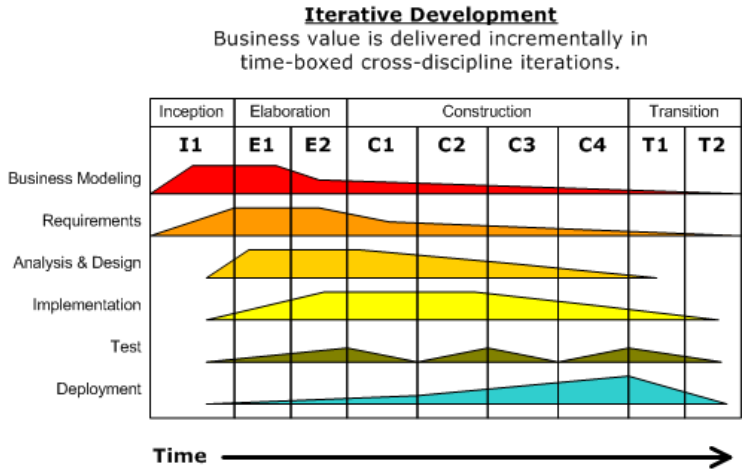
I starten av prosjektet ble det utarbeidet en forprosjektrapport hvor det ble identifisert karakteristikker ved oppgaven som gruppen mente var styrende for valg av utviklingsmodell. Disse karakteristikkene var:

- Fast tidsfrist: bachelorprosjektet hadde en fastsatt dato for når oppgaven skulle leveres.
- Få utviklere: prosjektet skulle utføres av en liten gruppe bestående av kun to studenter.
- Ingen spisset kompetanse: utviklerne hadde liknende utgangspunkt og få egne spesialiseringsområder. Dette gjorde at de enkelt kunne ta over eller bidra til hverandres arbeid dersom nødvendig.
- Liten sannsynlighet for endring av kravspesifikasjon: det ble forventet at hovedpunktene rundt kravspesifikasjonen ville være utarbeidet tidlig, og at disse med liten grad av sannsynlighet ville endre seg nevneverdig.
- Stor tilgang til kunde: utviklerne hadde ukentlige møter med oppdragsgiver.
- Høye dokumentasjonskrav: prosjektet skulle utføres som del av en bacheloroppgave og dokumentasjon ville være viktig for oppdragsgiver ved en eventuell videreutvikling av løsningen. Dette satt høye krav til dokumentasjon av hele prosessen.
- Uerfarne utviklere: gruppens medlemmer hadde lite erfaring med større prosjekter i forhold til utvikling og prosjektstyring. En mulig risiko som følge av dette var usikkerhet ved estimering av tidsbruk.

På bakgrunn av punktene over og planlegging av tidsbruk gjennom videre arbeid med Gantt-skjema, valgte gruppen å utarbeide en forenklet variant av utviklingsmodellen Rational Unified Process, også kalt RUP for bruk i dette prosjektet. RUP definerer seks fagdisipliner som kategoriserer oppgaver etter hvilke fokusområder de tilhører (Shuja & Krebs, 2008). Gruppen avgrenset og spisset disse fagdisiplinene på en slik måte at de, etter gruppens mening, beholdt hovedmomentene fra RUPs egne definisjoner samtidig som de ble tilpasset gruppens størrelse og behov.

- Business Modeling: kontakt med kunde.
- Requirements: forståelse av hva systemet skal gjøre.
- Analysis & Design: forståelse av hvordan systemet skal realiseres.
- Implementation: realisering av systemet.
- Test: kvalitetssikring, testing av enkeltstående komponenter og testing på tvers av algoritmens komponenter.
- Deployment: overføring av kodebase fra gruppen til oppdragsgiver for videreutvikling.

RUP definerer en rekke artefakter - dokumenter og diagrammer som utarbeides som følge av å benytte utviklingsmodellen (Rational Software Corporation, 2002). Dette var en av faktorene som talte for valget, men de mange artefaktene medfører også en fare for overdreven dokumentasjonsfokus på bekostning av fremgang. Alle artefaktene tilgjengelig i RUP ble derfor ikke benyttet. Gruppen foretok en vurdering av de artefaktene gruppemedlemmene anså som best egnet til å styre og dokumentere arbeidet. Artefaktene ble grunnlaget for tre hoveddokumenter: prosjektplan, kravspesifikasjonsdokument og designdokument. Disse dokumentene dannet så grunnlaget for selve bachelorrapporten.



FIGUR 2 : RUPFASER OG FAGDISIPLINER (FRIGITT UNDER PUBLIC DOMAIN)

Faseinndelingen inn i de fire fasene Inception, Elaboration, Construction og Transition som definert av RUP (Staffordshire University, 2008, s. 3) ble beholdt, se figur 2. Gruppen utarbeidet en definisjon for hva hver fase skulle innebære og hvilke milepæler de ulike fasene skulle lede opp til.

Inception: I denne fasen skulle prosjektet utredes. Dette skulle gjøres gjennom utvikling av en prosjektplan. I fasen ble det planlagt å ha høyt fokus på fagdisiplinene Business Modeling og Requirements. Dette skulle gjøres ved å gjennomføre tidlige møter med oppdragsgiver for å klargjøre oppgavens detaljer. Fasens milepæl ble i henhold til RUP kalt Life Cycle Objective og ble definert nådd ved godkjent prosjektplan. På grunn av krav satt fra NTNU i Gjøvik om tidspunkt for innlevering av denne planen ble det bestemt å flytte tidsfristen for denne milepælen fra avslutningen av Inception-fasen til litt ut i Elaboration fasen. Dette ble også gjort fordi gruppen så det som sannsynlig at det kom endringer til planen som følge av et møte med hjemmetjenesten i Elaboration fasen.

Elaboration: I denne fasen skulle prosjektets virkemåte og krav settes i fokus. Det var i denne fasen at kravspesifikasjonsdokumentet og designdokumentet for algoritmen skulle ta form. Også her var Business Modeling og Requirements fagdisiplinene i bruk, blant annet gjennom møte med hjemmetjenesten og oppdragsgiver for ytterligere utredelse av krav for oppgaven. Fagdisiplinen Analysis & Design var også i fokus under designet av løsningen. Fasen skulle ende i milepælen Life Cycle Architecture, der de utviklede planene skulle gjennomgås og sjekkes for eventuelle hull eller mangler som ville påvirke senere faser. Dersom ingen slike mangler ble oppdaget skulle milepælen anses som nådd.

Construction: Selve utviklingen av algoritmen skulle foregå i denne fasen. Gruppen planla å fortsette med ukentlige møter med oppdragsgiver for å sikre fremdrift i prosjektet og for å korrigere kursen for ikke å havne i eventuelle fallgruver som kunne oppstå. Business Modeling ville derfor ha et større fokus i denne fasen enn ved tradisjonell bruk av RUP. Algoritmedesign skulle også fortsette å være en viktig faktor i denne fasen for å ta høyde for mindre kravendringer, implementasjonssvakheter og nye begrensninger. Enhetstester skulle utvikles parallelt med skriving av kode. Disse faktorene medførte at Business Modelling, Analysis & Design, Implementation og Testing var spesielt viktige fagdisipliner i denne fasen. Det ble hentet inspirasjon fra utviklingsmodellen Scrum og dens inndeling i sprintsykler (Sommerville, 2011, s. 73) ved at Construction-fasen ble inndelt i ukentlige sprints. Dette for å gjøre gruppen mer mottakelige for endringer som kunne oppstå underveis og for å sikre fremdrift. Sprint inndelingen skulle også være med på å sikre at gruppen hadde ett produkt å levere,

selv om de ikke ble ferdige med hele løsningen. Andre detaljer fra Scrum slik som Product Owner og burn down chart ble utelatt men oppdragsgiver hadde mulighet til å komme med ønsker og råd på de ukentlige møtene. Det endelige valget stod allikevel hos gruppen. Etter planen skulle Construction-fasen ende med milepælen Initial Operation Capability der all funksjonalitet skulle være ferdigutviklet med tilhørende tester.

Transition: I denne fasen skulle test-fagdisiplinet være i fokus gjennom kvalitetssikring av algoritmen. Gruppen ønsket å sammenligne resultater fra hjemmesykepleiens manuelle planlegging slik det blir gjort i dag med algoritmens sluttresultat på reelle data. Slik gruppen anså det ville deployment-fagdisiplinet få en liten rolle i prosjektet da sluttresultatet ikke skulle direkte til forbruker, men videreutvikles hos oppdragsgiver. Ferdigstillingen av rapporten skulle isteden få en sentral rolle i denne fasen og skulle anses som Product Release milepælen.

For å minimere risiko og øke produktivitet bestemte gruppen seg også for å følge RUPs seks idealer (Rational, 2005, s. 2).

- Utvikle iterativt: Gjennom sprintinndeling av construction-fasen for å minimere konsekvensene av endring av krav.
- Håndtere krav: Fokusere på brukere, i hovedsak hjemmetjenesten, og deres krav til algoritmen.
- Modularisere: Lage programvaren og algoritmen modulær slik at det kan utvikles i frittstående komponenter.
- Visuell modellering: Høyt fokus på utarbeidelse av relevante diagrammer og modeller som for eksempel UML klasse diagram og use-case diagrammer.
- Kvalitetssikring: Ha testing i bakhodet under hele utviklingsperioden, ikke kun i transition-fasen.
- Kontroll av endringer: Benytte seg av versjonskontroll for å sikre at endringer synkroniseres mellom gruppemedlemmene.

2.1 Prosjektets Gjennomføring

Gruppen fulgte en arbeidsuke fra tirsdag til mandag. Dette valget ble tatt på grunn av at de viktige møtene med oppdragsgiver og veileder ble holdt på tirsdager og fordi disse møtene ville være styrende for fremtidig arbeid. Av samme grunn ble også deler av tirsdagen brukt til å planlegge arbeidsuken, spesielt under utviklingsperioden av prosjektet.

Prosjektperioden ble delt inn i fire faser, forprosjekt, utvikling, rapport og muntlig presentasjon. Disse fasene, med unntak av muntlig presentasjon, ble så fordelt på de fire fasene i RUP beskrevet over. Forprosjektet ble gitt en tidsramme på 31 arbeidsdager og ble inndelt i Inception- og Elaboration-fasene fra RUP. Da det originalt var planlagt at gruppen skulle utvikle hele algoritmen fra bunnen av så gruppen det nødvendig å tilegne god tid til disse fasene. Dette ble gjort for å sikre at det ble klarhet i hva som skulle utvikles og nok tid til å sette seg inn i- og lære nødvendige konsepter og teknologier. Dette viste seg å være gunstig da gruppen etter hvert innså at det ikke var hensiktsmessig å lage hele algoritmen fra bunnen av men isteden valgte å ta i bruk et optimaliseringsrammeverk, se 4.2 Bruk av optimaliseringsrammeverk for en beskrivelse rundt dette valget. Etter at valget falt på et optimaliseringsrammeverk gikk gruppen over i en periode der fokuset ble satt på å lære- og forstå rammeverket, samt tilpasse designdokumentet til det valgte rammeverket.

Selve utviklingsperioden ble gjennomført i sprinter hvor arbeidsoppgaver ble prioritert og utført av gruppens medlemmer med innspill fra oppdragsgiver. Fasen endte i milepælen Initial Operational Capability hvor gruppen anså seg som ferdig med implementasjon av ny funksjonalitet. Etter 19. april gikk gruppen inn i Transition fasen hvor de skiftet fokuset fra utvikling til dokumentasjon av løsningen og kvalitetssikring.

Tabell 1 viser viktige hendelser og beslutningspunkter gjort under gjennomføringen av prosjektet. Tabellen viser dato, detaljer rundt hendelsen, eller beslutningen og eventuelle henvisninger til utdypende materiale dersom dette er relevant.

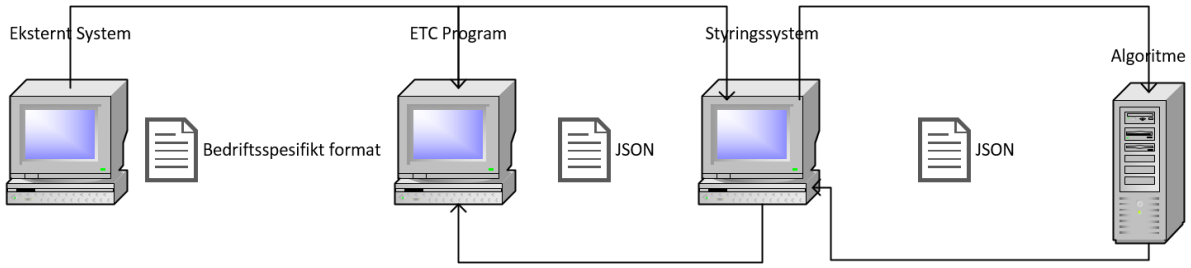
TABELL 1: VIKTIGE HENDELSER OG BESLUTNINGSPUNKTER UNDER UTVIKLINGEN AV PROSJEKTET

Dato	Beslutningspunkt	Henvisning
12.01.16	Første møte med oppdragsgiver. Gruppen avtalte sammen med oppdragsgiver rammene for oppgaven. Det ble enighet om at utvikling av selve algoritmen skulle komme i fokus, men at en modul for å vise frem resultatet også ville være fornuftig å implementere. Oppgaven ble også utdypet.	- Vedlegg F – Møtereferater, Møtereferat 1 - ETC
19.01.16	Gruppen begynte å se på kravspesifikasjonen denne uken.	- 3 Kravspesifikasjon
21.01.16	Gruppen og Øyvind Flatval fra ETC hadde møte med hjemmetjenesten for å utarbeide kravspesifikasjon fra hjemmetjenestens perspektiv	- Vedlegg F – Møtereferater, Møtereferat 1 - Hjemmetjenesten
25.01.16	På bakgrunn av møtet med ETC ble det bestemt at oppgaven skulle ta hensyn til kompatibilitet.	- Vedlegg F – Møtereferater, Møtereferat 3 - ETC
02.02.16	Det ble avtalt med oppdragsgiver at gruppen ikke skulle ta seg av geokoding.	- Vedlegg F – Møtereferater, Møtereferat 4 - ETC
10.02.16	Gruppen besluttet at kravspesifikasjonen var såpass utarbeidet at de kunne begynne på designdokumentet denne uken.	- Vedlegg F – Møtereferater, Møtereferat 4 - Veileder
19.02.16	Det ble diskutert med oppdragsgiver å skifte fokuset på oppgaven fra design av algoritme til bruk av optimaliseringsrammeverk for å gjennomføre en større del av oppgaven. ETC sa seg positive til dette. Oppdragsgiver trodde at det ville medføre at oppgaven kunne utvides med en ny langsiktig modus som kunne se på besparelser ved å bytte om, eller redusere biler og ansatte på de forskjellige depotene.	- Vedlegg F – Møtereferater, Møtereferat 6 - ETC
19.02.16	Valget av optimaliseringsrammeverket falt på Optaplanner og gruppen begynte å tilpasse tidligere designdokument til å passe med rammeverket. Valget medførte også at kodespråket skiftet fra C# til Java.	
23.02.16	Gruppen bestemte seg for å dele fordelingsalgoritmen i to faser, en hvor oppdragene blir fordelt på depoter og en hvor oppdragene blir fordelt på ansatte- og biler. Dette ble gjort fordi gruppen trodde det ville bli for tidkrevende å laste alle koblinger mellom alle oppdrag. Delingen ble også gjort for å gjøre læringen av Optaplanner lettere.	- 8.1 Diskusjon, Valg av faseinndeling
01.03.16	Endelig valg av karttjeneste ble bestemt og valget falt på Graphhopper med kartdata fra Open Street Map.	- Vedlegg F – Møtereferater, Møtereferat 7 - ETC - 4.4 Valg av kart- og routingtjeneste
08.03.16	Gruppen har de siste ukene så smått begynt å implementere basert på ett lett designdokument da mye måtte testes opp mot Optaplanner. Gruppen har også begynt å utvikle en problemgenerator for å lage problemsett til algoritmen	- Vedlegg F – Møtereferater, Møtereferat 8 - ETC
15.03.16	Gruppen informerte ETC om at de ble nødt til å avgrense oppgaven enda mer. Optimalisering av hvem ansatt som skal i hvem bil ble nødt til å forenkles. Det ble også bestemt at modulen for å redusere antall ansatte og biler som trengs på hvert depot måtte utelates.	- Vedlegg F – Møtereferater, Møtereferat 9 - ETC
05.04.16	Gruppen fant en god metode for å representere besøk til depoter ved å bruke en boolsk verdi. Gruppen ønsket å endre implementasjon slik at den ikke ble delt i to faser, men så nå at det ikke var tid.	- Vedlegg F – Møtereferater, Møtereferat 10 - ETC - 5.1 Design av Algoritmen, Rutefordelingsfasen
12.04.16	Daglig leder i ETC ønsket at gruppen skulle se på om gruppen kunne legge til en mulighet for å redusere antall ansatte og biler brukt i den modusen som er utviklet som en slags tilleggsmodul.	- Vedlegg F – Møtereferater, Møtereferat 11 – ETC - 8.1 Diskusjon, Avgrensninger som følge av tidsbegrensninger
19.04.16	Gruppen var i stand til å implementere enkle regler for å inkludere ønskene fra ETC.	- Vedlegg F – Møtereferater, Møtereferat 12 - ETC
25.04.16	Gruppen hadde 2. møte med hjemmesykepleien. Her håpet de å få tilgang til reelt datasett eller i hvert fall lage mer sannsynlig datasett.	- Vedlegg F – Møtereferater, Møtereferat 2 - Hjemmetjenesten

3 Kravspesifikasjon

3.1 Oversikt over systemet

Hver bedrift som vil benytte seg av optimaliseringsalgoritmen vil måtte oppgi data fra deres systemer - eksempelvis deres bestillings- eller journalsystem. Dataen sendes til et program som blant annet har i oppgave å omgjøre informasjonen til et format som algoritmen kan operere med. Det vil finnes et slikt program for hver bedrift som ønsker å benytte seg av algoritmen og disse programmene vil utvikles av ETC. Etter at dataen fra bedriftens egne systemer er prosessert og omgjort, sendes dette til et program som heretter vil bli omtalt som styringssystemet. Styringssystemet vil ha ansvar for å motta dataen fra ETC-programmene på en sikker måte og sende den videre til algoritmen. Algoritmen vil benytte datagrunnlaget til å danne en optimalisert løsning, for deretter å returnere dette til styringssystemet. Styringssystemet returnerer så resultatet tilbake til ETC programmet. Dataflyten mellom de ulike systemene er vist i figur 3.

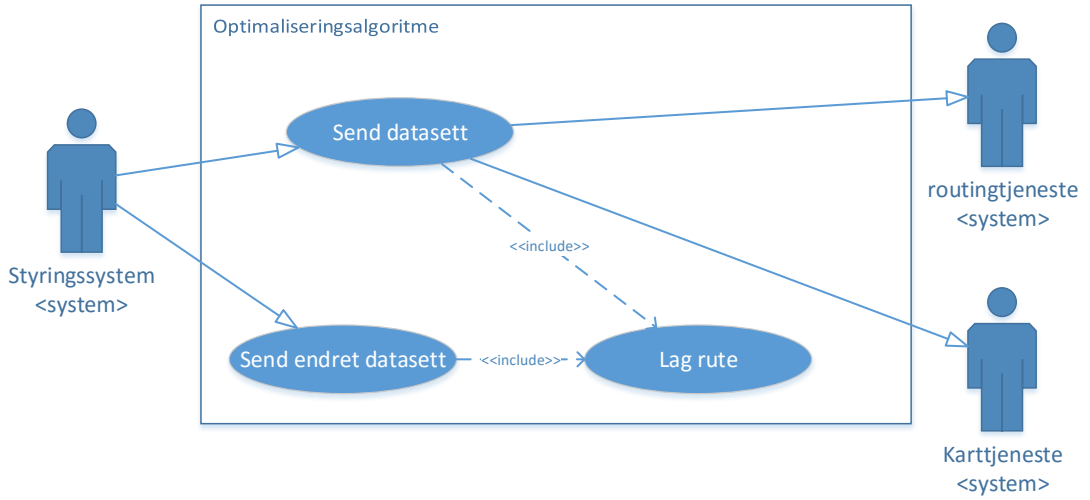


FIGUR 3 : OVERSIKT OVER SYSTEMENE OG DATAFLYTEN MELLOM DEM.

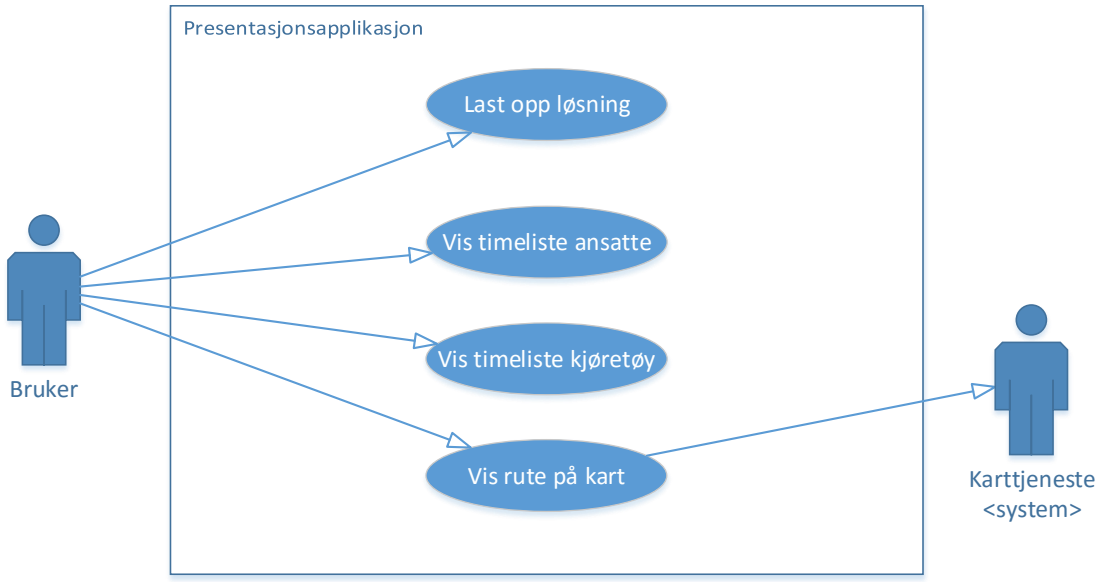
I denne oppgaven skulle gruppen utvikle selve algoritmen og en presentasjonsapplikasjon som kunne laste en løsning fra algoritmen og vise denne løsningen i timelister per ansatt og per kjøretøy, samt vise rutene på kart. Opprinnelig skulle dette programmet kunne laste et uløst problem og sende dette til algoritmen, for deretter å motta et optimalisert resultat tilbake. Dette programmet ville omgå styringssystemet og koble seg direkte til algoritmen, da det ligger utenfor oppgavens rammer å utvikle styringssystemet. Det ble imidlertid innført en avgrensning som førte til at denne kommunikasjonen mellom programmene ikke ble implementert. Vedlegg A – Utdrag fra kravspek viser de tenkte funksjonelle kravene for denne tidligere revisjonen av presentasjonsprogrammet.

3.2 Use-Case diagram

Figur 4 og figur 5 viser use-case diagrammene for henholdsvis algoritmen og presentasjonsapplikasjonen.



FIGUR 4: USE-CASE DIAGRAM FOR ALGORITME



FIGUR 5: USE-CASE DIAGRAM FOR PRESENTASJONSAPPLIKASJON.

3.3 High-Level Use-Case beskrivelser

Tabell 2 viser High-Level use-case definert for algoritmen. Tabell 3 viser High-Level use-case beskrivelser definert for presentasjonsapplikasjonen.

TABELL 2: HIGH LEVEL USE CASE BESKRIVELSER FOR ALGORITMEN

Use Case	*Send datasett*
Aktør	Styringssystem, karttjeneste, routingtjeneste
Mål	Opprette et datasett som algoritmen kan jobbe med.
Beskrivelse	Et styringssystem skal kunne sende inn data på et spesifisert format til algoritmen. Datasettet omformes så til passende datastrukturer og kart- og routingtjenester benyttes til å beregne avstander og reisetid mellom oppdrag og utkjøringssteder.

Use Case	*Send endret datasett*
Aktør	Styringssystem
Mål	Sende et endret datasett til algoritmen.
Beskrivelse	Styringssystemet skal kunne sende inn en allerede optimalisert rute, med noen endringer som må tas hensyn til.

Use Case	*Lag rute*
Aktør	Styringssystem
Mål	Lage et optimalisert forslag for en tidsplan for bedriftens utkjørere.
Beskrivelse	Ved hjelp av et datasett fra styringssystemet skal algoritmen finne tilnærmet optimale ruter for ett definert antall ansatte, biler og oppdrag. For å utføre optimaliseringen tar algoritmen i bruk en kart- og routingtjeneste. Dataen returneres så til styringssystemet.

TABELL 3: HIGH LEVEL USE CASE BESKRIVELSER FOR PRESENTASJONSAPPLIKASJONEN.

Use Case	*Last opp løsning*
Aktør	Bruker
Mål	Laste opp en løsning som deretter kan vises til brukeren.
Beskrivelse	Brukeren skal kunne velge et løsningsdatasett generert av algoritmen og få denne verifisert for formateringsfeil og lastet opp til applikasjonen for visning.

Use Case	*Vis timeliste ansatte*
Aktør	Bruker
Mål	Vis en oversikt over timelister for ansatte.
Beskrivelse	En bruker skal kunne velge å se timelister per ansatt, ved først å velge avdeling og deretter velge blant en liste av ansatte registrert ved avdelingen. Deretter skal timelisten for den valgte ansatte vises til brukeren.

Use Case	*Vis timeliste kjøretøy*
Aktør	Bruker
Mål	Vis en oversikt over timelister for kjøretøy.
Beskrivelse	En bruker skal kunne velge å se timelister per kjøretøy, ved først å velge avdeling og deretter velge blant en liste av kjøretøy registrert ved avdelingen. Deretter skal timelisten for det valgte kjøretøyet vises til brukeren.

Use Case	*Vis rute på kart*
Aktør	Bruker, karttjeneste
Mål	Vis et kjøretøys rute på kart.
Beskrivelse	En bruker skal kunne velge å se ruter på kart, ved først å velge avdeling og deretter velge blant en liste av kjøretøy registrert ved avdelingen. Deretter skal ruten for dette kjøretøyet tegnes opp på kart.

3.4 Extended Use-Case beskrivelser

Tabell 4 viser extended use-case beskrivelser for algoritmen. Kun de mer komplekse use-casene er utarbeidet som extended use-caser. Tabell 5 viser extended use-case beskrivelser for presentasjonsapplikasjonen. Også her er kun de mest komplekse use-casene utarbeidet.

TABELL 4: EXTENDED USE CASE BESKRIVELSE FOR ALGORITMEN.

Use Case	*Send datasett*
Aktør	Styringssystem, karttjeneste, routingtjeneste
Mål	Opprette et datasett som algoritmen kan jobbe med.
Prebetingelser	Ingen.
Postbetingelser	Melding er sendt tilbake til styringssystemet.
Detaljert hendelsesforløp	Data sendes av et styringssystem til algoritmen i JSON-format på et fastsatt spesifisert format. Datasettet skal bestå av oppdrag som skal utføres med spesifikasjoner for oppdraget, data om ansatte som er på jobb, data om biler som er tilgjengelig for utkjøring og data om hvilke begrensninger som algoritmen skal ta hensyn til. En begrensning kan foreksempel være om kjøretøyene skal ha en kapasitet knyttet til seg og om dette skal tas med i beregningene. Styringssystemet sender også inn en vekting av hvilke parametere som er viktige. Eksempler inkluderer redusere antall kilometer kjørt eller redusere ventetid for ansatte mellom oppdrag. Systemet modellerer dataen i passende strukturer som algoritmen kan benytte seg av og sender en 'status ok'-tilbakemelding til styringssystemet.
Alternative scenarier	**1.2** Feil i formatering på dataen. Dette inkluderer at forventet data ikke finnes i filen, at dataene er av feil type, eller at dataen har en verdi som er utenfor grensene av hva som forventes. Når systemet ikke kjenner igjen formateringen, sendes en feilmelding tilbake med informasjon om hvor det gikk galt.

Use Case	*Lag rute*
Aktør	Styringssystem
Mål	Lage et optimalisert forslag for en tidsplan for bedriftens utkjørere.
Prebetingelser	'Send datasett' har blitt gjennomført uten feil.
Postbetingelser	Data eller melding er sendt tilbake til styringssystemet.
Detaljert hendelsesforløp	Algoritmen finner tilnærmet optimale ruter basert på oppgitt datasett. Når den optimaliserte ruten er funnet, sendes dataen tilbake til styringssystemet i JSON-format.
Alternative scenarier	**1.1** Det er umulig å oppfylle betingelser. Dersom algoritmen finner ut at ett eller flere av kravene definert i datasettet ikke kan oppfylles, slik som for eksempel en leveranse som ikke kan nås innen tidsfristen, markeres bruddet og oppdraget beregnes som om betingelsen ikke eksisterer.

TABELL 5: EXTENDED USE CASE BESKRIVELSE FOR PRESENTASJONSAPPLIKASJONEN.

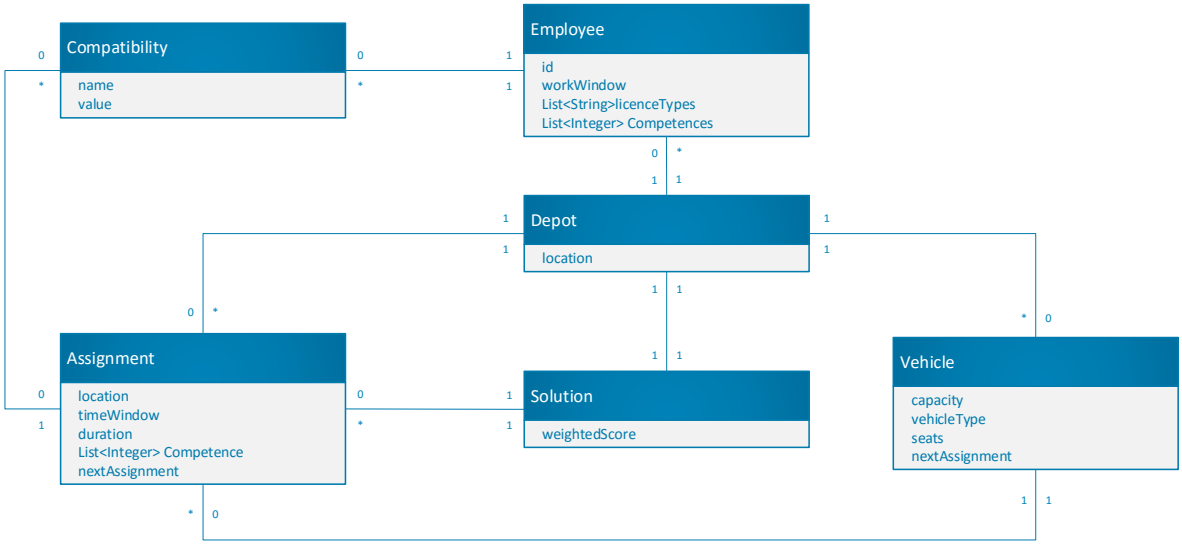
Use Case	*Last opp løsning*
Aktør	Bruker
Mål	Laste opp en løsning som deretter kan vises til brukeren.
Prebetingelser	Ingen.
Postbetingelser	En løsning er importert til applikasjonen.
Detaljert hendelsesforløp	1. Brukeren velger 'last opp en løsning'. 2. Systemet åpner en filutforsker for brukeren. 3. Brukeren velger filen som inneholder en generert løsning fra algoritmen. 4. Systemet verifiserer at filen har korrekt format, og innhold og importerer løsningen. En tilbakemelding vises til brukeren om at løsningen er importert.
Alternative scenarier	4.1 Filen følger ikke formateringsstandarder: Systemet avbryter og viser en tilbakemelding til brukeren om at filens format er gal. 4.2 Filen inneholder uforventede verdier: Systemet avbryter og viser en tilbakemelding til brukeren om at filen har uforventede verdier.

Use Case	*Vis rute på kart*
Aktør	Bruker, karttjeneste
Mål	Vis et kjøretøys rute på kart.
Prebetingelser	En løsning må være importert til systemet.
Postbetingelser	Ingen.
Detaljert hendelsesforløp	1. Brukeren velger 'Vis rute på kart'. 2. Systemet viser en liste over avdelingene i den importerte løsningen. 3. Brukeren velger en avdeling. 4. Systemet viser en liste av kjøretøy registrert på den valgte avdelingen. 5. Brukeren velger et kjøretøy. 6. Systemet går igjennom alle oppdrag som er tilegnet det valgte kjøretøyet, og benytter ankomsttid for å tegne dem som ruter på kartet.
Alternative scenarier	6.1 Systemet får ikke kontakt med karttjenesten: Kartet tegnes ikke opp og ruter vises ikke.

3.5 Detaljert kravspesifikasjon

Domenemodell

Algoritmen vil fungere på flere nivåer. Først vil den motta ett datasett med informasjon. Algoritmen vil så begynne med å fordele oppdrag til de forskjellige depotene basert på depotene og oppdragenes beliggenhet. Neste skritt er å fordele oppdrag inn i ruter basert på tilgjengelige ansatte og biler etter krav bestemt i datasettet. Siste skritt blir så å optimalisere rutene ved å endre på hvilke oppdrag som besøkes av hvilke ansatte og i hvilken rekkefølge disse besøkene foregår. Figur 6 viser en enkel domenemodell over systemet som utgjør algoritmen. Under følger en kort beskrivelse av de ulike komponentene.



FIGUR 6 DOMENEMODELL AV ALGORITMEN

Bedriftens utkjøringspunkter representeres av Depot-objekter. Hvert Depot-objekt inneholder informasjon om dens beliggenhet og hvilke biler og ansatte som er tilknyttet lokasjonen. Første skritt i algoritmen vil være å fordele oppdrag mellom de forskjellige utkjøringspunktene.

Ethvert kjøretøy som benyttes av bedriften til utkjøringer er representert av Vehicle-objekter. Et kjøretøy vil ha førerkortkrav og antall seter representert som felter i sin klassedefinisjon. Antall seteplasser er for eksempel avgjørende dersom flere ansatte skal kjøre samme bil. Denne verdien blir da en øvre grense for antall ansatte som kan delta. I Vehicle-objektet vil det også ligge en referanse til et oppdrag som så vil danne en kjede av oppdrag som denne bilen skal besøke. Opprettelse og optimalisering av denne kjeden vil være andre skritt i algoritmen.

Solution-klassen vil inneholde en mulig samling av ruter som dekker oppdrag tildelt et depot. Hver Solution-objekt vil tildeles en poengsum avhengig av kriterier som er definert før algoritmen kjøres. Denne poengsummen vil regnes ut på ulikt vis avhengig av hvordan bedriften som kjører algoritmen vekter kriteriene.

Assignment-objekter representerer oppdragene bedriften skal utføre. Disse inneholder informasjon om oppdragets beliggenhet og kompatibilitetskrav. Et Assignment-objekt vil også inneholde eventuelle tidsvinduer for når oppdraget skal utføres, hvor lang tid det vil ta og hvilke kompetanse som kreves av de ansatte for å utføre oppdraget. Oppdragene vil også inneholde en referanse til neste oppdrag i en kjede av oppdrag som starter i en bil.

Compatability klassen består av en personlig egenskap og en tilhørende karakter. For hjelpetjenesten kunne denne egenskapen for eksempel vært pleieevne, eller evne til å håndtere stress.

Integerlistene av competence som ligger i oppdrag og ansatte definerer hvilke kompetansekrav som skal til for å utføre en bestemt arbeidsoppgave, for eksempel kan kun sykepleiere sette sprøyter på pasienter hos hjemmesykepleien. Kompetansekrav settes derfor på oppdrag og sammenlignes med ansattes kompetanse. Fordi kompetansekrav ikke nødvendigvis er gradert, kan en ansatt ha flere kompetanser knyttet til seg. Et eksempel på et slikt ugradert hierarki kan finnes hos ansatte i et byggfirma. En snekker og en elektriker vil ha ulik kompetanse, men kan også ha overlappende oppgaver.

Employee-objekter representerer en ansatt hos bedriften. En ansatt jobber fra et depot og har en id, et tidsrom der den ansatte er tilgjengelig og data som forteller hvilke kjørekortklasser den ansatte er registrert med. Employee-objekter har også en referanse til en bil som den ansatte kjører i perioden algoritmen optimaliseres for. En del av andre fase er å finne den beste kombinasjonen av hvilke ansatte som skal være tildelt hvilke kjøretøy.

Harde og myke føringer

Kriteriene som algoritmen mottar kan deles inn i to kategorier. Gruppen har valgt å kalle disse for harde og myke føringer. Harde føringer defineres som kriterier der løsningen fra algoritmen ikke kan brukes hvis disse brytes fordi det vil føre til uakseptable løsninger. Et eksempel på en slik løsninger kan være at ansatte settes til å gjøre oppgaver de ikke har den nødvendige kompetansen til å utføre. De myke føringene er de kriteriene hvor algoritmen må gjøre en avveining av hvordan den skal fordele oppdragene og hva den skal optimalisere løsningen etter. Dette må gjøres ved at bedriften definerer hvor viktig de forskjellige kriteriene er i forhold til hverandre. Ut ifra dette vil algoritmen kunne tilegne løsningsforslagene en poengsum som den vil forsøke å minimere. Det skal legges til mulighet for å utvide algoritmen med flere kriterier etter hvert.

Datasett

Input til algoritmen

Datasettet er utarbeidet av gruppen og består av tre hovedobjekter - meta-data, oppdragsdata og depotdata. Meta-data objektet inneholder bedriftsspesifikke data og informasjon om hvordan algoritmen skal løse optimaliseringsproblemet. Oppdragsdataen inneholder informasjon om alle oppdrag som skal utføres i det aktuelle tidsrommet og depotdata inneholder data om alle depoter og deres tilgjengelige ressurser. Figur 7 viser et eksempel på hvordan et slikt datasett kan se ut.

FIGUR 7: : EKSEMPEL PÅ DATASETT SOM MOTTAS AV ALGORITMEN I JSON-FORMAT.

Meta-dataene brukes av algoritmen til å bygge opp problemet som skal løses. Her defineres hvilket tidsrom algoritmen skal optimalisere innenfor, hvordan ulike faktorer skal prioriteres under utregningene og hvilke kompatibilitets- og kompetansegrader som skal tas hensyn til under optimaliseringen. Tidsrommet algoritmen skal optimalisere innenfor er definert som perioden mellom to angitte datoer. Faktorene prioriteres etter vekter som heltall fra 0 og oppover og angir hvor mye de ulike kriteriene vil påvirke poengberegninger. Gruppen vil selv danne basisvekting for datasettet. Basisvektingen oppnås ved å sette alle vektene lik én. I figur 7 er for eksempel vekten for jevn fordeling satt til to. Dette innebærer at føringer relatert til jevn fordeling, påvirker poengsummen dobbelt så mye som ved bruk av basisvektingen.

Oppdragsdataen er samlet som en egen post i datasettet. Dette er gjort fordi disse oppdragene ikke skal være tildelt depoter før de ankommer systemet. Det er algoritmens oppgave å vurdere hvilke oppdrag som skal utføres av de ulike avdelingene. Hvert oppdrag er tildelt en id og inneholder data om dets koordinater i bredde- og lengdegrader, hvilket tidsvindu oppdraget må utføres i, en boolsk verdi som tilsier om tidsvinduet er obligatorisk, eller kun ønskelig å følge, oppdragets antatte varighet og antall ansatte som kreves for at oppdraget kan utføres. En liste over hvilke kompatibilitetsegenskaper som bør være tilstede og hvilke kompetanser som skal være tilstede hos den eller de ansatte som utfører oppdraget er også listet her.

Depotdataen inneholder informasjon om alle depoter. Til hvert depot er det tilknyttet koordinatdata på samme måte som for hvert oppdrag. I tillegg finnes data om alle ansatte og kjøretøy tilknyttet

depotet. Hver ansatt har en id, samling av tilegnede kjøresertifikat-typer og liste over arbeidsskift de er tilgjengelige i løpet av den tiden algoritmen skal optimalisere for. I tillegg har hver ansatt lister med hvilke kompetanser, og kompatibiliteter de er registrert med. Hvert kjøretøy har informasjon om antall seter kjøretøyet er utstyrt med og hvilket type førerkort som trengs for å manøvrere kjøretøyet.

Det ble gjort en vurdering om ikke kjøretøydata også burde skilles ut fra depotet og fordeles av algoritmen på samme måte som som oppdragsdataene. Gruppen kom frem til at algoritmen da hadde flyttet for mye på kjøretøyene mellom avdelinger ved hver kjøring av algoritmen og at dette ikke hadde vært hensiktsmessig ved daglig bruk.

Output fra algoritmen

Algoritmen vil sende ett datasett tilbake til styringssystemet i ett format som vist i figur 8.


```
"depots": [
  {
    "id": 1000,
    "location": {
      "latitude": 60.770855,
      "longitude": 10.673871
    }
  }
],
"vehicles": [
  {
    "id": 1237,
    "depot": 1000,
    "seats": 4,
    "licenceNeeded": "B"
  }
],
"employees": [
  {
    "id": 230,
    "licenceTypes": [
      "B",
      "C"
    ],
    "workShifts": [
      {
        "dateTimeFrom": "02.03.2016 08.00",
        "dateTimeTo": "03.03.2016 18.00"
      }
    ],
    "compatibilities": [
      {
        "id": 1,
        "grade": 2
      },
      {
        "id": 2,
        "grade": 2
      },
      {
        "id": 3,
        "grade": 3
      },
      {
        "id": 4,
        "grade": 2
      },
      {
        "id": 5,
        "grade": 4
      }
    ],
    "competences": [
      1,
      2,
      3
    ],
      "depot": 1000,
      "vehicle": 1237
    }
  ],
  "assignments": [
    {
      "id": 2,
      "location": {
        "latitude": 60.781525064205255,
        "longitude": 10.67523075947595
      },
      "timeWindow": {
        "hardRequirement": false,
        "window": {
          "dateTimeFrom": "02.03.2016 08.27",
          "dateTimeTo": "02.03.2016 14.27"
        }
      },
      "duration": 48,
      "employeesNeeded": 1,
      "competenceNeeded": [
        2,
        3
      ],
      "compatibilityRequired": [
        {
          "id": 1,
          "grade": 2
        },
        {
          "id": 2,
          "grade": 4
        },
        {
          "id": 3,
          "grade": 1
        },
        {
          "id": 4,
          "grade": 1
        },
        {
          "id": 5,
          "grade": 3
        }
      ],
      "arrivalTime": "02.03.2016 13.13",
      "depotVisitAfter": false,
      "vehicle": 1237
    }
  ]
}
```


FIGUR 8: EKSEMPEL PÅ DATA SOM RETURNERES FRA ALGORITMEN I JSON-FORMAT.

Algoritmen returnerer løsningen tilbake til styringssystemet når dens kalkulasjoner er fullført. I datasettet vil depotene, kjøretøyene, de ansatte og oppdragene samles i separate samlinger. Ansatte og kjøretøy inneholder derfor et ekstra datafelt som representerer hvilket depot de er registrert på. Alle ansatte og oppdrag vil også ha et felt som tilsier hvilket kjøretøy den ansatte eller det oppdraget er tildelt. Oppdragene har i tillegg fått to nye felter: ankomsttiden for når kjøretøyet dette oppdraget er tildelt skal utføre oppdraget og en boolsk verdi som tilsier om kjøretøyet skal tilbake til depotet før utføringen av det neste oppdraget. Hvert depotobjekt vil også tildeles en todelt poengsum som tilsier hvor sterk løsningen for det depotet er. Den todelte poengsummen samsvarer med de harde og

myke føringene beskrevet tidligere. Dersom poengsummens harde del ikke er lik tallet null, fant ikke algoritmen en løsning som ikke bryter med føringene innad i dette depotet. Den myke poengsummen synker for eksempel dersom kjøretøyene i depotet kjører mer eller kommer for sent til tidsvinduer som ikke er obligatoriske. Hvert kjøretøy har også et eget felt som representerer tiden i sekunder kjøretøyet blir benyttet til kjøring.

Merk at algoritmen kun opererer med data som er nødvendig for å utføre optimaliseringen. Det er systemene som benytter seg av algoritmen sin oppgave å tilknytte den genererte dataen tilbake til relevante datastrukturer brukt i deres systemer. For eksempel benyttes kun en id for de ansatte og ikke annen data som navn og hjemadresse.

3.6 Ikke-funksjonelle krav

De ikke-funksjonelle kravene beskrevet her vil være gjeldende kun for algoritmen. Presentasjonsapplikasjonen som skal utvikles vil det ikke settes spesielle krav til foruten de funksjonelle kravene definert ovenfor. Dette er gjort da denne applikasjonen kun utvikles for eget bruk og ikke skal benyttes av andre enn gruppen selv for å teste algoritmen.

Krav til dokumentasjon

Da algoritmen skal overtas av ETC etter at prosjektet er gjennomført er det viktig å ha god dokumentasjon av løsningen. Det settes derfor krav til å utvikle plandokumenter som kravspesifikasjon og designdokument som skal overrekkes oppdragsgiver. Gruppen vil også fokusere på å produsere visuelle diagrammer for å gjøre systemet mer forståelig. Diagrammene inkluderer blant annet domenemodell, use-case diagram, klasse-diagram og diagram over arkitekturmodell. Det settes også krav til at kode skal kommenteres med det formål å gjøre koden så forståelig som mulig ved overtagelse av systemet. Alle klasser, funksjoner og spesielt kompleks kode skal kommenteres.

Sikkerhet og personvern

Algoritmen vil håndtere mye sensitiv data, men all kommunikasjon mellom styringssystemet og algoritmen vil foregå innenfor en sikker sone. Denne kommunikasjonen ansees som sikker og trenger ikke krypteres. Styringssystemet, som faller utenfor oppgavedefinisjonen, har ansvar for å håndtere sikker kommunikasjon utad. Algoritmen og styringssystemet skal ikke lagre noe data som mottas fra eksterne systemer og tar som utgangspunkt at bruker sørger for nødvendig godkjennelse av behandling av personvernsopplysninger i samsvar med lov om behandling av personopplysninger (Personopplysningsloven, 2000)

Etiske hensyn

I de tilfeller der det er hensiktsmessig vil algoritmen forsøke å fordele arbeidsoppgavene basert på blant annet kompatibilitet. Et eksempel på områder hvor dette vil være hensiktsmessig er hjemmesykepleien, her er det svært viktig at vanskelige pasienter håndteres av kompetente pleiere. Kompatibilitetslisten vil måtte genereres til de ansatte, men dette vil gjøres i et eksternt system. Algoritmen vil kun motta og behandle denne informasjonen. De etiske hensynene rundt innhenting av informasjonen blir derfor utenfor prosjektets problemområde.

Lovgitte krav

Dersom algoritmen foreslår løsninger som bryter med nasjonale lover og derfor ikke reelt kan brukes må dette fremgå klart i løsningen. Eksempler på slike lovgitte føringer er:

- Ansattes kompetanse under fordelingen av oppdrag.
- Fartsgrenser.
- Førerkortklasser.

4 Teori og valg av teknologi

4.1 Introduksjon til Constraint satisfaction problemer

Hva er et Constraint Satisfaction problem?

Et constraint satisfaction problem, eller CSP er et matematisk problem definert som et sett med variabler der variablenes tilstand må overholde fastsatte føringer. Hver variabel har et definert domene med verdier variabelen kan holde (Poole & Macworth, 2010). Sudoku er et velkjent problem og et godt eksempel på et CSP. Hver celle på et 9x9 sudokubrett kan ha verdiene 1 til 9. Cellene er variablene i problemet, mens deres domene er heltallene 1 til 9. Reglene i sudoku blir føringene. Ingen tall kan forekomme to ganger i en rad, kolonne eller boks.

De fleste slike problemer er NP-Complete eller NP-hard. En grov definisjon kan gis ved at det ikke finnes en bevist fremgangsmåte som garanterer å finne den optimale løsningen på problemet i polynomisk tid (Eppstein, 1996). Det er imidlertid mulig å verifisere en gitt løsning på problemet i polynomisk tid. Dette gjør blant annet at "Brute Force"-algoritmer som systematisk beveger seg gjennom alle mulige løsninger vil bruke for lang tid.

En typisk brute force implementasjon for suduko vil forsøke å fylle inn hver celle med tallene fra en til ni, ved å finne det laveste tallet som ikke bryter noen regler, å fylle dette inn, for så å forflytte seg til neste celle. Dersom ingen tall kan plasseres uten at regler brytes, forflytter algoritmen seg tilbake til forrige celle og forsøker et høyere mulig tall. Slik fortsetter det til en gyldig løsning er funnet. Ved løsning av et 9x9 sudokuproblem vil brute force gi raske svar. Øker man derimot størrelsen på brettet vil løsningstiden øke dramatisk. Dette skjer fordi antall mulige løsninger har en sterk korrelasjon med tiden det tar å løse problemet.

Løsningen og dens styrke

En løsning på et CSP er en tilstand for problemet der alle variabler er blitt initialisiert med verdier i sitt domene. Man deler ofte løsningene opp i kategorier etter hvor godt begrensningene overholdes i den gjeldende løsningen:

- Possible: Alle mulige løsninger uavhenging av om føringene overholdes eller brytes.
- Feasible: Alle løsninger der føringene overholdes.
- Optimal: Den eller de løsningene som anses som mest optimale blant løsningene definert som feasible.
- Best: Den nåværende beste løsningen som er funnet, blant løsningene definert som feasible.

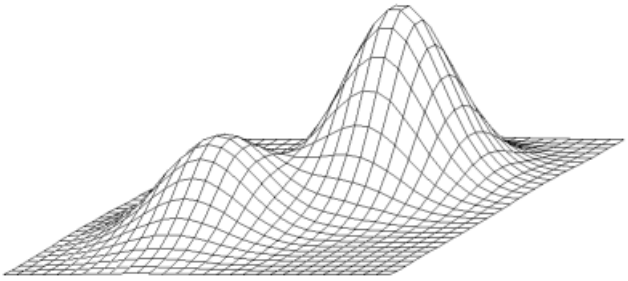
I sudoku og andre klassiske CSPer gir det lite mening å skille mellom feasible og optimal i kategoriseringen av løsninger. Har du funnet en løsning som ikke bryter med føringene har du også funnet den optimale løsningen. Weighted Constraint Satisfaction problemer, eller WCSP definerer myke føringer ved å introdusere kostfuksjoner der målet er å finne løsningen som minimerer kostsummen fra alle føringene (Terrence, 2011). Denne ideen kan også utvides ved å innføre flere lag med føringer. For eksempel kunne et lag inneholde føringer som skal opprettholdes og et annet lag føringer som kan brytes. Dersom ruten for et avisbud skulle optimaliseres og føringen i laget som må opprettholdes var at en spesifikk kunde må besøkes i løpet av den første halve timen, vil man potensielt ha mange løsninger som tilfredsstiller dette kravet. Hvis man i tillegg introduserer et lag med føringer med en kostfunksjon basert på distansen avisbudet forflytter seg i løpet av ruten vil

man kunne tilegne hver løsning en poengsum for å vurdere løsningens styrke. Alle løsninger som får avisene levert i løpet av to timer vil være definert som feasible. Løsningen definert som best vil være den løsningen algoritmen hittil har funnet der avisbudet beveger seg den korteste distansen. Denne løsningen vil også ha den laveste poengsummen blant løsningene vurdert til nå. Løsningen definert som optimal vil være den eller de løsningene med den totalt laveste poengsummen blant løsningene i hele løsningsmengden.

Bevegelse gjennom løsningsmengden

Brute force implementasjoner viser at å gå igjennom hver eneste løsning vil være lite hensiktsmessig. Ofte benyttes isteden en kombinasjon av heuristiske og metaheuristiske søkemetoder for å oppnå gode resultater. En heuristisk søkemetode benytter teknikker for å løse et problem raskere ved å gjøre tilnærminger og oppnå en god løsning uten å kunne bevise hvor god løsningen egentlig er (Marshall, 1996). Løsninger funnet gjennom heuristiske søkemetoder vil kunne fungere som utgangspunkt for utvidede søk etter enda bedre løsninger. Mange metaheuristiske søkemetoder krever nemlig en allerede konstruert løsning som kan bygges videre på og forbedres. Metaheuristiske søkemetoder har som mål å effektivt utforske større deler av løsningsmengden. Slike søkemetoder er ikke problem-spesifikke. Fremgangsmåten for søkemetodene varierer mellom de ulike variantene som finnes. Tre eksempler på lokale metaheuristiske søkemetoder er beskrevet nedenfor.

Hill Climbing baserer seg på å se på den nåværende løsningens nabolag. Søkemetoden velger den første av disse løsningene som er mest optimal og vurderer deretter nabolaget til denne nye løsningen. Slik fortsetter det til ingen av de nærliggende løsningene gir bedrede resultater. (Luke, 2015, s. 17) Søkemetoden finner et lokalt topp- eller bunnpunkt i løsningsmengden, også kalt henholdsvis lokale maksima og lokale minima. I figur 9 vises en flate med to lokale maksima. Hill Climbing vil finne én av disse lokale maksima-punktene men kan ikke garantere at dette er det optimale resultatet.



FIGUR 9: FLATE MED TO LOKALE MAKSIMA. (PUBLIC DOMAIN).

Tabu Search baserer seg på Hill Climbing men inneholder mekanismer som gjør det mulig å unnslippe lokal minima eller lokal optima (Luke, 2015, s. 26). Dette gjøres ved å gi søkemetoden mulighet til å velge et trekk som gir en mindre gunstig løsning dersom løsningen befinner seg i et lokal minima eller lokal maksima. I tillegg opprettes en tabu-liste med en definert størrelse. For hvert trekk som blir utført plasseres et objekt relatert til trekket i tabu-listen. Objektet kan for eksempel være trekket selv, eller objektet som ble modifisert under trekket. Dette avhenger av implementasjonen av søkemetoden. Tabu-listen inneholder altså objekter som er svartlistet relatert til tidligere valgte trekk. Nye trekk knyttet til et objekt i tabu-listen blir ikke akseptert. Dette oppfordrer søkemetoden til å bevege seg vekk fra tidligere besøke løsningsområder. På grunn av dette klassifiseres Tabu Search som en blanding av de lokale- og globale søkemetodene.

Simulated Annealing ser kun på én eller et fåtall trekk i hver iterasjon. Et trekk velges dersom det ikke fører til en mindre optimal løsning, eller dersom det fører til en mindre optimal løsning og trekket består en randomisert sjekk (Luke, 2015, s. 25). Sjansen for å bestå denne sjekken reduseres ettersom kjøretiden for søkemetoden øker, og avhengig av hvor mye dårligere den nye løsningen er. Faktoren som reduserer sjansen for å bestå denne sjekken over tid kalles temperaturen, der en høy temperaturverdi tillater oftere trekk som gir mindre optimale løsninger. I praksis vil temperaturen medføre at Simulated Annealing først ser på et bredt spekter av løsningsområdet mens temperaturen er høy og deretter vurderer lokale løsningsområder når temperaturen synker. Dette gjør at søkemetoden både klassifiseres som en blanding av de lokale- og globale søkemetodene i likhet med Tabu Search.

Trekk og nabolag

Alle endringer av verdier på variabler, som transformerer en løsning til en annen løsning er definert som et trekk. Et trekk er problemspesifikt og man velger selv hvordan et trekk implementeres. For eksempel kan et trekk være en enkel endring av verdien til én variabel, endring av verdien til to variabler samtidig, eller at verdien for to variabler byttes om. Det som er avgjørende er at trekket er klart definert på forhånd for algoritmene som benytter seg av dem. Alle løsninger som kan nås ved å utføre disse definerte trekkene på en løsning er definert som den løsningens nabolag. En løsnings nabolag er altså løsninger som er direkte nærliggende den originale løsningen. Lokale søkemetoder benytter seg typisk av trekk for å bevege seg gjennom løsningsmengden og finne frem til bedre løsninger.

Den optimale løsning

Når løsningsmengden øker blir det usannsynlig å finne de optimale løsningene for et CSP i løpet av en akseptabel tidsperiode. Målet for slike problemer blir derfor ikke å finne den optimale løsningen, men en løsning som befinner seg på et akseptabelt lokalt optima eller lokalt minima.

4.2 Bruk av optimaliseringsrammeverk

Under arbeidet med designdokumentet ble det tydelig for gruppen at oppgavens kompleksitet og størrelse var av en slik grad at det ikke ville være tid til å implementere all ønsket funksjonalitet dersom algoritmen skulle utvikles fra bunnen av. I arbeidet med å lære seg forskjellige metoder for å løse optimaliseringsproblemer kom gruppen dessuten stadig over ett omfattende optimaliseringsrammeverk som så ut til å kunne benyttes for å løse gruppens problemstilling svært godt kalt Optaplanner. Gruppen så derfor på to mulige alternativer: snevre inn oppgaven og utvikle en algoritme som løser en mindre og mer spesifikk del av problemet, eller ta i bruk ett slikt rammeverk og fortsette prosjektet uten avgrensninger. Gruppen diskuterte derfor muligheten for å benytte ett slikt rammeverk for å løse oppgaven med både veileder og oppdragsgiver. Veileder så ingen problemer med å skifte fokuset på oppgaven fra å utvikle en algoritme helt fra bunnen til å benytte ett slikt rammeverk. Valget av utviklingsmodell og planleggingen av en lang forprosjektperiode viste seg også å være hensiktsmessig da gruppen enda ikke hadde startet på implementering. Et slik skifte i oppgaven var derfor enklere å gjennomføre. Oppdragsgiver stilte seg også svært positiv til å ta i bruk ett slikt rammeverk fremfor å snevre inn oppgaven.

Da gruppen så dypere på muligheten for benytte et optimaliseringsrammeverk identifiserte de spesielt fire kriterier som måtte oppfylles av rammeverket. Rammeverket måtte kunne tilpasses i slik grad at problemet kunne bli representert på en god måte, det måtte muliggjøre enkel utvidelse av

føringer i fremtiden, det måtte være godt dokumentert for å gjøre det så lett som mulig for gruppen å sette seg inn i det og det var spesielt viktig at det benyttet seg av en lisens som muliggjorde kommersiell bruk med lukket kildekode. Optaplanner tilfredsstilte alle disse kravene. Gruppen så allikevel på andre alternativer enn Optaplanner, men fant at de fleste var designet for spesifikke problemområder, inneholdt for lite frihet til å modellere problemet, eller benyttet seg av proprietære lisenser som for eksempel GPL. Ingen av alternativene ga derfor ett like godt inntrykk som Optaplanner.

Optaplanner er et optimaliseringsrammeverk for bedrifters ressursplanlegging (Optaplanner, 2016). Rammeverket spesialiserer seg på løsning av CSPer og benytter seg av en Apache 2.0 lisens som tillater kommersiell bruk. Optaplanner gir stor frihet til modellering av problemet da det benytter seg av vanlige java-klasser. Føringene for problemet separeres fra resten av koden og gjør det enkelt å utvide med flere ved senere anledninger. Optaplanner er gratis å benytte, men har betalte løsninger for bedrifter som ønsker ekstra veiledning under utvikling.

4.3 OptaPlanner

Optaplanner definerer terminologi som benyttes i det videre designet av algoritmen. Her følger en forklaring av de ulike begrepene og konseptene som vil gjøre seg gjeldende knyttet opp til et tenkt sudokuproblem. Problemet er designet av gruppen og ment for å illustrere konseptene brukt i Optaplanner. Det bør derfor ikke vurderes som en reell løsning på et slikt problem.

Planvariabler og planentiteter

I Optaplanner er variablene i et CSP kalt Planning Variables, som gruppen har valgt å oversette til planvariabler. Dette er altså variablene som i løpet av optimaliseringen endrer sine verdier. Optaplanner introduserer også Planning Entities, kalt planentiteter av gruppen. Planentiteter er objekter som inneholder slike planvariabler (Optaplanner Team, 2016, s. 87). For at Optaplanner skal kunne vite hvilke objekter som inneholder planvariabler annoteres klassene til slike objekter med @PlanningEntity som vist i figur 10. Planvariablene annoteres også indirekte da Optaplanner krever at slike felter har en get-funksjon. Denne funksjonen annoteres med @PlanningVariable og en referanse til en verdileverandør som angir domenet med lovlige verdier for variabelen. Figuren viser en tenkt implementasjon for en sudokucelle, der planvariabelen er cellens verdi. Verdileverandøren ville i et 9x9 sudokubrett returnere en samling av integer-verdiene fra én til ni.

```
@PlanningEntity
public class Cell {
	private Integer value;

	private int row;
	private int col;
	private boolean fixed;


	@PlanningVariable(valueRangeProviderRefs = {"cellRange"})
	public Integer getValue() {
		return value;
	}
}
```

FIGUR 10: TENKT PLANENTITET FOR EN SUDOKU CELLE.

Løsningsplan

En løsningsplan kalles en Planning Solution i Optaplanner og definerer en mulig løsning på et CSP. I likhet med planentiteter annoteres løsningsplanklasser med @PlanningSolution. En løsning inneholder en samling av planentiteter som skal få sine verdier endret i løpet av optimaliseringen (Optaplanner Team, 2016, s. 107). Figur 11 viser en tenkt løsningsplan for sudokuproblemet. Her er det samlingen cells som er planentiteter. Dette annoteres på samlingens get-funksjon med @PlanningEntityCollection. Verdileverandører med domenene for planvariablene er også definert her. I figuren er funksjonen getPossibleCellValues annotert med @ValueRangeProvider og id-en cellRange som returnerer samlingen med verdier som planvariablene linket opp mot denne verdileverandøren kan ha.

Løsningsplanklassen må også implementere en poengsum. En løsnings tilegnede poengsum er et resultat av hvor sterk løsningen er og er styrt av føringene som er definert for problemet. En høyere poengsum tilsvarer en bedre løsning. I figuren er poengsummen av typen SimpleScore som vil si at poengsummen kun inneholder ett lag med føringer. Tilsvarende finnes HardSoftScore og HardMediumSoftScore for henholdsvis to og tre lag med føringer (Optaplanner Team, 2016, s. 134). Lagene vektes separat. For eksempel vil en HardSoftScore og poengsum 0 hard og -1000 soft regnes som en bedre løsning enn en poengsum med -1 hard og 0 soft.

```
@PlanningSolution
public class SudokuSolution implements Solution<SimpleScore>, Serializable {

	private List<Cell> cells;
	private List<Integer> possibleCellValues;
	private SimpleScore score;


	@PlanningEntityCollectionProperty
	public List<Cell> getCells() {
		return cells;
	}                                                    }

	@ValueRangeProvider(id = "cellRange")
	public List<Integer> getPossibleCellValues() {
		return possibleCellValues;
	}

}
```

FIGUR 11: TENKT LØSNINGSPLAN FOR ET SUDOKU PROBLEM.

Kalkulasjon av føringer

Utregningen av poengsummen for en løsning gjøres etter hvert trekk som utføres under optimaliseringen. Optaplanner gir støtte for tre mulige fremgangsmåter for å utføre denne utregningen (Optaplanner Team, 2016, s. 136):

- Enkel Java implementasjon.
- Inkrementell Java implementasjon.
- Poengberegning ved hjelp av Drools.

Enkle Java implementasjoner krever kun én klasse som benytter seg av interfacet EasyScoreCalculation, og som overskriver metoden calculateScore. Utvikleren skriver koden som evaluerer løsningen som ønsket og returnerer Score-objektet. Fremgangsmåten er lett å lære fordi

den kun krever kjennskap til Java, men skalerer dårlig da hele metoden kjøres for hvert trekk som utføres, uavhengig av om kun deler av beregningene endrer seg som følge av dette trekket.

Inkrementelle Java implementasjoner retter opp i denne mangelen ved å la utvikleren definere når beregninger som endrer poengsummen skal utføres. Denne fremgangsmåten er rask og skalerbar, men er mer kompleks å forstå og skrive. En klasse må arve fra AbstractIncrementalScoreCalculation og benytte interfacet IncrementalScoreCalculation med tilhørende metoder. Metodene definerer blant annet hva som skal skje før og etter en Planentitet legges til og før og etter en Planvariabel endres.

Poengberegning ved hjelp av Drools utfører inkrementell poengberegning bak kulissene og lar utvikleren definere føringene ved hjelp av separate regler. Separasjonen av regler gjør det også lett å legge inn nye føringer senere i utviklingen dersom brukerens krav endres. Reglene skrives imidlertid i en egen syntaks kalt Drools Rule Syntax (DRL) som kan være krevende å sette seg inn i.

Drools

Drools er et Business Rule Management System med en tilhørende Rule Engine (Red Hat, 2016). Som nevnt deles logikk inn i separate regler som har et tilhørende handlingsscenario som utføres dersom regelen inntreffer. Reglene er definert i egne DRL-filer.

```
@PlanningSolution
public class SudokuSolution implements Solution<SimpleScore>, Serializable {
	.
	.
	.

	@Override
	public Collection<? extends Object> getProblemFacts() {
		List<Object> facts = new ArrayList<>();
		return facts;
	}
}
```

FIGUR 12: DERSOM DROOLS BENYTTES MÅ AKTUELLE PROBLEMFAKTA LEGGES TIL SCOREDIRECTOR-EN.

Bruk av Drools krever at klassen som er annotert med @PlanningSolution implementerer get-funksjonen getProblemFacts. Funksjonen kalles av DroolsScoreDirector og returnerer en samling av objekter som settes inn i Drools Working Memory. Dette medfører at DRL-reglene får tilgang til objektene under poengberegningene (Optaplanner Team, 2016, s. 110). Planentitetene legges til automatisk og skal ikke returneres av denne funksjonen. Det er objekter fra domenemodellen som ikke endres under optimaliseringen, men som allikevel benyttes under poengkalkuleringen som skal legges til her. Slike objekter kalles i Optaplanner for Problem Facts og gruppen har valgt å kalle dem problemfakta. I figur 12 vises denne metoden i den tenkte klassen SudokuSolution fra tidligere. Fordi Planentitetene legges inn i Drools Working Memory automatisk, returneres kun en tom samling objekter.

```
rule "NoMultipleValuesRow"
	when
		Cell($row : row, $value : value)
		Cell($row == row, $value == value)
	then
		scoreHolder.addConstraintMatch(kcontext, -1);
end
```

FIGUR 13: TENKT REGEL SKREVET I DRL SOM IVARETAR AT CELLER I EN RAD IKKE HAR SAMME VERDI.

DRL-regler er inndelt i to deler – en betingelse og et handlingsscenario. Betingelsen må inntreffe for at handlingen skal eksekveres (JBoss Drools Team, 2016, s. 272). Figur 13 viser regelen "noMultipleValuesRow". Betingelsen er her definert slik: Handlingen utføres dersom det finnes en celle med en gitt rad og en gitt verdi og det finnes en annen celle med samme rad og tildelte verdi. Dette gjøres ved å binde verdiene row og value til midlertidige variabler. Disse variablene gjenkjennes ved at variabelnavnet begynner med et $-symbol. Handlingsdelen av regelen kan inneholde java-kode. Når Drools benyttes i sammenheng med Optaplanner brukes et scoreHolder-objekt som er knyttet opp til poengsummen i den gjeldende løsningsplanklassen. Det første parameteret i scoreHolderens metode addConstraintMatch er variabelen kContext. Parameteret må være tilstede og er Drools-spesifikt. Den brukes blant annet for å utføre inkrementell poengberegning. Det andre parameteret er påvirkningen betingelsen har på poengsummen i Solution-klassen. I eksempelet vil enhver forekomst av betingelsen føre til at poengsummen reduseres med én.

Solveren

For å utføre selve optimaliseringen kreves et Solver-objekt. Solver-objektet inneholder innstillinger for hvordan problemet skal løses (Optaplanner Team, 2016, s. 80). Dette inkluderer informasjon om hvilke planentitet- og løsningsplanklasser som benyttes, hvilke regler som skal benyttes under poengberegning, hvilke algoritmer som skal tas i bruk under optimaliseringen, innstillinger for når optimaliseringen skal terminere og hvilke trekk som er tilgjengelige for de ulike algoritmene. Disse innstillingene blir vanligvis definert i en xml-fil slik som i figur 14, men kan også lages programmatisk ved å opprette en ny instans av klassen SolverConfig og justere verdiene i denne. Denne klassen har definerte get- og set-metoder for alle verdier som kan defineres i xml-filen.

Solver-konfigurasjonen inneholder også minst én fase Solveren skal utføre under optimaliseringen. Fasene er kategorisert som enten exhaustive search, construction heuristic, eller local Search (Optaplanner Team, 2016, s. 160). Exhaustive search er faser som benytter seg av typiske brute force algoritmer der det gjøres uttømmende søk av løsningsmengden. Construction heuristics som definert i Optaplanner er heuristiske søkemetoder som finner et forslag til en løsning i løpet av en endelig periode med tid. Dette innebærer å gi planvariablene sine initialverdier. Local search faser benytter metaheuristiske algoritmer som Hill Climbing, Tabu Search eller Simulated Annealing. De metaheuristiske fasene er avhengig av at en construction heuristisk fase har satt verdiene til alle planvariabler før fasen kan begynne. Hver fase kan justeres med ulike konfigurasjoner. For eksempel kan man ved å benytte Tabu Search, konfigurere størrelsen på tabu-listen som benyttes under optimaliseringen.

Hver fase kan også definere aktuelle trekk for den fasen. Dette gjøres ved å inkludere en MoveSelector i fasen (Optaplanner Team, 2016, s. 177). Det finnes flere ulike typer MoveSelector-er. ChangeMoveSelector endrer for eksempel en planvariabels verdi til en annen verdi i dens domene, og SwapMoveSelector bytter om verdiene for planvariablene til to planentiteter som tilhører samme klasse.

```xml
<solver>
  <solutionClass>no.ntnu.sudoku.SudokuSolution</solutionClass>
  <entityClass>no.ntnu.sudoku.Cell</entityClass>

  <scoreDirectorFactory>
    <scoreDefinitionType>SIMPLE</scoreDefinitionType>
    <initializingScoreTrend>ONLY_DOWN</initializingScoreTrend>
  </scoreDirectorFactory>

  <termination>
    <secondsSpentLimit>30</secondsSpentLimit>
  </termination>

  <constructionHeuristic>
    <constructionHeuristicType>FIRST_FIT</constructionHeuristicType>
  </constructionHeuristic>

  <localSearch>
    <changeMoveSelector>
      <selectionOrder>ORIGINAL</selectionOrder>
    </changeMoveSelector>
    <acceptor>
      <entityTabuSize>7</entityTabuSize>
    </acceptor>
    <forager>
      <acceptedCountLimit>1000</acceptedCountLimit>
    </forager>
  </localSearch>

</solver>
```

FIGUR 14: EKSEMPEL PÅ EN SOLVERCONFIG.XML FIL

I figur 15 bygges en Solver opp ved hjelp av en xml-fil. Deretter bygges det opp et utgangspunkt for en løsning. Dette innebærer at løsningen får sine samlinger av planentiteter og problemfakta innsatt. Planvariablene trenger ikke å få tilegnet noen verdi i dette stadiet. Utgangspunktet brukes som parameter i Solver-objektets solve metode. Denne metoden setter verdiene til planvariablene som spesifisert i innstillingene for Solverens construction heuristikk og beveger seg deretter gjennom løsningsmengden ved hjelp av algoritmen definert i innstillingene for localSearch. Etter at optimaliseringen terminerer kan den beste løsningen hentes ut med Solverens getBestSolution metode.

```java
public static void main(String[] args) {
	Problem dataset;
	Solver sudokuSolver = SolverFactory
		.createFromXmlResource(xmlResource).buildSolver();
	SudokuSolution initialSolution = new SudokuSolution(dataSet);
	sudokuSolver.solve(initialSolution);
	SudokuSolution bestSolution = (SudokuSolution)
		sudokuSolver.getBestSolution();
}
```

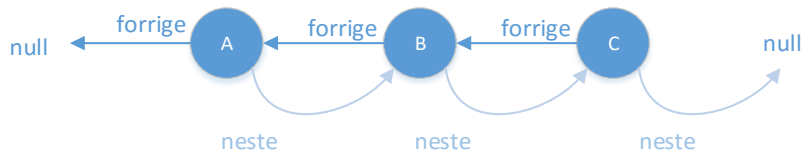
FIGUR 15: EN SOLVER BYGGES OG BRUKES DERETTER FOR Å FINNE EN LØSNING PÅ PROBLEMET.

Skyggevariabler

Skyggevariabler er variabler som har verdier som kan beregnes ut ifra verdiene til de genuine planvariablene. Hver gang en genuin Planvariabel får sin verdi endret, sikres det at skyggevariablene som er tilknyttet denne verdien endres tilsvarende (Optaplanner Team, 2016, s. 100). Merk at skyggevariabler ikke trenger å være en direkte kopi av en genuin planvariabel, men kan også være en transformasjon av dataen. Bruk av skyggevariabler gjør beregning av føringer enklere da verdiene allerede er tilgjengelige og ikke må regnes ut i for eksempel DRL regler ved bruk av Drools. Man kan

definere selv hvordan en skyggevariabel skal regnes ut ved å definere en egen klasse som implementerer VariableListener interfacet. Denne klassen definerer nettopp hva som skal skje når en planvariabel eller planentiteten som inneholder variabelen endrer status. På skyggevariabelens get-funksjon annoteres @CustomShadowVariable med en link til VariableListener-klassen og den genuine planvariabelen som skal speiles. Optaplanner støtter i tillegg ferdige implementasjoner for typiske bruksområder for skyggevariabler. To slike implementasjoner inkluderer toveisskyggevariabler og ankerskyggevariabler.

Bi-Directional skyggevariabler, eller toveisskyggevariabler er skyggesiden i et variabelpar bestående av en genuin planvariabel og en skyggevariabel. Paret peker alltid mot hverandre eller ingenting. I figur 16 vises en lenke der bruk av toveisskyggevariabler gjør seg gjeldende. Hver planentitet har en link til forrige og neste objekt i kjeden. Variabelen "neste" er her definert som en skyggevariabel med en toveissammenheng til den genuine planvariabelen 'forrige' som peker på sitt eget objekt.



FIGUR 16: BRUK AV TOVEIS-SKYGGEVARIABLER I EN LENKE.

Anchor skyggevariabler brukes kun i spesielle lenker som dannes av såkalte lenkede planvariabler. Slike lenker, i likhet med figur 16, inneholder planentiteter, men har alltid rot i et problemfaktum, også kalt ankeret for lenken. En ankerskyggevariabel definert i hver planentitet i lenken vil alltid peke mot lenkens anker.

4.4 Valg av kart- og routingtjeneste

Ved valg av karttjeneste var det fem hovedkriterier som var avgjørende; at kartdataen kunne lastes ned og benyttes uten nettilgang, at det fantes et bibliotek for ruteplanlegging for den aktuelle kartdataen, at kartdataen hadde godt oppdatert datagrunnlag for Norge, at tjenesten var lett å sette seg inn i og at tjenesten ikke benyttet en lisens som stod i strid med at prosjektets kildekode må være lukket. Som et ekstra kriterium så gruppen også på kostnaden ved å benytte tjenesten.

Kartdataene må kunne lastes ned og brukes uten tilgang til nett. Dersom spørringer mot kartdataen skulle bli gjort over nett, ville dette skapt forsinkelser ved at algoritmen måtte ha ventet på svar. Dessuten ville algoritmen måtte utføre mange spørringer i løpet av et kort tidsrom. Dette er ikke disse tjenestene laget for og mange API-er knyttet opp til karttjenester på nett har restriksjoner for hvor mange spørringer som kan gjøres for eksempel hvert sekund.

Det må eksistere et bibliotek som knytter seg opp mot kartdataen og kan brukes ved ruteplanlegging. Minimumskravet for dette ruteplanleggingsbiblioteket er å kunne finne avstand og varighet med bil mellom to punkter. Valget av å benytte et slikt bibliotek ble gjort da en implementasjon av slik funksjonalitet av gruppen selv, ville gjort oppgaven større enn det gruppestørrelsen og tidsperioden oppgaven må utføres i tillater.

Algoritmen skal i første omgang testes opp mot hjemmetjenesten som opererer i Gjøvik-regionen. Det er derfor viktig at karttjenesten inneholder oppdatert og komplett veiinformasjon for Norge.

Karttjenesten må være lett å sette seg inn i. Dette innebærer at den er godt dokumentert og ikke krever mye forarbeid for å sette opp. Åpen og tilgjengelig kode vil regne positivt for dette kriteriet.

Et krav satt av ETC var at koden må være lukket. Dette utelukker tjenester som benytter seg av lisenser som for eksempel GPL, GNU General Public License, som krever at all kode som benytter seg av åpne tjenester under denne lisensen også må forbli åpne.

Som et ekstrakriterie vurderte gruppen også kostnad. ETC uttrykte tidlig at betalte tjenester ikke var en hindring. De var villig til å betale dersom valget falt på en tjeneste der dette var aktuelt.

Diskusjon rundt mulige valg

Karttjenestene som stod til vurdering var Google Maps, Open Street Map, kart fra Kartverket, Here Maps og Bing Maps. Tabell 6 viser en oversikt over noen av kvalitetene for disse karttjenestene.

TABELL 6: OVERSIKT OVER KARTTJENESTENE SOM STOD TIL VURDERING

Tjeneste	*Kan nedlastes*	*Bibliotek for ruteplanlegging*	*Datadekning i Norge*
Google Maps	nei	ja	God
Open Street Map	ja	ja	Bra
Kartverket	ja	nei	God
Here Maps	kun android and IOS	ja	God

Google Maps er en av de mest brukte karttjenestene i dag. Tjenesten krever tilgang til nett og kartdata kan derfor ikke enkelt nedlastes. Tjenesten har gode API-er for hovedsakelig mobilutvikling og web som dekker behovet for ruteplanlegging og har solide data for store deler av verden, inkludert Norge. Tjenesten er godt dokumentert og det er lett å finne hjelp om tjenesten på nett. Fordi bruken av tjenesten vil benyttes internt av bedrifter og ikke er åpent for alle, krever Google at deres Premium Plan License benyttes (Google Maps for Work, 2016). Etter epostkorrespondanse med representant fra Google Maps ble det bekreftet at disse planene starter på rundt 80.000 kroner i året og øker etter bruk.

Open Street Map er den mest populære Open Source karttjenesten (Open Street Map Wiki, 2016). Kartdataen er gjort tilgjengelig for bruk av alle og kan fritt lastes ned og benyttes uten nett. Flere biblioteker er blitt utviklet for å jobbe med kartdataen på mange ulike plattformer. Open Street Map har generelt god veidekning i Norge og utøver i tillegg mulighet for integrering av gratis veidata fra kartverket som er under arbeid fra frivillige (Open Street Map Wiki, 2016). Bruken av Open Street Map er blitt stor nok til at det finnes god dokumentasjon om bruk både for karttjenesten og routing-tjenester som benytter seg av dens data. Man kan benytte kartdataen i lukkede prosjekter gratis, men hver routing-tjeneste har ofte sin egen lisens og prisplaner.

Kartverket er ett statlig organ som har i oppgave å holde gode oppdaterte kart over hele Norge (Kartverket, u.d.). Kartene kan lastes ned og brukes offline. Gruppen fant derimot ingen routing tjeneste som er i stand til å benytte deres filformater for offline routing, men deres kartdata integreres løpende inn i Open Street Map. Kartdataen er frigitt under en creative commons lisens som tillater lukket bruk, men krever at Kartverket krediteres ved bruk.

Here Maps er en karttjeneste sameid av Audi, BMW og Daimler (Here Company, 2016). Tjenesten har mulighet for nedlasting av kartpakker til bruk uten nett, men kun for Android og IOS. I likhet med Google Maps sine API-er retter Here seg mot mobil og web hovedsakelig, og API-ene de tilbyr inneholder i stor grad samme funksjonalitet som Google sine. Here tilbyr kartdata med god dekning i Norge og store deler av verden. Tjenesten kommer med solid offisiell dokumentasjon, men har mindre eksterne hjelperessurser tilgjengelig enn de mer populære karttjenestene. Here Maps benytter seg av lignende lisensplaner som Google Maps med krav om at deres Enterprise mapping plan benyttes for intern bruk i en bedrift. Prisen starter på ca. 1400 kr per måned eller 14100 kr per år og øker etter bruk.

Open Street Map og GraphHopper

Kriteriet om at kartdataen måtte være tilgjengelig uten nett ble ansett som såpass viktig at karttjenestene som ikke tilbød dette raskt ble utelukket. Da algoritmen ikke skal kjøre på mobile enheter utelukket dette også Here Maps. Open Street Map med styrker som god dokumentasjon og flere routingbiblioteker og Kartverket med bedre kartdekning i Norge stod derfor igjen som sterke kandidater. På grunn av tidsbegrensninger ble det valgt å gå for Open Street Map da gruppen ellers ville måttet utvikle routingdelen selv. Dette hadde tatt fokus vekk fra utviklingen av algoritmen og krevd mye investering av tid. I tillegg gjorde det valget lettere at Open Street Map stadig oppdateres med integrering av data fra nettopp kartverket.

Ved valg av routingbibliotek for Open Street Map så gruppen hovedsakelig på om den hadde den funksjonaliteten som krevdes av algoritmen, at den var lett å ta i bruk og om lisensen biblioteket benyttet seg av ikke satte restriksjoner for oppdragsgivers krav om lukket kode. Gruppen valgte GraphHopper's java-bibliotek. Biblioteket tilbyr routing funksjonalitet mellom punkter for blant annet bil- og sykkeltransport. Biblioteket benytter Apache 2.0 lisensen som gjør at tjenesten kan brukes fritt så lenge bruken krediteres. Dersom veidekningen eller andre faktorer ved Open Street Map og GraphHopper gjør at ETC vil bytte karttjeneste dersom de velger å videreutvikle algoritmen, skal dette være lett å gjennomføre. For en utdypning av hvordan dette tas hensyn til, se avsnitt 5.1 Design av Algoritmen, Modularisering av karttjenesten.

4.5 Utviklingsrammeverk for presentasjonsapplikasjon

Gruppen så opprinnelig på å utvikle presentasjonsapplikasjonen som en desktop-applikasjon utviklet i Java med Swing for GUI da gruppen hadde tidligere erfaring med dette. En rask prototype viste imidlertid at det var mer kompleks å implementere grafisk visning av kart ved hjelp av disse verktøyene. Det ble i stedet besluttet å utvikle applikasjonen som en webapplikasjon. Dette valget ble tatt hovedsakelig av to grunner. HTML, javascript og CSS gjør det lett å implementere applikasjoner raskt. Spesielt utarbeiding av brukergrensesnitt for webapplikasjoner er lettere å utvikle og tilpasse enn for desktop applikasjoner (Magic Web Solutions, 2013). Den andre grunnen var at utvikling på web gjorde integrering med karttjeneste betraktelig enklere grunnet solide javascript API-er, god dokumentasjon og en stor og aktiv brukermasse.

Applikasjonens funksjonelle krav tilsier at applikasjonen skal presentere den samme dataen på ulike måter. Blant annet skal dataen vises i timelister og som kartvisning. Gruppen så tidlig at å benytte et MVC-mønster ville være fordelaktig som arkitekturmodell for denne applikasjonen. Rammeverket AngularJS er oppbygd ved å benytte MVCs konsepter og ble derfor et naturlig valg.

AngularJS

AngularJS er et rammeverk for utvikling av webapplikasjoner basert på javascript. Visjonen bak rammeverket er å forenkle prosessen ved å arbeide med AJAX-applikasjoner ved å gjøre utviklingsprosessen mer intuitiv. AJAX-applikasjoner er webapplikasjoner bestående av kun én side der data fra serveren dynamisk settes inn og byttes ut i statisk HTML-kode. Samtidig forsøker AngularJS å gjøre det enklere å teste, skalere og vedlikeholde applikasjoner (Green & Seshadri, 2013, s. 1). Dette gjøres ved å samle kjente konsepter fra andre utviklingsmiljøer:

- Model View Controller-mønsteret: En klar separering mellom databehandling (Model), logikk i applikasjonen (Controller) og visning av data til bruker (View). Viewene henter data fra modellen for å vise frem for brukeren. Når brukeren utfører en interaktiv handling som for eksempel ved å klikke på en knapp, responderer kontrolleren ved å endre data i modellen. Ved endring av Modellen sendes en melding til viewene slik at visningen av dataen kan oppdateres.
- Client-Side-Templates: Data og HTML-kode sendes separat til nettleseren som bygger opp innholdet selv, ved å la HTML inneholde parametere som kan byttes ut med argumenter av systemet som prosesserer dokumentet.
- Data Binding: Fungerer ved å binde ulike UI-komponenter til javascript felter og la dem synkroniseres automatisk uten å laste siden på nytt. Konseptet går godt i lag med MVC-mønsteret da data fra et view flyttes til modellen automatisk.
- Dependency injection: Avhengigheter mellom klasser skapes ved at klassene på strukturert vis, lister opp sine egne avhengigheter. Dette er med på å modularisere applikasjonen ved at moduler har begrenset kjennskap til andre moduler og kun snakker med moduler de har oppgitt en avhengighet til.
- Directives: Direktiver er utvidelser til vanlig HTML-tags som knytter spesiell funksjonalitet opp til DOM-elementer. Angular kommer med egne direktiver som ngBind, ngModel og ngApp. Man kan også utvikle egne direktiver til bruk med AngularJS.

5 Design

5.1 Design av Algoritmen

Det var ønsket fra oppdragsgiver at algoritmen skulle operere med to moduser. *Fastsatte ressurser* og *fordeling av ressurser*. Førstnevnte skulle forsøke å fordele oppdrag på et fast antall ansatte og biler mens sistnevnte skulle forsøke å redusere antall ansatte og biler som er nødvendig for å gjennomføre oppdragene. På grunn av tidshensyn har gruppen kun hatt tid til å designe og implementere den første modusen. Etter ett møte med ETC mot slutten av utviklingsperioden ble modusen for fastsatte ressurser allikevel utvidet for å gi mulighet til å redusere ressursbruk, se 8.1 Diskusjon, Avgrensninger som følge av tidsbegrensninger for detaljer rundt denne utvidelsen.

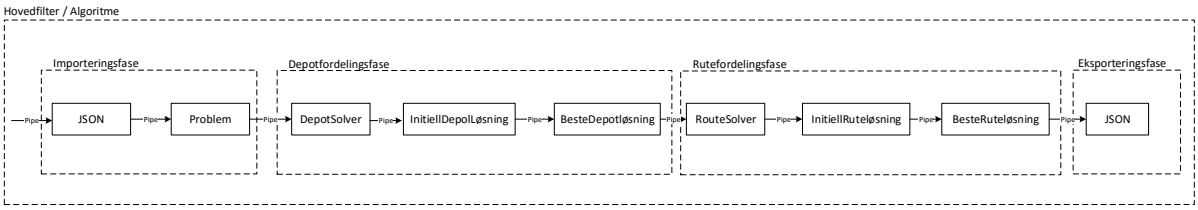
I modusen *fastsatte ressurser* tar algoritmen utgangspunkt i at antall ansatte og biler tilgjengelig på depotet er fast, den ser altså ikke på muligheten for å flytte på biler og ansatte mellom depoter. Algoritmen vil typisk kjøres før et skift og ha strenge krav til hvor lenge den kan kjøre før den gir en løsning. Algoritmen vil oppnå en kortere løsningstid ved å dele problemet i to. Første fase er å dele oppdrag mellom depotene og andre fase vil være å fordele oppdragene i ruter innad i depotet den er blitt tildelt.

Tidsbruken reduseres altså ved å kun benytte informasjon om oppdragenes avstand til hvert depot i første fase. Algoritmen har dermed ikke tilgang til avstander mellom de ulike oppdragene under denne fasen. Dette gjør at algoritmen må gjøre antagelser basert på distansen fra de ulike depotene, data om disse og deres ansatte og kjøretøy.

Etter at oppdragene er inndelt i depoter vil det så lages en løsning per depot der et av kravene som settes er jevn fordeling i forhold til depotets tilgjengelige ressurser. Distansen mellom hvert nodepar, altså distansen fra alle oppdrag til alle andre oppdrag og distansen mellom depotet og oppdragene regnes deretter ut.

Algoritmens flyt

Under følger en beskrivelse av algoritmens flyt



FIGUR 17: PIPE AND FILTER OPPBYGNINGEN ALGORITMEN VIL BENYTTE SEG AV.

Algoritmen vil benytte en pipe and filter oppbygning (Sommerville, 2011, s. 163). Den har et forhåndsdefinert grensesnitt for hvordan data kommer inn og hvordan data kommer ut og vil motta batch-jobber fra styringssystemet. Algoritmen vil så foreta fire filtreringsjobber identifisert i figur 17. I importeringsfasen oversettes JSON dokumentet som mottas fra styringssystemet til Java objektet Problem som benyttes videre i algoritmen. Data som finnes i Problem objektet benyttes så i depotfordelingsfasen til å bygge opp en DepotSolver og en initialløsning som Solveren kan optimalisere. Etter at visse kriterier definert i Solveren er møtt, slik som for eksempel tidsperioden gitt for å finne en løsning er nådd, avbrytes optimaliseringsforsøket og den hittil beste løsningen

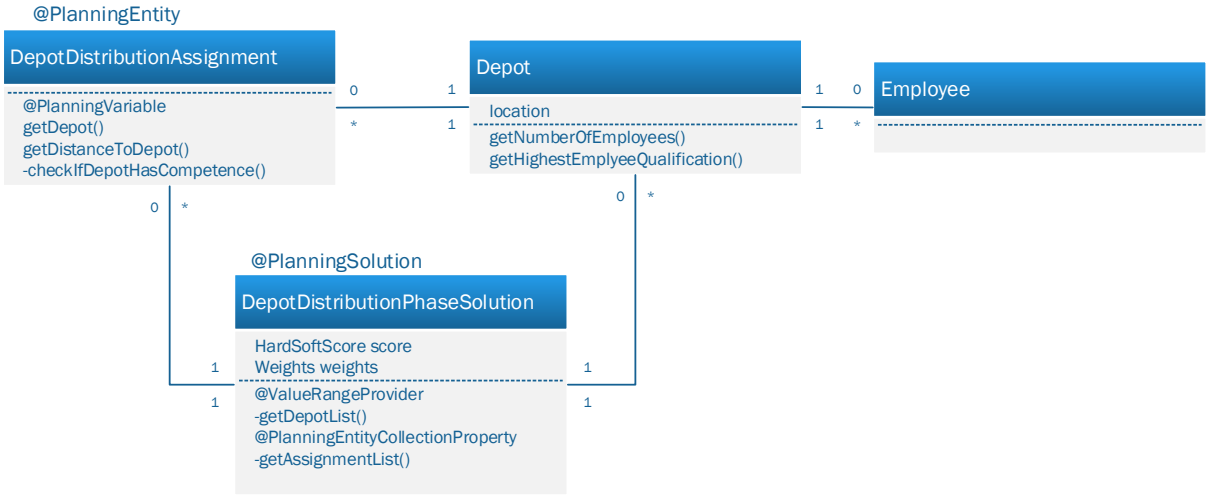
fungerer som grunnlag for neste filtreringsfase. Rutefordelingsfasen bygges så opp på samme måte, men baserer seg nå på data fra den hittil beste depotløsningen for å bygge opp en initialløsning for rutefordelingen. Når kriteriene definert for denne fasens Solver så er møtt blir den hittil beste ruteløsningen omgjort til JSON og returnert til styringssystemet i eksporteringsfasen.

Importerings- og eksporteringsfasene har ansvar for å konvertere til og fra JSON etter spesifikasjonene beskrevet i 3.5 Detaljert kravspesifikasjon, Datasett. Depot- og rutefordelingsfasene benytter så rammeverket Optaplanner for å optimalisere resultatet. Det er utarbeidet ett komplett klassediagram for fasene, se Vedlegg B – Detaljert Klassediagram. Diagrammet ble derimot så omfattende og tunglest at gruppen isteden valgte å trekke ut nøkkeldataen om de to fordelingsfasene og lage mer omfattende designdokumentasjon for disse.

Depotfordelingsfasen

Fasens mål er å fordele oppdrag mellom depoter med tanke på å minimere den totale avstanden mellom alle noder og deres tilegnede depot og holde antall oppdrag per depot jevn i forhold til depotets tilgjengelige ressurser, samtidig som krav om kompetanse og tidsvinduer overholdes.

Input til fasen er et datasett bestående av alle oppdrag som skal utføres i det aktuelle kjøre-intervallet og en liste med alle depoter med tilhørende ansatte og kjøretøy. Resultatet fra fasen vil være at alle oppdrag har blitt tilegnet et depot.



FIGUR 18: KLASSEDIAGRAM FOR DEPOTFORDELINGSFASEN

Som beskrevet i 4.3 OptaPlanner, Solveren så må en Solver vite hvordan fordelingsproblemet er definert. Dette gjøres ved hjelp av @-annotasjoner over klassenavn og funksjoner som vist i Figur 18 over. Løsningsplanklassen er her DepotDistributionPhaseSolution. Klassen inneholder to lister, en bestående av depoter og en av oppdrag. Depot-listen fungerer som en samling av problemfakta som reglene skrevet i Drools benytter for å sammenligne mot de tilsvarende problemfaktaene i hvert oppdrag. I tillegg til problemfaktaene inneholder Assignment-klassen en depot variabel. Det er dette depotet som Optaplanner forsøker å optimalisere via Drools reglene. Drools reglene brukes til å oppdaterer score variabelen i DepotDistributionPhaseSolution som indikerer hvor godt løsningen samlet sett oppfyller føringene.

For denne fasen har gruppen utarbeidet følgende føringer som må implementeres i DRL-regler, disse føringene som i Optaplanner deles i Hard og Soft Constraints tilsvarer harde og myke føringene slik som definert i 3.5 Detaljert kravspesifikasjon, Harde og myke føringer.

Harde føringer:

- Depotet har ikke nødvendig kompetanse: For at et oppdrag skal kunne plasseres på et depot, må depotet ha ansatte som har kompetanse som dekker oppdragets krav.
- Konflikt med tidsvinduer: Det kan ikke tilegnes flere oppdrag i et tidsvindu enn depotet har ansatte og biler.

Myke føringer:

- Distanse fra depotet til oppdraget straffes: Avstanden fra depotet til oppdraget vil tilegnes en negativ verdi basert på avstanden i mellom dem.
- Høy arbeidslastning på ett depot straffes: Løsningen tilegnes en negativ score basert på antall oppdrag tildelt et depot og ressursene tilgjengelig til dette depotet etter formelen:

$$Score = -\frac{antallOppdrag^2}{antallRessurser}$$

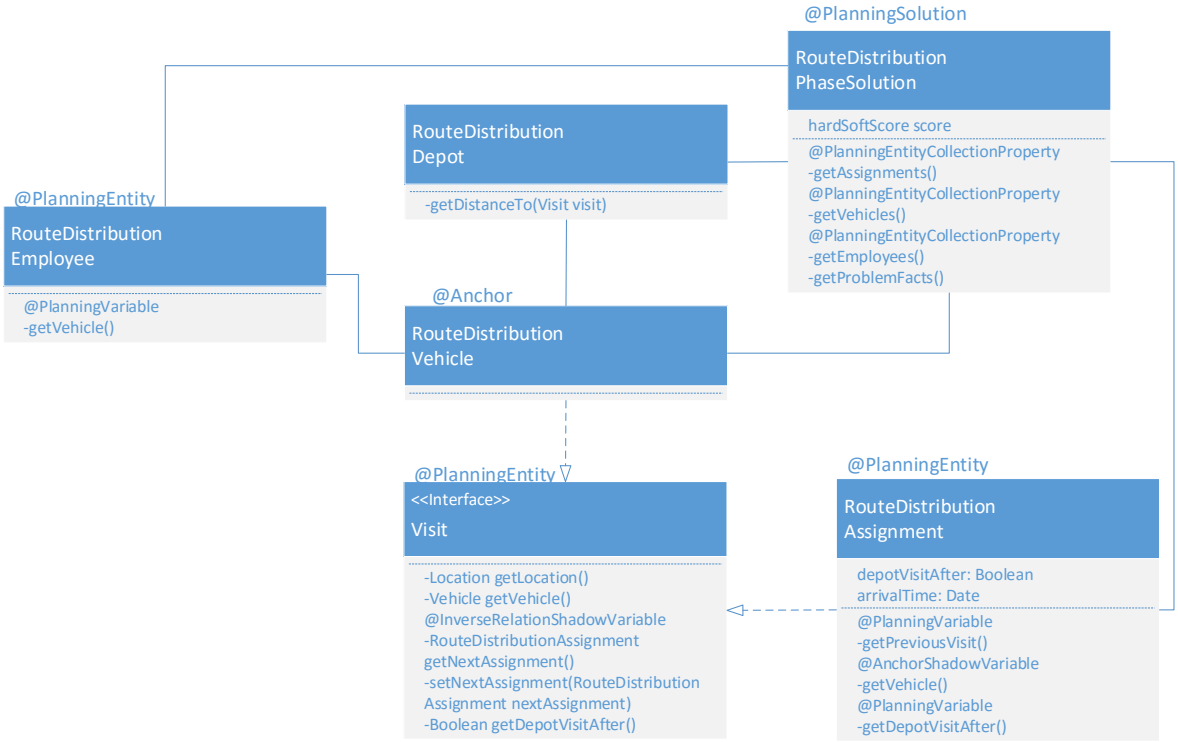
  Formelen er designet for at en perfekt fordeling skal gi den laveste negative påvirkningen. For eksempel vil to depoter med henholdsvis åtte og to bemannede kjøretøy fordele ti oppdrag perfekt ved at det første depotet utfører åtte og det andre utfører to. Med formelen gir dette en samlet negativ poengsum på -10. Dersom vi forskyver fordelingen ved at den største avdelingen dekker syv og den andre tre, eller den største dekker ni og den andre ett, blir poengsummen i begge tilfeller -10.625.

Rutefordelingsfasen

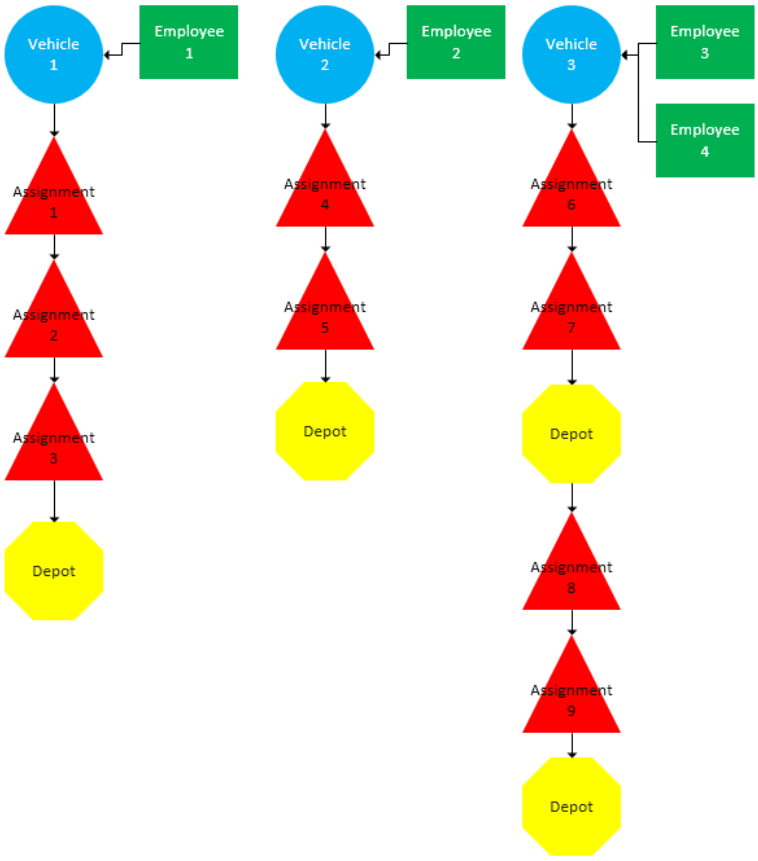
Denne fasens mål er å fordele oppdragene på et depot inn i kjøreruter som ansatte registrert på depotet skal besøke. Kjørerutene skal blant annet oppfylle tidsfrister og må utføres av ansatte som oppfyller oppdragenes krav til kompetanse. Fasen forsøker å redusere kjøreavstand og ventetid mellom oppdrag.

Input til fasen er et datasett bestående av ett depot med tilhørende ansatte og biler og oppdragene som ble tilegnet dette depotet i forrige fase. Resultatet fra fasen vil være at alle oppdragene har blitt lagt inn i en rute med en forventet ankomst- og avreisetid.

I likhet med fase én får Solveren for denne fasen også vite hvordan fordelingsproblemet er definert ved hjelp av @-annotasjoner over klasse- og funksjonsnavn som vist i Figur 19. Løsningsplanklassen er her RouteDistributionPhaseSolution i denne fasen. Dette betyr at den inneholder alle problemfakta og planvariabler. Klassen inneholder derfor lister av oppdrag, biler og ansatte. I tillegg inneholder klassen en score som indikerer hvor godt løsningen samlet sett oppfyller føringene definert i form av DRL-regler.



FIGUR 19: KLASSEDIAGRAM FOR RUTEFORDELINGSFASEN.



FIGUR 20: BRUK AV LENKER FOR Å OPTIMALISERE REKKEFØLGE PÅ OPPDRAG

Algoritmen vil fungere ved å bygge opp lenker som vist i figur 20. Hver lenke starter med et Vehicle-objekt som tilsvarer en av kjøretøyene i depotet. Dette objektet kalles i Optaplanner ankeret for lenken og har restriksjoner satt av rammeverket til å måtte være et problemfaktum. Med andre ord kan Vehicle objekter ikke være planentiteter og må derfor ha statiske verdier gjennom optimaliseringen. Det vil kun eksistere ett kjøretøy i hver lenke og dette vil være det første objektet i lenken. Lenken vil ellers inneholde Assignment-objekter. Disse Assignment-objektene har en boolsk verdi som tilsier om det skal utføres et besøk til depotet før neste oppdrag. Et kjøretøys rute i løpet av algoritmens kjøreintervall er dermed representert ved disse objektenes plassering i lenken. Det første objektet som besøkes vil være Vehicle-objektets første etterfølger i lenken, det andre objektet den andre som besøkes også videre. Lenkene som består av ett Vehicle-objekt og Assignment-objekter representeres ved hjelp av klassemedlemmet Visit som også er et interface som begge disse klassene implementerer. Algoritmen forsøker å finne ulike løsninger ved å bytte rundt på Assignment-objekter og sette inn eller fjerne depotbesøk i lenken.

Hvert Employee-objekt vil være tilknyttet et spesifikt kjøretøy igjennom hele kjøreintervallet algoritmen optimaliserer for. Algoritmen vil forsøke å tildele Employee-objektene til et kjøretøy som sørger for at Assignment-objektenes krav i forhold til kompatibilitet og kompetanse ivaretas.

I likhet med første fase har gruppen utarbeidet regler for denne fasen som må implementeres ved hjelp av Drools.

Harde føringer:

- Kjøretøyet må ha den nødvendig kompetansen: For at en bil skal kunne besøke ett oppdrag må det finnes ansatte i bilen som har den nødvendige kompetansen til å dekke oppdragets kompetansekrav. Dette sjekkes ved at hvert kompetansekrav fra oppdraget sammenlignes med alle ansattes kompetanse i bilen.
- Kjøretøyet må ha nok ansatte til å utføre oppdraget: For at et kjøretøy skal besøke ett oppdrag må det være nok ansatte i bilen til å oppfylle oppdragets krav til antall ansatte. Dette verifiseres ved at hver av de ansatte sjekkes for om de er på jobb i tidsperioden oppdraget skal utføres.
- Kjøretøyet må komme i tide til harde tidsvinduer: Hvis oppdraget har ett hardt tidsvindu må en bil besøke oppdraget før tidsvinduet til oppdraget lukkes. Dette sjekkes ved å sammenligne ankomsttid med slutten av tidsvinduet. Hvis ankomsttiden er senere enn tidsvinduet tillater tilegnes løsningen en negativ score basert på formelen:

$$Score = \ tidsvinduSlutt - ankomstTid$$

- Kjøretøyet må ha minimum en ansatt: For at et kjøretøy skal kunne besøke ett oppdrag må det minimum være en ansatt i bilen. Dette sjekkes ved å gå igjennom listen av ansatte og se om noen er tilegnet denne bilen og er på jobb. Hvis dette brytes tilegnes løsningen en fastsatt negativ score.
- Kjøretøyet må ha et lovlig antall passasjerer: For at en bil skal kunne besøke ett oppdrag kan det ikke være flere ansatte i bilen enn det som er lovlig for denne bilen. Dette sjekkes ved å gå igjennom listen av ansatte og telle antallet som er tilegnet kjøretøyet og som er på jobb i tidsperioden, hvis dette er flere enn antall lovlige passasjerer definert for bilen tilegnes løsningen en negativ score basert på antall ansatte i bilen.
- Kjøretøyet må ha en ansatt som kan kjøre den: For at et kjøretøy skal kunne besøke ett oppdrag må det være en person i bilen som har lov til å kjøre bilen. Dette sjekkes ved å gå

igjennom listen av ansatte og sjekke om noen i kjøretøyet har nødvendig førerkort. Hvis dette brytes tilegnes løsningen en fastsatt negativ score.

Myke føringer:

- Total kjøreavstand ønskes redusert: En bil oppfordres til å besøke oppdrag i en rekkefølge slik at den totale kjøreavstanden reduseres. Dette gjøres ved å kombinere tre regler:
    - Avstanden fra ett besøk til neste straffes med en negativ score basert på avstanden imellom dem.
    - Hvis ett oppdrag etterfølges av ett depotbesøk før neste oppdrag regnes denne avstanden også med og straffes på lik linje som regelen over.
    - Avstanden fra siste besøk tilbake til depotet straffes også med en slik negativ score.
- Oppdrag ønskes besøkt innenfor angitt tidsvindu: En bil oppfordres til å besøke oppdrag i en slik rekkefølge at oppdragene besøkes innenfor det angitte tidsvinduet for oppdraget, selv der dette ikke er ett hardt krav. Dette sjekkes ved å sammenligne ankomsttid med slutten av tidsvinduet på samme måte som regelen for harde tidsvinduer. Hvis dette brytes tilegnes løsningen en negativ score basert på formelen:

$$Score = tidsvinduSlutt - ankomstTid$$

- Ventetid mellom oppdrag ønskes redusert: En bil oppfordres til å besøke oppdrag i en rekkefølge slik at ansatte ikke behøver å vente lenge ved et oppdraget før det kan påbegynnes, eventuelt oppfordres bilen å besøke depotet før neste oppdrag. Dette gjøres ved å tilegne alle løsninger hvor ansatte drar rett fra et oppdrag til neste en negativ score basert på ventetiden før oppdraget kan begynnes etter formelen

$$Score = ankomsttid - tidsvinduStart.$$

  Dersom oppdraget besøkes etter ett depotbesøk tilegnes det ingen ventetid, men dette straffes i regelen for å redusere avstand kjørt istedenfor.

- Høy arbeidslast på en bil straffes: Løsningen tilegnes en negativ verdi basert på antall oppdrag per bil etter formelen:

$$Score = -(antallOppdrag^2)$$

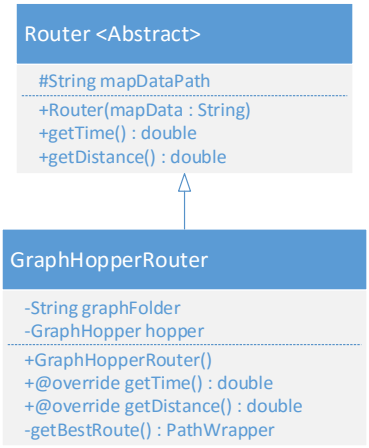
  Formelen sørger for at ujevne fordelinger tilegnes en større negativ poengsum i forhold til jevne fordelinger.

Modularisering av karttjenesten

Et krav fra oppdragsgiver var at kart- og routingtjeneste må kunne byttes ut, og nye implementasjoner må lett kunne implementeres uten for mye endring i kode. Dette var nødvendig da problemer knyttet til lisensrettigheter, mangel på service, betalte løsninger eller andre grunner kan føre til at et bytte kan være fordelaktig i fremtiden.

Gruppen utarbeidet et forslag på en løsning på dette problemet ved å lage en abstrakt klasse som implementerer metodene som det settes krav til for kart- og routingtjenestene – å finne avstanden i meter mellom to punkter og tiden i millisekunder det tar å bevege seg mellom disse punktene. Alle kall som gjøres i algoritmen skal være definert i Router som abstrakte metoder. De abstrakte metodene inneholder ingen implementasjon, men krever at subklasser implementerer dem. I Figur 21 vises hvordan klassen GraphHopperRouter implementerer metodene getTime og getDistance og

velger selv hvordan dette utføres. En ny implementasjon vil i likhet med GraphHopperRouter arve fra Router og må implementere de samme metodene.



FIGUR 21: DESIGN SOM STØTTER UTBYTTE AV SPESIFIKKE KART- OG ROUTINGIMPLEMENTASJONER.

Gruppen så også på muligheten for å dele mer opp ved å la kartdata og routing separeres i to uavhengige moduler. Dette viste seg å være lite hensiktsmessig da routingtjenestene som regel fungerte opp mot spesifikke karttjenester og derfor ikke kunne brukes på annen kartdata enn det de var programmert for.

5.2 Design av Presentasjonsapplikasjon

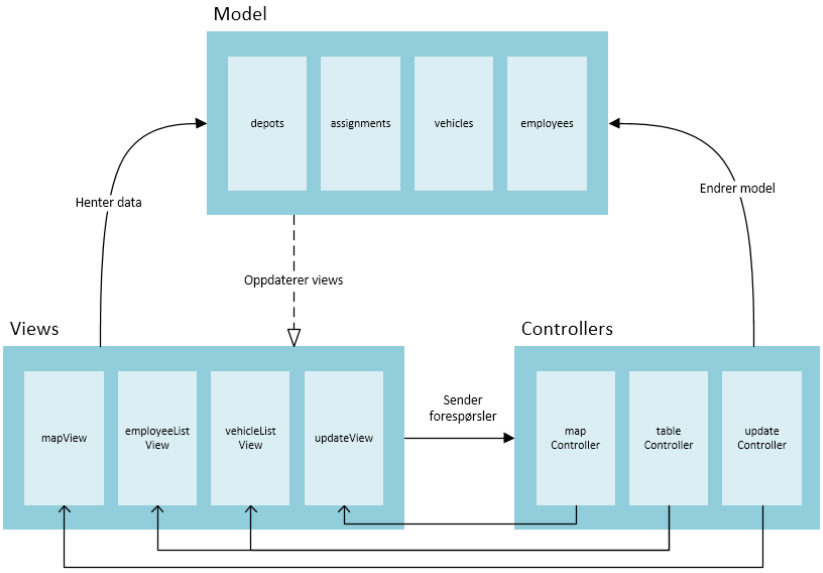
Presentasjonsapplikasjonen var nødvendig for gruppen under utviklingen av algoritmen som et visuelt hjelpemiddel for å bistå i vurderingen om hvorvidt algoritmen oppnådde gode nok resultater. Samtidig fungerte den som en demonstrasjon av hvordan en løsning generert av algoritmen kunne eksporteres til andre systemer. Applikasjonen ble kun utviklet som et verktøy for gruppens egen del og vil ikke bli benyttet av oppdragsgiver. Det var av den grunn lite fokus på interaksjonsdesign, og mer fokus på funksjonalitet knyttet til presentasjon av løsningen.

Presentasjonsapplikasjonens oppgave er å presentere dataen fra en løsning som algoritmen har fremstilt. En fil bestående av en generert løsning på JSON-format skal kunne lastes opp og deretter vises til brukeren. Brukeren vil ha mulighet til å se dataen som timelister per bil, timelister per ansatt og som kartvisning der ruter tegnes opp for hver bil i datasettet. Kartvisningen lar brukeren velge et kjøretøy i den gjeldende løsningen og viser dette kjøretøyets kjørerute igjennom en tidsperiode grafisk på kart. De to viewene relatert til timelister lar brukeren velge et kjøretøy eller en ansatt og viser så hvilke oppdrag som gjelder for akkurat det kjøretøyet eller den ansatte. Tabellen sorterer visningen etter når oppdragene skal besøkes med den tidligste som den øverste visningen i tabellen. Opplasting av løsning gir brukeren muligheten til å laste opp en løsning i JSON-format fra filsystemet.

Arkitekturmodell

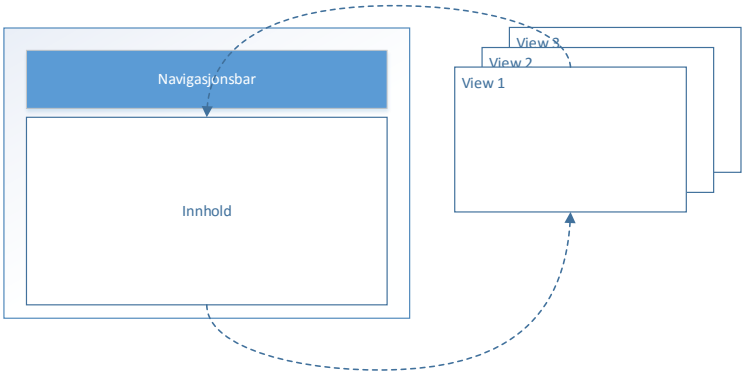
Applikasjonen følger Model-View-Controller-mønsteret. Dette var et naturlig valg fordi applikasjonen benytter det samme datagrunnlaget i de forskjellige visningene og presenterer den samme dataen på ulikt vis til brukeren. I Figur 22 under vises det hvordan MVC-mønsteret er brukt til å strukturere applikasjonen ved å dele den opp i tre separate deler. Modellen som inneholder data fra en løsning generert av algoritmen, viewene som danner grensnittet ut mot brukeren og fyller dette med data fra modellen og kontrollere som mottar forespørsler fra viewene, er ansvarlig for logikken bak sine

respektive views og utfører endring på modellen. Webapplikasjonen er utviklet som en dynamisk webside ved hjelp av rammeverket AngularJS.



FIGUR 22: MVC-PATTERN BRUKT I PRESENTASJONSAPPLIKASJONEN.

Applikasjonen opererer med fire ulike views: kartvisning, timelister per kjøretøy, timelister per ansatt og opplastning av løsning. Foruten om de to nært relaterte viewene som er ansvarlig for visning av timelister, har alle viewene egne kontrollere som utfører logikk innad i viewet. Alle disse separate kontrollerne har direkte tilgang til dataen som befinner seg i modellen. Kontrollerne for kart- og timelistevisning vil kun lese dataen som befinner seg i modellen, mens kontrolleren for opplastning av løsninger vil være ansvarlig for å konvertere JSON-filer til datastrukturer. Dette vil oppdatere modellen med ny data. De andre viewene vil ikke kunne vise data før filopplastningen har funnet sted og dataen er tilgjengelig.

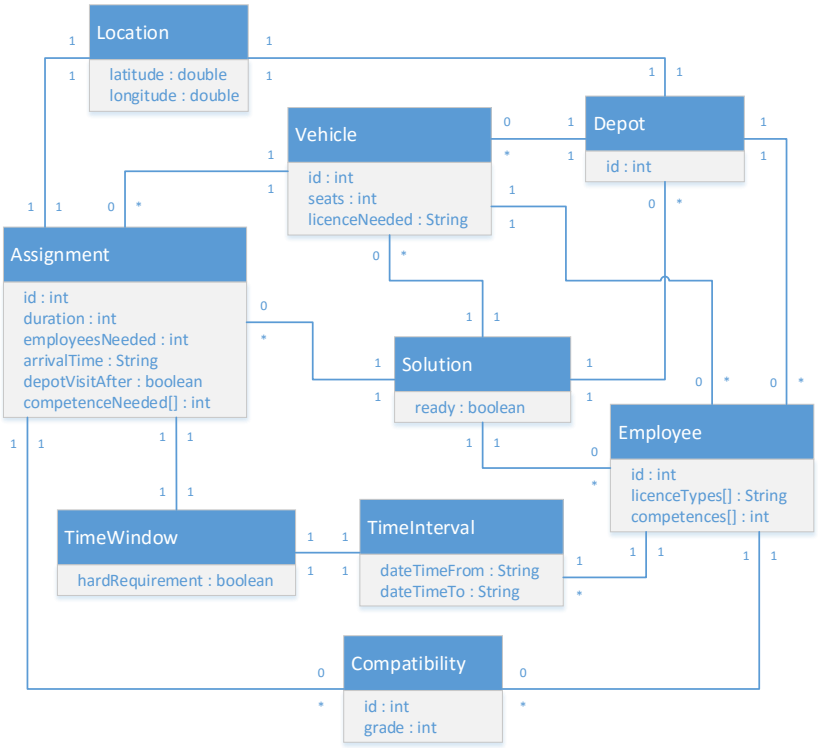


FIGUR 23: INNHOLD VIL BYTTES UT MED AKTUELLE VIEWS.

Siden bygges opp som som en Single Page Application (SPA). Designet baserer seg på en tung klient der all nødvendig HTML-, CSS- og javascriptkode lastes kun én gang. Dette er gjort for å slippe å lagre løsningen som lastes av brukeren på en server. Dette innebærer at en ny lasting av siden fører til tap av data og at løsningen må lastes inn på nytt. For å motvirke effekten av dette er hovedsiden som vises til brukeren oppbygd som en statisk HTML-side med en navigasjonsbar på toppen av siden og en innholdskomponent under som vist i Figur 23 over. Navigasjonsbaren inneholder lenker til de ulike

delene av siden. Ved å klikke på en link, plasseres det aktuelle viewet inn i innholdskomponenten. På denne måten kan man navigere seg rundt i webapplikasjonen uten å laste inn siden på nytt.

Datastruktur



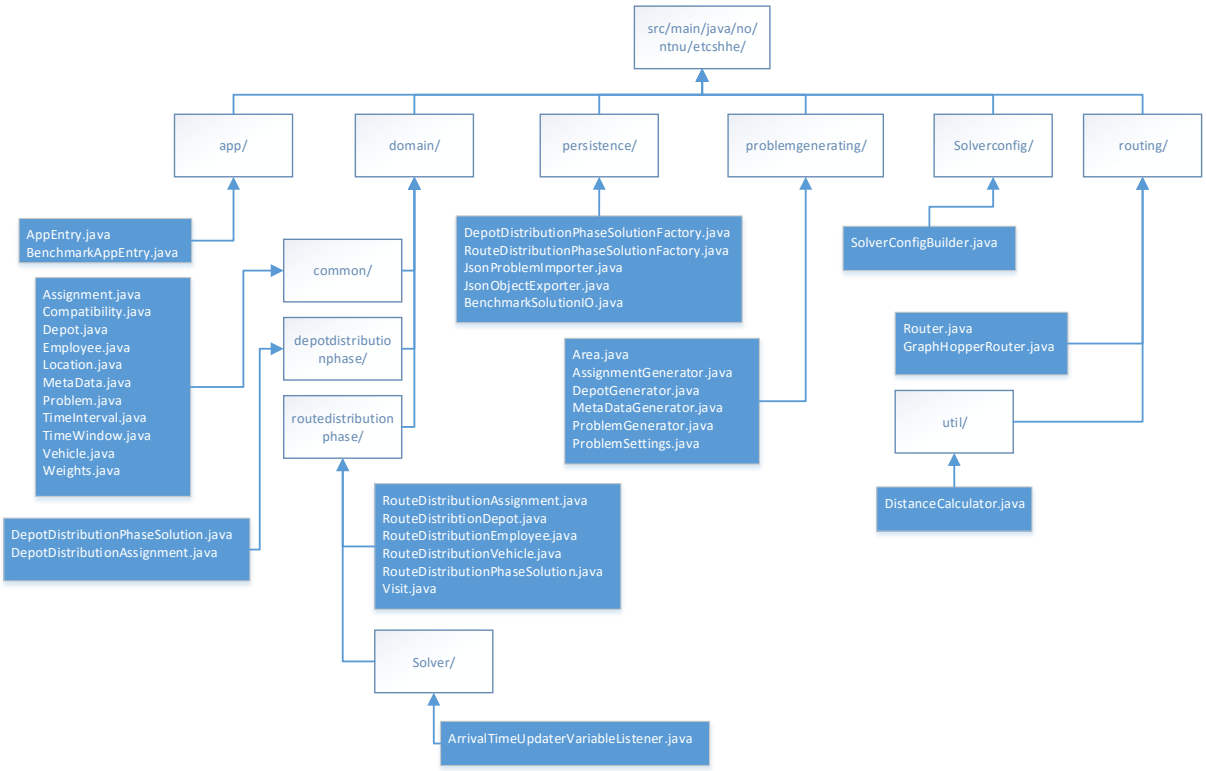
FIGUR 24: KLASSEDIAGRAM FOR MODELLEN

Modellen er vist i figur 24 og vil i hovedsak være identisk med oppsettet for JSON-filen generert som output fra algoritmen. Se 3.5 Detaljert kravspesifikasjon, Datasett for et eksempel på og en beskrivelse av et slikt datasett. I tillegg vil den boolske verdien ready tilsi om en løsning er blitt lastet korrekt opp. Denne verdien er med på å bestemme om views skal vise data til brukeren. Kun dersom en løsning er blitt korrekt lastet opp blir denne boolske verdien satt som sann. Modellen vil som tidligere nevnt være tilgjengelig for alle kontrollere i applikasjonen.

6 Implementasjon

6.1 Algoritme

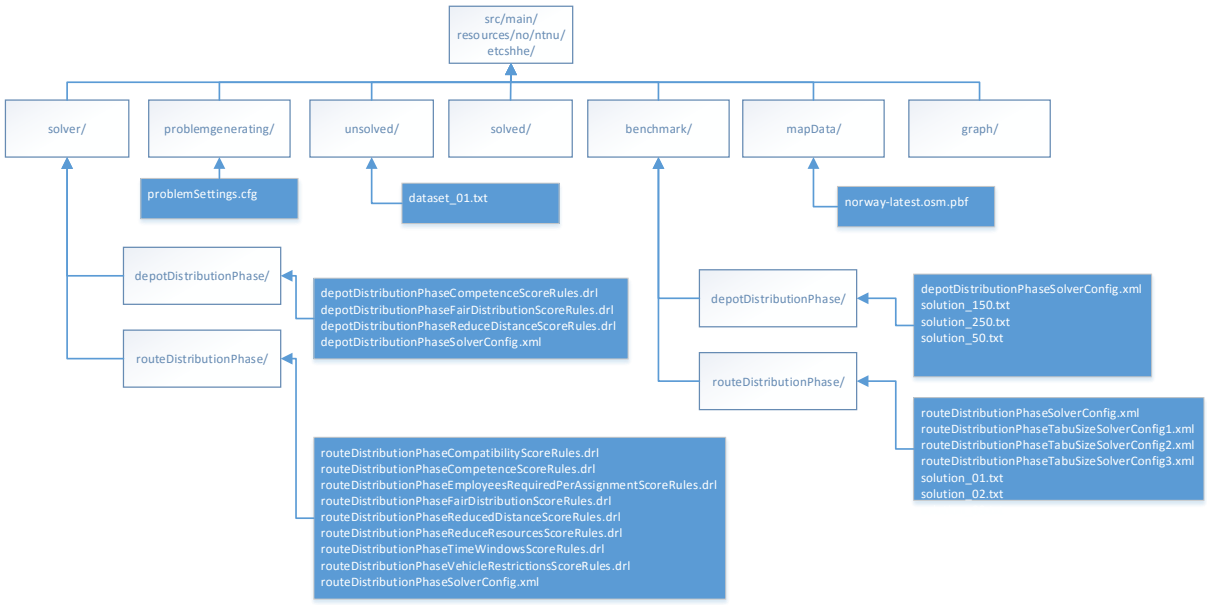
Filstruktur



FIGUR 25: FILSTRUKTUR FOR ALGORITMEN

Filstrukturen for algoritmen er strukturert som vist i figur 25. De hvite blokkene representerer her pakker, mens de blå blokkene viser filene i disse pakkene. En hovedinndeling er gjort ved å dele filer i pakkene app, domain, persistence, problemgenerating, solverconfig og routing. App-pakken inneholder inngangsportaler for algoritmen. Her finnes to filer: AppEntry og BenchmarkAppEntry. AppEntry inneholder algoritmens programflyt og er derfor ansvarlig for å starte algoritmen. BenchmarkAppEntry er en spesialversjon av AppEntry som utfører analyse av algoritmens optimalisering samtidig som algoritmen kjøres. Se Kapittel 7.4 Benchmarking for mer informasjon om benchmarking. Domain-pakken inneholder alle domeneklasser som benyttes av algoritmen. Denne pakken er igjen inndelt i tre underpakker: common, depotDistributionPhase og routeDistributionPhase. Common inneholder domene-klasser som benyttes på tvers av algoritmens to faser, mens de to andre pakkene inneholder Solution-klassene og arvede klasser fra common-pakken som representerer Planentiteter og Problemfakta for sine respektive faser. RouteDistributionPhase-pakken inneholder også underpakken solver som inneholder klassen ArrivalTimeUpdaterVariableListener ansvarlig for oppdatering av skyggevariabler som benyttes under denne fasen. Persistence-pakken inneholder fabrikklasser ansvarlig for å bygge opp Solution-objekter for de to fasene og klasser relatert til importering av datasett og eksportering av løsninger i JSON-format. Her finnes også klassen BenchmarkSolutionIO som i likhet med BenchmarkAppEntry benyttes under analyse av algoritmens optimalisering. Problemgenerating-pakken er relatert til generering av datasett for kvalitetssikring av algoritmen. Se 7.1 Realistiske og genererte datasett for en utdypende

forklaring av hvordan disse klassene ble tatt i bruk under utviklingen av algoritmen. SolverConfig-pakken som kun inneholder filen SolverConfigBuilder benyttes for å bygge opp konfigureringsinnstillinger som brukes av algoritmen under optimaliseringen basert på innstillinger fra datasettet som sendes inn til algoritmen. Routing-pakken inneholder filer relatert til kart- og routingimplementasjoner. Her finnes den abstrakte klassen Router og GraphHopperRouter-klassen som arver fra denne klassen. Disse klassene benyttes av klassen DistanceCalculator i underpakken util for å regne ut reisetid eller distanser mellom et sett med lengde- og breddegrader representert som Location-objekter.



FIGUR 26: FILSTRUKTUR FOR ALGORITMENS RESSURSER

Algoritmen benytter ressurser som er strukturert som i figur 26. De hvite boksene er her ressursmapper og de blå boksene inneholder filer i disse mappene. Solver-mappen inneholder DRL-filer med regler som har påvirkning på en løsnings poengberegning under løsningen av algoritmen. Den inneholder også konfigurasjonsfilene for de ulike fasene. Filene er organisert i undermappene depotDistributionPhase og routeDistributionPhase avhengig av hvilke faser de benyttes i under optimaliseringen. Problemgenerating-mappen inneholder en fil med innstillinger som leses under generering av datasett brukt for kvalitetssikring av algoritmen. I mappen unsolved forventer algoritmen å finne datasett som skal leses inn, og i mappen solved legges datasett som er ferdig løst av algoritmen. Formatet på disse filene er spesifisert i kapittel 3.5 Detaljert kravspesifikasjon, Datasett. Benchmark-mappen inneholder konfigurasjonsfiler for forskjellige benchmarkoppsett og datasett som skal testes i hver fase. Disse filene er organisert i undermappene depotDistributionPhase og routeDistributionPhase. MapData-mappen inkluderer kartdata som benyttes av den implementerte karttjenesten. Graph er spesifikk ved bruk av GraphHopper. Denne mappen inneholder grafdata generert fra kartdataen som tillater raskere oppslag av routing-tjenesten.

Avhengigheter

Algoritmen har avhengigheter til eksterne biblioteker og verktøy som vist i tabell 7.

TABELL 7: OVERSIKT OVER AVHENGIGHETER TIL EKSTERNE BIBLIOTEKER FOR ALGORITMEN.

Avhengighet	*Versjon*	*Lisens*	*Beskrivelse*
optaPlanner-core	6.3.0	Apache 2.0	Biblioteket brukt for løsning av constraint satisfaction problemer.
optaPlanner-benchmark	6.3.0	Apach 2.0	Biblioteket brukt for å teste løsningsmetoder implementert i Optaplanner-core
graphHopper	0.6.0	Apache 2.0	Routingtjeneste basert på open street map brukt til å finne distanser og reisetid mellom geografiske punkter.
Logback-Classic	1.1.6	Eclipse Public 1.0	Bibliotek brukt for logging.
gson	2.6.1	Apache 2.0	Javabibliotek for serialisering og deserialisering av java-objekter til og fra JSON.

Klassebeskrivelser

AppEntry

AppEntry-klassen starter algoritmen og er ansvarlig for den overordnede programflyten. Programmet starter med å laste inn og opprette et datasett som inneholder problemområdet og konfigurasjonskrav til Solveren. Dette benyttes av klassene SolverConfigBuilder og DepotDistributionPhaseSolutionFactory for å opprette en Solver og en initialløsning som denne så optimaliserer. Rutefordelingsfasen har tilnærmet samme fremgangsmåte, men RouteDistributionPhaseSolutionFactory benytter den beste løsningen depotfordelingsfasen til å bygge opp en liste med initialløsninger for depotene som routeSolver løser hver for seg. Løsningen konverteres så til JSON med JsonObjectExporter og skrives til fil.

JsonProblemImporter

Importøren har ansvar for å importere problemområdet og konfigurasjonskrav fra JSON-filen. Dette gjøres ved å oppretter ett Gson-objekt via en GsonBuilder. Gson-objektet må opprettes ved hjelp av en Builder fordi det benyttes Date-objekter i Problem-klassen som skal opprettes. Gson må derfor informeres om hvordan formatet på disse Date-objektene er. JSON-dataen leses så inn i en JsonReader som Gson-objektet bruker for å opprette ett Problem-objekt som har klassemedlemmer tilsvarende alle feltene i JSON-filen.

Problem

Problem-klassen inneholder problemområdet og konfigurasjonskrav. Problemområdet er representert av de to listene assignments og depots som sammen inneholder all data som skal til for å representere problemet som algoritmen skal optimalisere. Metadata-klassen inneholder konfigurasjonskravene til Solverene som skal brukes i de to fasene.

SolverConfigBuilder

SolverConfigBuilderen oppretter Solvere som brukes til å optimalisere de ulike fasene. Metodene buildDepotDistributionPhaseSolver og buildRouteDistributionPhaseSolver i denne klassen fungere på samme måte, men oppretter Solvere designet for sine respektive faser.

```
public static final Solver buildDepotDistributionPhaseSolver(Weights settings) throws
IllegalStateException {
        List<String> scoreRules = new ArrayList<>();
        if(settings.getCompetenceWeight() > 0) {
                scoreRules.add(DEPOT_DISTRIBUTION_SCORE_RULES_COMPETENCE);
        }
        if(settings.getFairDistributionWeight() > 0) {
                scoreRules.add(DEPOT_DISTRIBUTION_SCORE_RULES_FAIR_DISTRIBUTION);
        }
        if(settings.getDistanceReductionWeight() > 0) {
                scoreRules.add(DEPOT_DISTRIBUTION_SCORE_RULES_REDUCE_DISTANCE);
        }
        if(scoreRules.size() == 0) {
                throw new IllegalStateException("Competence, Fair Distribution or distance reduction
                must be included in rule set.");
        }
        return buildPhaseSolver(DEPOT_DISTRIBUTION_SOLVER_CONFIG_XML, scoreRules);
}


public static final Solver buildRouteDistributionPhaseSolver(Weights settings) {
        ...
}

private static final Solver buildPhaseSolver(String xmlResource, List<String> scoreDrlList) {
        SolverFactory solverFactory = SolverFactory.createFromXmlResource(xmlResource);
        ScoreDirectorFactoryConfig scoreDirectorFactoryConfig =
        solverFactory.getSolverConfig().getScoreDirectorFactoryConfig();
        scoreDirectorFactoryConfig.setScoreDrlList(scoreDrlList);

        return solverFactory.buildSolver();
}
```

FIGUR 27: FUNSKJON FOR Å DEFINERE EN SOLVER

Solvere bygges ved at vektene for hva som skal prioriteres under optimaliseringen leses av. Dette vises i figur 27. Vektingen avgjør hvilke regelsett som skal brukes. Dersom en regel er vektet større en 0 vil filstien til denne regelen legges inn i en liste av slike filstier som så sendes til funksjonen buildPhaseSolver sammen med filstien til xml-filen for fasen. Xml-filen for konfigurasjon av Solveren brukes til å opprette en SolverFactory som bygger selve Solveren. Drools reglene som skal vektes ligger ikke i xml-filen fordi dette må velges av brukeren. Disse reglene settes derfor programmatisk ved å hente ut en ScoreDirectorFactoryConfig fra SolverFactory-objektet. Den ferdige Solveren returneres så til appEntry.

DepotDistributionPhaseSolutionFactory

```
public DepotDistributionPhaseSolution createDepotDistibutionPhaseSolution(Problem dataSet) {
        DepotDistributionPhaseSolution solution = new DepotDistributionPhaseSolution();
        solution.setDepots(dataSet.getDepots());
        solution.setAssignments(
            convertAssignmentsToDepotDistributionAssignments(dataSet.getAssignments())
        );
        solution.setWeights(dataSet.getMetaData().getWeights());
try {
        calculateDistances(solution);
        } catch( InterruptedException ie) {
                ie.printStackTrace();
return solution;
}
```

FIGUR 28: FUNKSJON FOR Å OPPRETTE EN DEPOTDISTRIBUTIONPHASESOLUTION

Depotfordelingsfasen fordeler oppdrag ut på de forskjellige depotene fra datasettet ved hjelp av Solveren. For at Solveren skal kunne løse optimaliseringsproblemet må en initialløsning bygges opp fra Problem-datasettet. Denne løsningen bygges av funksjonen createDepotDistributionPhaseSolution i DepotDistributionPhaseSolutionFactory-klassen som vist i figur 28. For at fasen skal kunne optimalisere basert på å minimere avstanden fra depotet til oppdraget må disse distansene kalkuleres. Dette gjøres med en DistanceCalculator beskrevet senere i dette kapittelet. Løsningsplanen for depotfordelingsfasen kan hovedsakelig benytte seg av grunnklassene fra problem-klassen, men oppdragene som skal fordeles er planentitetsklassen og må derfor ha en planvariabel som kan endres.

DepotDistributionAssignment

```
@PlanningEntity
public class DepotDistributionAssignment extends Assignment {
        private Depot depot;

public DepotDistributionAssignment(Assignment assignment) {
                super(assignment);
        }

@PlanningVariable(valueRangeProviderRefs = {"depotRange"})
        public Depot getDepot() {
                return depot;
        }
```

FIGUR 29: DEPOTDISTRIBUTIONASSIGNMENT KLASSEN MED VIKTIGE FUNKSJONER

Grunnklassen Assignment har ingen planvariabel og det må derfor benyttes en arvet versjon kalt DepotDistributionAssignment som vist i figur 29 over. Denne klassen har ett Depot-klassemedlem som planvariabel. Klassen har også annoteringen @PlanningEntity over klassenavnet som definerer at dette er en planentitet og @PlanningVariable(valueRangeProviderRefs = {"depotRange"}) over get-funksjonen til planvariabelen som definerer at denne kan settes med verdier fra listen av depoter som finnes i løsningsplan-klassen DepotDistributionPhaseSolution.

DepotDistributionPhaseSolution

For at Solveren skal kunne løse optimaliseringsproblemet må den ha definert visse verdier i en klasse som implementerer Solution-interfacet. Denne klassen kalles en løsningsplan-klasse og er annotert med @PlanningSolution. Figur 30 under viser annotasjonene som implementeres og viktige funksjoner.

```
@PlanningSolution
public class DepotDistributionPhaseSolution implements Solution<HardSoftScore>, Serializable {

        private List<Depot> depots;
        private List<DepotDistributionAssignment> assignments;
        private Weights weights;
        private HardSoftScore score;

 @PlanningEntityCollectionProperty
 public List<DepotDistributionAssignment> getAssignmentList() {
        return assignments;
 }

 @ValueRangeProvider(id = "depotRange")
 public List<Depot> getDepotList() {
        return depots;
 }

 @Override
 public Collection<? extends Object> getProblemFacts() {
        List<Object> facts = new ArrayList<>();
        facts.addAll(depots);
        facts.add(weights);
        return facts;
 }

}
```

FIGUR 30: DEPOTDISTRIBUTIONPHASESOLUTION KLASSEMEDLEMMER OG VIKTIGE FUNKSJONER

Funksjonen getProblemFacts må overstyres fra Solution-interfacet. Funksjonen legger inn problemfakta som brukes i DRL-regler for å avgjøre om løsningen bryter med noen av føringene definert i Solveren. Disse klassene må ha statiske verdier og ikke endres under optimaliseringen. Weights-objektene som legges inn avgjør hvor stor påvirkning hver regel skal ha på løsningsplanen ved å multiplisere regelenes poengsum med de relevante vektene.

getAssignmentList funksjonen er annotert med @PlanningEntityCollectionProperty og definerer at det er denne listen med oppdrag som inneholder planentitetene i løsningsplanen. Funksjonen getDepotList er annotert med @ValueRangeProvider(id = "depotRange") og definerer hvilke verdier planvariablene i oppdragene kan ha. Dette gjøres ved at getter funksjonen for planvariabelen i DepotDistributionAssignment også er annotert med id="depotRange", som vist i figur 29 og figur 30 over. Funksjonene getScore og setScore brukes til å oppdatere poengsummen som tilsier hvor god løsningen oppfyller reglene definert for Solveren.

RouteDistributionPhaseSolutionFactory

RouteDistributionPhaseSolutionFactory er ansvarlig for å bygge opp en initialløsning av typen RouteDistributionPhaseSolution. Dette gjøres ved å konvertere data fra en løst DepotDistributionPhaseSolution der objektene omgjøres til andre objekter som er kompatible med fasen. For eksempel brukes en kopikonstruktør som vist i figur 31 til å omdanne et DepotDistributionAssignment til et RouteDistributionAssignment. Konverteringen er nødvendig fordi verdiene som skal endres under optimaliseringen er forskjellige i de ulike fasene.

```
@PlanningEntity
public class RouteDistributionAssignment extends Assignment implements Visit {

        public RouteDistributionAssignment(Assignment assignment) {
                super(assignment);
        }
        ...

}

public class Assignment implements Serializable {

        protected Assignment(Assignment assignment) {
                this.id = assignment.getId();
                this.location = new Location(assignment.getLocation());
                this.timeWindow = new TimeWindow(assignment.getTimeWindow());
                ...
        }
        ...

}
```

FIGUR 31: EN KOPIKONSTRUKTØR BRUKES TIL Å KONVERTERE OBJEKTER MELLOM FASENE.

Den nye fasen trenger også informasjon om reisetider fra hvert oppdrag til alle andre oppdrag og depotet, samt reisetiden fra depotet til hvert annet oppdrag. I likhet med DepotDistributionPhaseSolutionFactory-klassen benyttes DistanceCalculator-klassen til dette formålet.

RouteDistributionPhaseSolution

I likhet med løsningsplan-klassen for depotfordelingsfasen inneholder RouteDistributionPhaseSolution alle planentiteter og problemfakta for fasen, metoder som definerer domenene for de ulike planvariablene og funksjonen getProblemFacts som legger til problemfaktaene slik at de kan aksesseres i DRL-regler. Som nevnt vil denne fasen kjøres en gang per depot i datasettet og hver gjennomgang vil optimalisere innad i et gitt depot. Planentitetene er i denne klassen en liste av oppdrag som skal fordeles på kjøretøy og en liste av ansatte som er registrerte på det gjeldende depotet og skal tilegnes kjøretøy. Problemfaktaene består av en liste med de tilgjengelige kjøretøyene på depotet det skal optimaliseres innenfor og en referanse til dette depot-objektet.

RouteDistributionAssignment

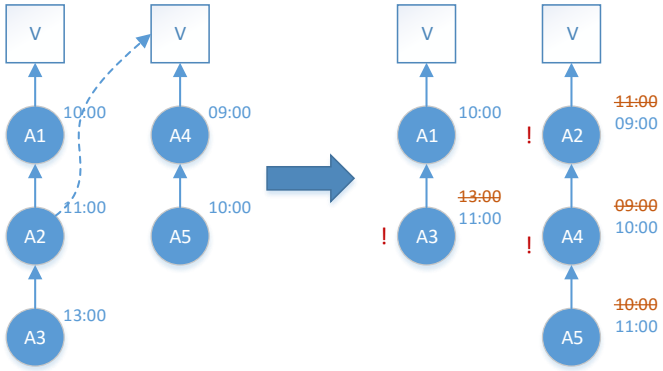
```
@PlanningEntity
public class RouteDistributionAssignment extends Assignment implements Visit {

       private Visit previousVisit;
       private Boolean depotVisitAfter;

       private RouteDistributionAssignment nextAssignment;
       private RouteDistributionVehicle vehicle;
       private Date arrivalTime;

       ...

}
```

FIGUR 32: PLANNING VARIABLER OG SHADOW VARIABLER I ROUTEDISTRIBUTIONASSIGNMENT

RouteDistributionAssignment er planentitet i rutefordelingsfasen og arver fra Assignment-klassen. Denne klassen inneholder to planvariabler som får sine verdier endret under optimaliseringen som vist i figur 32 over. PreviousVisit er en lenket planvariabel som definert i avsnitt 4.3 OptaPlanner, Skyggevariabler og som danner en lenke av objekter av denne klassen med et RouteDistributionVehicle som anker. For å gjøre skriving av DRL-regler enklere ble det også introdusert en toveisskyggevariabel, nextAssignment som peker til det oppdraget som har sin

previousVisit pekende på det aktuelle oppdraget og en ankerskyggevariabel som for hver av objektene i lenken, peker opp til lenkens anker. Dette medfører at lenken kan traverseres i begge retninger, og at man fra ethvert oppdrag enkelt kan finne ut hvilket kjøretøy det er tildelt. Fordi lenken opererer med to ulike objekter, implementeres Visit-interfacet i både RouteDistributionAssignment- og RouteDistributionVehicle-klassen. Domenet for previousVisit variabelen inkluderer alle kjøretøy og oppdrag definert i fasen. DepotVisitAfter er en boolsk-verdi som tilsier om et depotbesøk skal gjennomføres mellom dette oppdraget og det neste oppdraget i lenken. Domenet for denne variabelen er de boolske verdiene sant og usant. For en oppfriskning samt en grafisk fremstilling av hvordan disse lenkene ser ut, se 5.1 Design av Algoritmen, Rutefordelingsfasen. Ankomsttiden for oppdragene er representert ved Date-variabelen arrivalTime. Denne vil variere avhengig av et oppdrags plassering i lenken og er derfor også definert som en skyggevariabel. Når et oppdrags previousVisit endres til å peke på et nytt objekt må altså ankomsttiden regnes ut på nytt. Dette gjøres ved å implementere en egen VariableListener-klasse som definerer nettopp når og hvordan dette skal regnes ut. RouteDistributionAssignment inneholder også metoden getDepartureTime som regner ut avreisetiden fra oppdraget ved å se på ankomsttiden og legge til tiden det tar å utføre oppdraget.

ArrivalTimeUpdaterVariableListener



FIGUR 33:REKALKULERING AV ANKOMSTTIDER VED HJELP AV EGENIMPLEMENTERT VARIABLE LISTENER.

ArrivalTimeUpdaterVariableListener-klassen implementerer VariableListener-interfacet. Dette krever at klassen implementerer seks metoder som inneholder prosedyrer for hva som skal skje før og etter den aktuelle planentiteten legges til, før og etter den fjernes og før og etter en endring av denne verdien gjennomføres. Ankomsttiden skal rekalkuleres etter at en ny RouteDistributionAssignment legges til eller planvariabelen previousVisit endres. Metodene afterVariableChanged og afterEntityAdded er derfor de eneste funksjonene som ikke har tomme implementasjoner. I figur 33 vises en mulig endring av to lenker som kan oppstå under en optimalisering. Oppdraget A2 flyttes fra den første lenken og over til den andre. Dette medfører at både A2, A3 og A4 vil ha verdiene i previousVisit endret og vil derfor utløse den aktuelle metoden. Oppdraget der endringen ble oppdaget sendes med til denne metoden som et parameter. Den nye ankomsttiden regnes ut for oppdraget som forårsaket endringen og alle de etterfølgende objektene nedover i lenken, med mindre den utregnede ankomsttiden er den samme som ankomsttiden som er registrert hos oppdraget fra før. Det sistnevnte grepet vil reduserer unødvendig kalkulasjon. For eksempel vil oppdraget A4 i figuren allerede ha sin verdi ferdig kalkulert av at ankomsttider ble kalkulert ved endring av A2. Metoden trenger derfor ikke utføre ytterligere endringer.

Selve ankomsttiden regnes ut på ulikt vis avhengig av oppdragets plassering i lenken og om et depotbesøk skal gjennomføres mellom det forrige oppdraget og det nåværende oppdraget. Dersom oppdraget er det første som utføres antas det at oppdraget ankommes ved begynnelsen av oppdragets tidsvindu. Dersom et depotbesøk skal gjennomføres sammenlignes tiden for når depotet må forlates for å nå frem til det etterfølgende oppdraget ved starten av tidsvinduet og tiden depotet ankommes fra det forrige oppdraget. Dersom tiden depotet må forlates er senere enn tiden depotet ankommes, kan den ansatte vente på depotet frem til dette tidspunktet og oppdraget får sin ankomsttid satt lik starten på sitt tidsvindu. Hvis ikke må depotet forlates umiddelbart og reise videre til oppdraget. Ankomsttiden er da definert som tiden depotet ankommes addert med reisetiden til neste oppdrag. Ellers regnes den nye ankomsttiden ut ved å se på det forrige oppdragets avreisetid og addere verdien med reisetiden til dette oppdraget.

Visit

Visit-interfacet implementeres av RouteDistributionVehicle og RouteDistributionAssignment og representerer et besøk på en rute. Hovedformålet for dette interfacet er å kunne representere ruter som en lenke bestående av både kjøretøy og oppdrag. Variabelen som representerer lenken er derfor av typen Visit og ikke de spesifikke klassene. Interfacet sikrer også at besøk må ha et sett med koordinater ved å kreve at metoden getLocation implementeres. For kjøretøy vil dette si depotet kjøretøyet er tilknyttet sine koordinater, mens for oppdrag vil dette være koordinatene hvor oppdraget skal utføres. Et besøk må også kunne finne reisetiden fra et besøk til et annet besøk definert av metoden getDistanceTo.

RouteDistributionVehicle

RouteDistributionVehicle opererer som en problemfaktum i denne andre fasen. Objekter av klassen arver fra Vehicle-klassen og fungerer som ankere for lenkene som danner rutene som skal utføres i RouteDistributionfasen. Hvert kjøretøy har en link til depotet de er tilknyttet. Klassen inneholder også en metode som regner ut den totale tiden som kjøretøyet benyttes til kjøring. Dette regnes ut ved å følge lenken og addere reisetiden fra besøk til besøk til enden av lenken nås.

RouteDistributionEmployee

RouteDistributionEmployee-klassen representerer en ansatt under rutefordelingsfasen og er denne fasens andre planentitet. Planentitetens planvariabel er navngitt vehicle og tilsier hvilket kjøretøy denne ansatte er tildelt. Domenet for denne variabelen er gitt som alle kjøretøyene registrert på depotet den gjeldende optimaliseringen gjelder for. RouteDistributionEmployee har også de boolske metodene working og hasCompatibility. Metodene sjekker om den ansatte arbeider innenfor en periode definert som to dato-objekter og om den ansatte har høy nok grad av kompatibilitet i en gitt kompatibilitetsfaktor. Disse metodene blir benyttet i DRL-regler for utregning av en løsnings poengsum.

RouteDistributionDepot

RouteDistributionDepot er et problemfaktum under den andre fasen og arver fra klassen Depot. Denne nedarvede versjonen inneholder ingen ekstra funksjonalitet i forhold til et vanlig Depot-objekt, men ble lagt til for å øke leselighet og forståelse på tvers av de ulike fasene.

JsonObjectExporter

JsonObjectExporter er ansvarlig for å eksportere en liste av løste RouteDistributionSolution-objekter på formatet som er spesifisert i 3.5 Detaljert kravspesifikasjon, Datasett. For at formatet skulle samsvare med spesifikasjonen var det ikke nok å kun deserialisere objektene ved bruk av Gson på samme måte som ved innlesning av problemdatasett. En omstrukturering var nødvendig for å skille depoter, kjøretøy, ansatte og oppdrag i egne bolker.

Distance Calculator

Klassen DistanceCalculator er ansvarlig for å regne ut distanser og reisetid mellom en mengde koordinater, representert som Location-objekter. Dette gjøres ved hjelp av kart- og routingtjenesten. Klassen benytter flere tråder for å gjøre kalkulering raskere. Beregningene gjøres ved å opprette en ny instans av klassen DistanceCalculator. Det nye objektet oppretter en kobling til kart- og routingtjenesten gjennom sin konstruktør. Deretter er det definert to metoder for beregning av distanser og reisetid som kan kalles på objektet. Det første kallet tar imot to lister med Location-objekter og regner ut avstander fra objektene i den første listen, til objektene i den andre listen. Denne metoden benyttes i algoritmens første fase for å kun beregne avstander fra depotene til oppdrag. Den andre metoden tar imot kun én liste og regner ut avstander fra hvert objekt til alle andre objekter i listen. Denne metoden benyttes i rutefordelingsfasen. Metoden definerer også et boolsk parameter for å vurdere om det er reisetiden eller avstandene mellom objektene som skal beregnes. Se 8.1 Diskusjon, Bruk av distanse og tid for en beskrivelse av dette valget. Distansene blir lagret ved at hvert Location-objekt inneholder en hashtabell der nøklene til tabellen er andre Location-objekter. Nøklene linkes til et flyttall som representerer reisetiden i millisekunder, eller avstanden i meter fra det gjeldende Location-objektet til Location-objektet brukt som nøkkel.

Router og GraphHopperRouter

Implementasjonen av Router- og GraphHopperRouter-klassen følger beskrivelsen tidligere gitt i 5.1 Design av Algoritmen, Modularisering av karttjenesten. Den abstrakte Router-klassen inneholder metodene getTime og getDistance som klassene som arver fra Router må implementere.

```
public class GraphHopperRouter extends Router {
    private GraphHopper hopper;
    ...

    @Override
    public double getTime(double latFrom, double lngFrom, double latTo, double lngTo) {
        PathWrapper res = getBestRoute(latFrom, lngFrom, latTo, lngTo);
        return res.getTime();
    }

    private PathWrapper getBestRoute(double latFrom, double lngFrom, double latTo, double lngTo) {
        GHRequest req = new GHRequest(latFrom, lngFrom, latTo, lngTo)
                .setWeighting("fastest")
                .setVehicle("car")
                .setLocale(Locale.getDefault());
        GHResponse rsp = hopper.route(req);

        ...
        return rsp.getBest();
    }
}
```

FIGUR 34: GRAPHHOPPERROUTERS IMPLEMENTASJON AV METODEN GETTIME.

Figur 34 viser hvordan metoden getTime implementeres i klassen GraphHopperRouter. Denne metoden finner først den raskeste ruten mellom to par av bredde- og lengdekoordinater. Dette

gjøres ved å opprette en forespørsel i form av en GHRequest. Her spesifiseres blant annet hvilken transportmetode som skal brukes og om man skal prioritere raskere eller kortere ruter. Forespørselen sendes med GraphHopper-objektets route-metode som finner ruteforslag i henhold til forespørselen. Ruteforslagene returneres så i en GHResponse og den ruten som er vurdert som best brukes til å hente ut tiden mellom de to koordinatpunktene.

Solver xml og DRL filer

Xml

Som beskrevet i kapittel 4.3 OptaPlanner, Solveren defineres en Solver enten med en xml-fil eller programmatisk. Gruppen har benyttet seg av begge metodene. Solveren i fase en og to defineres både ved hjelp av en xml-fil og programmatisk.

```
<?xml version="1.0" encoding="UTF-8"?>
<solver>  <solutionClass>no.ntnu.etcshhe.domain.depotdistributionphase.DepotDistributionPhaseSolution</solutionClass>
  <entityClass>no.ntnu.etcshhe.domain.depotdistributionphase.DepotDistributionAssignment</entityClass>
  <scoreDirectorFactory>
    <scoreDefinitionType>HARD_SOFT</scoreDefinitionType>
    <initializingScoreTrend>ONLY_DOWN</initializingScoreTrend>
  </scoreDirectorFactory>
  <termination>
    <bestScoreLimit>0hard/0soft</bestScoreLimit>
    <unimprovedSecondsSpentLimit>30</unimprovedSecondsSpentLimit>
  </termination>
  <constructionHeuristic>
    <constructionHeuristicType>FIRST_FIT</constructionHeuristicType>
  </constructionHeuristic>
  <localSearch>
    <changeMoveSelector>
      <selectionOrder>ORIGINAL</selectionOrder>
    </changeMoveSelector>
    <acceptor>
      <entityTabuSize>7</entityTabuSize>
    </acceptor>
    <forager>
      <!-- Real world problems require to use of <acceptedCountLimit> -->
      <acceptedCountLimit>1000</acceptedCountLimit>
    </forager>
  </localSearch>
</solver>
```

FIGUR 35: XML SOLVER KONFIGURASJON FOR DEPOTFORDELINGSFASEN

Solveren for depotfordelingsfasen er definert som vist i figur 35 over. Her defineres DepotDistributionPhaseSolution som løsningsplanklassen og DepotDistributionAssignment som planentitetsklassen. ScoreDirectorFactory-elementets innehold definerer med innstillingen HARD_SOFT at løsningen skal bestå av to lag med føringer, en for harde og en for myke. ONLY_DOWN definerer at poengsummen kun kan bli lavere hvis en ny planvariabel initialiseres. Termination-elementet definerer når Solveren skal avslutte optimaliseringsforsøket. Her er dette definert ved at det avsluttes når ingen harde eller myke krav brytes eller når det ikke har blitt funnet noen forbedret løsning i løpet av 30 sekunder. FIRST_FIT verdien definerer construction heuristikken som skal benyttes, denne er ikke veldig intelligent og bygger bare opp en initialløsning ved å plassere oppdrag på depotet som gir den høyeste poengsummen (Optaplanner Team, 2016, s. 223) . LocalSeach-elementet definerer hvilken metaheuristisk algoritme som skal benyttes og parametere for denne. SelectionOrder definerer i hvilken rekkefølge endringer skal gjennomføres og verdien ORIGINAL betyr at dette skal gjøres i den rekkefølge de er definert (Optaplanner Team, 2016, s. 194). EntityTabuSize-elementet definerer hvor stor tabulisten til den metaheuristiske algoritmen skal være. Selve metaheuristikken defineres ikke i denne xml-filen siden den benytter seg av Tabu Seach

som er standardvalget. Med acceptedCountLimit-elementet defineres hvor mange trekk som skal evalueres (Optaplanner Team, 2016, s. 244).

```
  <constructionHeuristic>
    <queuedEntityPlacer>
        <entitySelector id="placerEntitySelector">
            <entityClass>no.ntnu.etcshhe.domain.routeDistributionPhase.RouteDistributionEmployee</entityClass>
            <cacheType>PHASE</cacheType>
        </entitySelector>
        <changeMoveSelector>
            <entitySelector mimicSelectorRef="placerEntitySelector"/>
        </changeMoveSelector>
    </queuedEntityPlacer>
</constructionHeuristic>
<constructionHeuristic>
    <queuedEntityPlacer>
        <entitySelector id="placerEntitySelector">
            <entityClass>no.ntnu.etcshhe.domain.routeDistributionPhase.RouteDistributionAssignment</entityClass>
            <cacheType>PHASE</cacheType>
        </entitySelector>
        <changeMoveSelector>
            <entitySelector mimicSelectorRef="placerEntitySelector"/>
            <valueSelector>
                  <variableName>previousVisit</variableName>
            </valueSelector>
        </changeMoveSelector>
        <changeMoveSelector>
            <entitySelector mimicSelectorRef="placerEntitySelector"/>
            <valueSelector>
                  <variableName>depotVisitAfter</variableName>
            </valueSelector>
        </changeMoveSelector>
    </queuedEntityPlacer>
</constructionHeuristic
```

FIGUR 36: ENDRINGER AV STANDARD CONSTRUCTION HEURISTIC I FASE TO

Fase to defineres på samme måte som fase én, men siden denne fasen hadde flere typer planning variables måtte dette defineres spesifikt ved å legge inn egendefinerte verdier for elementer i queuedEntityPlacer-elementet.

DRL

DRL-regler benyttes for å avgjøre en løsnings styrke. Under følger et utdrag av disse reglene og en forklaring av deres virkemåte. Disse reglene ble valgt som representanter fordi de viser et bredt spekter av funksjonalitetene brukt i Drools.

```
rule "MinimizeDistanceToDepot"
        when
                Weights($weight : distanceReductionWeight)
                $depot : Depot()
                        DepotDistributionAssignment(depot != null, depot == $depot,
                        $distanceToDepot : getDistanceToAssignedDepot())

        then
                scoreHolder.addSoftConstraintMatch(kcontext, -$distanceToDepot * $weight);
end
```

FIGUR 37: MINIMIZEDISTANCETODEPOT REGEL SKREVET I DROOLS RULES

Regelen "MinimizeDistanceToDepot" som vist i figur 37 er en av reglene som hører til depotfordelingsfasen. Regelen brukes for å avgjøre hvilken depot ett oppdrag skal tilegnes. Regelen begynner med å hente ut vektingen som avgjør hvor stor påvirkning regelen skal ha for løsningsplanens poengsum. Denne vekten er en integer verdi som multipliseres med sluttresultatet. For hvert depot objekt som eksisterer og som er tilegnet ett depot, hentes avstanden mellom

oppdraget og det tilegnede depotet. Avstanden multipliseres så med vekten og legges inn som et bidrag i løsningsplanens poengsum som en negativ verdi.

```
rule "VehicleMustHaveCompetence"

	when
		Weights($weight : competenceWeight)
		RouteDistributionAssignment(vehicle != null, $vehicle : vehicle,
			$competenceNeeded : competenceNeeded, $arrival : arrivalTime,
			$departure : getDepartureTime())

		$value : Integer() from $competenceNeeded

		not RouteDistributionEmployee( vehicle == $vehicle, working($arrival, $departure) == true,
		$value memberOf competences )

	then
		scoreHolder.addHardConstraintMatch(kcontext, - 1 * $weight);

end
```

FIGUR 38: VEHICLEMUSTHAVECOMPETENCE REGEL SKREVET I DROOLS RULES

"VehicleMustHaveCompetence" regelen benyttes i rutefordelingsfasen for å avgjøre om ett oppdrag kan løses av de ansatte tilgjengelig i kjøretøyet oppdraget har blitt tilegnet. Dette gjøres ved at for hvert oppdrag som er tilegnet et kjøretøy, bindes data om kjøretøyet, kompetansekrav og ankomst- og avreisetid opp til midlertidige variabler. Hvert kompetansekrav i oppdraget sammenlignes så med kompetansen til de ansatte som befinner seg i kjøretøyet og er på jobb. Hvis det ikke finnes noen ansatte i kjøretøyet som oppfyller kompetansekravet i oppdraget tilegnes løsningen en negativ score for denne kompetansen. Det må nevnes at denne og den tilsvarende "VehicleShouldHaveComatibility" regelen er svært ineffektive regler som vist i kapittel 7.4 Benchmarking.

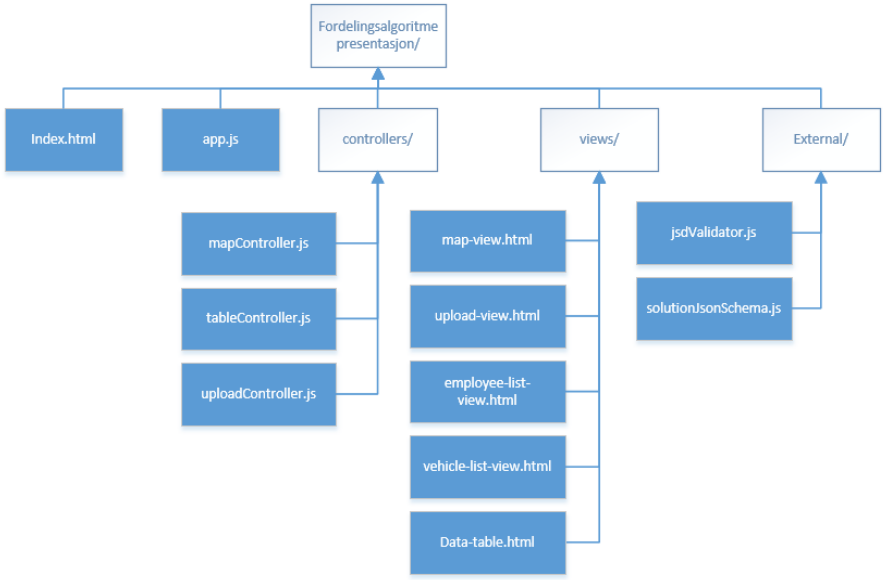
```
rule "DistanceFromPreviousVisitNoDepotVisit"
        when
                Weights($weight : distanceReductionWeight)
                RouteDistributionAssignment(previousVisit != null, previousVisit.depotVisitAfter == false,
                $distanceFromPreviousVisit : getDistanceFromPreviousVisit() )

        then
                scoreHolder.addSoftConstraintMatch(kcontext, - ($distanceFromPreviousVisit * $weight) /
                150);
end

rule "DistanceFromPreviousVisitIncludingDepotVisit"
        when
                Weights($weight : distanceReductionWeight)
                $assignment : RouteDistributionAssignment($previous : previousVisit != null,
                previousVisit.depotVisitAfter == true)

        then
                RouteDistributionVehicle vehicle = $assignment.getVehicle();
                scoreHolder.addSoftConstraintMatch(kcontext, -(($previous.getDistanceTo(vehicle) +
                vehicle.getDistanceTo($assignment)) * $weight) / 150);
end

rule "DistanceFromLastAssignmentToDepot"
    when
                Weights($weight : distanceReductionWeight)
                $assignment : RouteDistributionAssignment(previousVisit != null)
                not RouteDistributionAssignment(previousVisit == $assignment)
    then
                RouteDistributionVehicle vehicle = $assignment.getVehicle();
                scoreHolder.addSoftConstraintMatch(kcontext, - ($assignment.getDistanceTo(vehicle) *
                $weight) / 150);
end
```

FIGUR 39: REGLER FOR Å MINIMERE AVSTAND KJØRT SKREVET I DROOLS RULES

Total avstand ønskes redusert implementeres ved å kombinere de tre reglene "distanceFromPreviousVisitNoDepotStop", "distanceFromPreviousVisitIncludingDepotVisit" og "DistanceFromLastAssignmentToDepot" som vist i figur 39 over. Den første regelen går igjennom alle oppdrag og finner de som har en previousVisit der depotVisitAfter ikke er sann. Dette betyr at oppdraget er tilegnet en kjørerute og skal besøkes direkte etter forrige oppdrag. Der dette er tilfellet hentes avstanden mellom disse ut og legges inn som en negativ score i løsningsplanens poengsum. Neste regel fungerer på nesten samme måte, men denne slår inn dersom depotVisitAfter er sann. Altså dersom det skal utføres et stopp innom depotet imellom oppdragene. Den ekstra avstanden innom depotet må med i avstandskalkulasjonen. Den siste regelen beregner avstanden fra det siste oppdraget i kjeden og tilbake til depotet og legger inn denne avstanden som en negativ score på lik linje med de andre to reglene.

6.2 Presentasjonsapplikasjon

Filstruktur



FIGUR 40: FILSTRUKTUR FOR PRESENTASJONSAPPLIKASJONEN.

Filstrukturen for webapplikasjonen er vist i figur 40. Mappen controllers inneholder alle kontrollere definert i applikasjonen. Mappen views inneholder html-filene som utgjør viewene i applikasjonen. External mappen inneholder filer knyttet til validering av lastede løsninger. Filen jsdValidator.js inneholder ekstern kode som ikke er utviklet av gruppen selv. Øverst i hierarkiet finnes også filene index.html og app.js. Index.html inneholder den statiske html-koden som alltid vises til brukeren uavhengig av hvilket view som er aktivt. Dette inkluderer navigasjonsbaren som brukes for å navigere til nye views. App.js inneholder hovedmodulen for applikasjonen og er hovedsakelig ansvarlig for å knytte views og kontrollere som hører sammen opp mot hverandre.

Avhengigheter

Presentasjonsapplikasjonen har avhengigheter til eksterne biblioteker og verktøy som vist i tabell 8.

TABELL 8: OVERSIKT OVER AVHENGIGHETER TIL EKSTERNE BIBLIOTEKER FOR PRESENTASJONSAPPLIKASJONEN.

Avhengighet	*Version*	*Lisens*	*Beskrivelse*
Bootstrap	3.3.6	MIT	HTML, CSS og javascript rammeverk for utforming av responsive webapplikasjoner.
Google Material Icons	-	CC BY 4.0	Fritt bruk av ikoner rettet mot web og mobil.
AngularJS	1.4.9	MIT	Rammeverk sentrert rundt utvikling av Single Page Applications (SPA).
Angular ui-router	0.2.18	MIT	Fleksibel utbytte av views ved å organisere applikasjonens grensesnitt som en tilstandsmaskin.
Google maps API	-	Google Maps Standard Licence	API som gir mulighet for å inkludere kartvisning og uthenting av data fra Google Maps.
jsdValidator	-	FreeBSD	Validerer javascript-objekter og sjekker om de samsvarer med definerte jsd-skjemaer.

Klassebeskrivelser

Index.html og app.js

Index.html representerer den statiske delen av webapplikasjonen som presenteres til brukeren i samsvar med beskrivelsen gitt i 5.2 Design av Presentasjonsapplikasjon, Arkitekturmodell. Her inkluderes også avhengighetene applikasjonen har til Bootstrap, Google Material Icons, AngularJs, Angular ui-router og Google Maps API-et. Alle andre javascriptfiler som benyttes i applikasjonen lastes også inn. I figur 41 vises body-elementet i index.html. Dette elementet benytter ng-app direktivet som meddeler til AngularJS at dette elementet er rotelementet i applikasjonen. Direktivet spesifiserer også at modulen routerApp skal lastes sammen med applikasjonen. Denne modulen er definert i app.js filen. Bootstrap benyttes til å utforme navigasjonsbaren som vises på toppen av siden. Hver knapp på navigasjonsbaren har et ui-sref attributt knyttet til seg. Dette attributtet styrer hvilken tilstand applikasjonen skal befinne seg i etter knappetrykket, som igjen påvirker hvilket view som skal vises til brukeren. Div-elementet med direktivet ui-view vil fylles med innholdet fra viewet som er knyttet opp til den nåværende tilstanden til applikasjonen.

```
<body ng-app="routerApp">
    <!-- NAVIGATION -->
      <nav class="navbar navbar-inverse" role="navigation">
          <div class="navbar-header">
              <a class="navbar-brand" ui-sref="uploadData">ETC, datavisning</a>
          </div>
          <ul class="nav navbar-nav">
              <li><a ui-sref="mapView">Kartvisning</a></li>
              <li><a ui-sref="vehicleList">Timelister per bil</a></li>
              <li><a ui-sref="employeeList">Timelister per ansatt</a></li>
              <li><a ui-sref="uploadData">Last opp løsning</a></li>
          </ul>
      </nav>

      <!-- CONTENT CONTAINER -->
      <div class="container">
          <div ui-view></div>
    </div>
</body>
```

FIGUR 41: BODY-ELEMENTET I INDEX.HTML.

App.js inneholder routerApp modulen - hovedmodulen for applikasjonen. Modulen inneholder en stateProvider som knytter applikasjonens tilstand opp til spesifikke views og kontrollere. Dette er vist i figur 42 der skifte til tilstanden mapView skal føre til injeksjonen av innholdet i map-view.html inn i det tomme ui-view direktivet i index.html som tidligere nevnt. Kontrolleren som skal knyttes opp mot dette viewet er mapController. Hver tilstand applikasjonen kan befinne seg i er på tilsvarende måte listet opp slik.

```
$stateProvider.state('mapView', {
      url: '/mapview',
      templateUrl: 'views/map-view.html',
      controller: 'mapController'
}).state('uploadData', {
      url: '/upload',
      templateUrl: 'views/upload-view.html',
      controller: 'uploadController'
});
```

FIGUR 42: STATEPROVIDEREN ER ANSVARLIG FOR UTBYTTE AV VIEWS I APPLIKASJONEN.

Modulen er også ansvarlig for å gjøre modellen tilgjengelig til alle kontrollere. Dette gjøres ved å bygge opp en tjeneste som kontrollerne abonnerer på gjennom dependency injection som vist i figur 43. Denne tjenesten returnerer et dataobjekt som de ulike kontrollerne så kan endre eller aksessere dataen på.

```
routerApp.factory('data', function() {
      return {
            ready: false,
            vehicles: [],
            assignments: [],
            employees: [],
            depots: []
      };
});
```

FIGUR 43: MODELLEN GJØRES TILGJENGELIG VED Å REGISTRERE EN TJENESTE PÅ HOVEDMODULEN.

Upload-view og uploadController

Upload-view.html gir brukeren mulighet til å åpne en filutforsker og velge en løsning optimalisert av algoritmen i JSON-format fra filsystemet. Denne løsningen kan så lastes opp ved å trykke på knappen "last opp". Dette forårsaker et kall i viewets tilegnede kontroller som er definert i uploadController.js.

Denne metoden leser innholdet i filen som et tekstdokument og forsøker å konvertere JSON-data inn i et javascript-objekt. Dersom filen som er lastet ikke inneholder korrekt JSON-formatering avsluttes metoden og en feilmelding vises til brukeren. Dersom filen er korrekt formatert sjekkes innholdet i filen ytterligere ved bruk av et JSDValidator-objekt. Objektet benytter seg av et jsd-skjema som definerer hvordan objektene skal være strukturert med lovlige verdier for objektenes ulike felter. Skjemaet sikrer altså at dataen som mottas er kompatibel ved bruk i andre deler av applikasjonen. I figur 44 vises det hvordan skjemaet validerer en ansatts kompatibiliteter ved å kreve en array-struktur bestående av objekter der hvert objekt har nummer-feltene id og grade. Dersom datasettet består valideringen, oppdateres modellen med de aktuelle verdiene og den boolske verdien ready som tilsier at en løsning er korrekt lastet og klar til å vises i andre views settes til sann.

```
{
    Name: "compatibilities",
    Type: "Array",
    Values:
    [
      {
        Type: "Object",
        Attributes:
        [
          {
            Name: "id",
            Type: "Number"
          },
          {
            Name: "grade",
            Type: "Number"
          }
        ]
      }
    ]
}
```

FIGUR 44: UTDRAG FRA SOLUTIONJSONSCHEMA.JS

List-views og tableController

Employee-list-view.html og vehicle-list-view.html er ansvarlig for å vise timelister bestående av oppdrag som skal gjennomføres av henholdsvis ansatte og kjøretøy. De to viewene fungerer nokså likt. Begge krever at brukeren velger et depot og deretter en ansatt eller et kjøretøy som er registrert på depotet slik at timelisten kan vises for det valgte objektet. AngularJS definerer flere direktiver som benyttes flittig i disse viewene. Som vist i figur 45 under brukes for eksempel direktivene ng-repeat og ng-click på et hyperlink-element. Ng-repeat avgjør at dette elementet skal forekomme for hvert depot som er registrert i modellen. Ng-click definerer at klikk på hyperlinken kaller metoden updateDepot med det tilknyttede depotet som parameter i viewets tilegnede kontroller som er definert i filen tableController.js. Denne metoden setter variabelen currentDepot definert i kontrolleren til depotet som ble klikket på. På hyperlink-knappen vises depotets id ved at verdien i variabelen omringet av krøllparanteser vises når siden prosesseres.

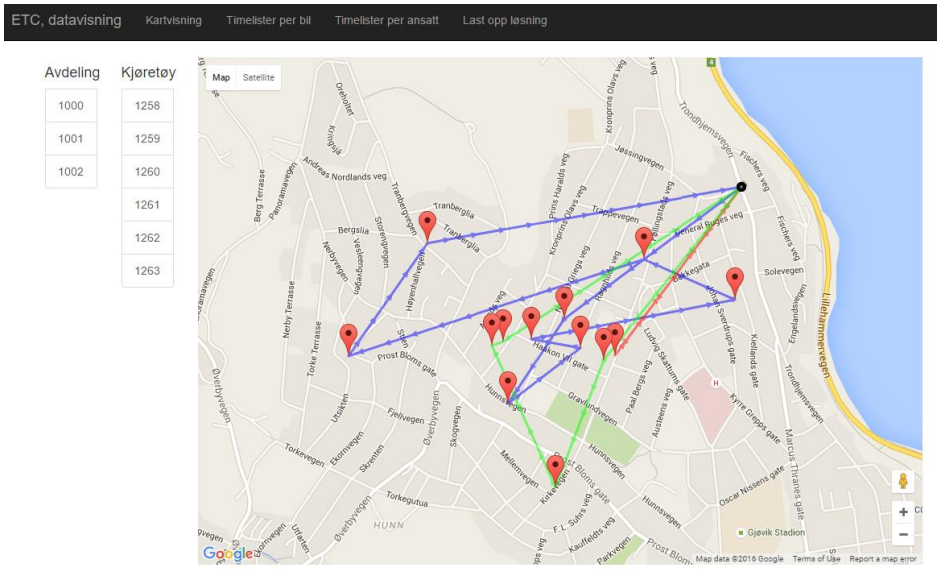
```
<div class="col-sm-1">
    <h4>Avdeling</h4>
    <a href="" class="list-group-item" ng-repeat="depot in data.depots" ng-click="updateDepot(depot)">
        {{ depot.id }}
    </a>
</div>
```

FIGUR 45: BRUK AV ANGULAR DIREKTIVENE NG-REPEAT OG NG-CLICK FOR Å INTERAKTERE MED KONTROLLEREN.

Listen med ansatte og kjøretøy bygges opp på tilsvarende vis ved hjelp av hyperlink-elementer og ng-repeat, men her brukes i tillegg et filter som utelukker alle objekter som ikke har tilknytning til det

valgte depotet som nå ligger lagret i variabelen currentDepot. Ved klikk på disse lenkene, oppdateres currentEmployee eller currentVehicle avhengig av hvilket view som er aktivt. Dette fører til at timelisten bestående av oppdrag vises i en tabell. Oppdragene filtreres slik at kun oppdrag som utføres av den valgte ansatte eller det valgte kjøretøyet vises. I tillegg sorteres de etter ankomsttid slik at oppdragene som utføres tidligst vises først i tabellen. Data om den valgte ansatte eller det valgte kjøretøyet vises også under denne tabellen.

Det siste viewet, map-view.html er ansvarlig for å vise rutene for kjøretøyene på kart. Google Maps benyttes for fremvisning av kart som vist i figur 46. Viewet har lik funksjonalitet som viewene relatert til timelistevisning. Ved å velge fra en liste med depoter vises kjøretøyene som er registrert på det valgte depotet. Dersom man deretter velger et kjøretøy fra listen, kalles metoden updateRoute med det valgte kjøretøyet som parameter i kontrolleren definert i mapController.js. Denne metoden er vist i figur 47 under. Først kalles metoden clearMap som fjerner alle tidligere ruter som er tegnet opp på kartet. Denne metoden er nødvendig dersom brukeren ønsker å se rutene for et nytt kjøretøy uten å måtte laste inn viewet på nytt. Metoden getFlightPlanCoordinates returnerer en multidimensjonell array bestående av ruter. Hver rute består av LatLng-objekter – Google Maps representasjon av bredde- og lengdegrader. En ny rute dannes hver gang kjøretøyet returnerer til depotet. Denne dataen danner grunnlaget for de to neste metodekallene i updateRoute, getFlightPaths og getMarkers. Disse metodene konverterer rutedataen til linjer og markører som kan tegnes på et Google-kart. UpdateMapToFitPoints benytter dataen fra rutene slik at kartet sentreres og forstørres på en slik måte at alle punktene som besøkes er synlig på kartet. Tilslutt utføres kallet drawMap som tegner inn den klargjorte dataen på kartet.



FIGUR 46: VISNING AV RUTER I VIEWET ANSVARLIG FOR KARTVISNING.

```
$scope.updateRoute = function (vehicle) {
        clearMap();
        var flightPlanCoordinates = getFlightPlanCoordinates(vehicle);
        currentFlightPaths = getFlightPaths(flightPlanCoordinates);
        currentMarkers = getMarkers(flightPlanCoordinates, getDepot(vehicle.depot));
        updateMapToFitPoints(flightPlanCoordinates);
        drawMap();

}
```

FIGUR 47: METODEN UPDATEROUTE I MAPCONTROLLER.JAVA

7 Kvalitetssikring

7.1 Realistiske og genererte datasett

Det ble tidlig i prosjektet identifisert utfordringer knyttet til kvalitetssikring av algoritmen da det ble klart at denne ikke kan garantere å finne det beste mulige resultatet. En mulig løsning på denne utfordringen var å sammenligne resultatene som genereres av algoritmen med reelle data fra hjemmetjenesten i Gjøvik og deres planlegging slik det gjennomføres i dag. Denne muligheten ble diskutert med Monica Kristiansen som representant fra hjemmetjenestens avdeling i Hunndalen i Gjøvik. Monica lovet å undersøke om en slik overlevering av data kunne la seg gjøre, men så to utfordringer knyttet til dette. Den første utfordringen var rent praktisk å hente ut den nødvendige dataen fra deres systemer og overføre dette til gruppen. Den andre utfordringer var hvorvidt det var innenfor lovverkets rammer å utlevere disse dataene. Etter en samtale med en kollega kom det frem at utlevering av denne type data ville kreve godkjennelse fra alle berørte pasienter, eller at all data ble anonymisert. På grunn av det høye antallet pasienter og arbeidet dette ville medføre for hjemmetjenesten lot ingen av disse alternativene seg gjennomføre.

Som et alternativ til reelle data fra hjemmetjenesten bestemte gruppen seg for å utvikle et selvutviklet program som ble omtalt som problemgeneratoren for å generere datasett som gruppen så kunne bruke for å vurdere hvorvidt algoritmen løste problemstillingen godt. Programmet genererer datasett som følger formatet definert i kapittel 3.5 Detaljert kravspesifikasjon, Datasett. Programmet fungerer ved at et geografisk område defineres hvor oppdrag kan genereres. Depotenes koordinater settes manuelt for å sikre at deres beliggenhet ikke plasseres vilkårlig innenfor det oppgitte geografiske området. Deretter opprettes randomiserte verdier for alle felter som er nødvendige i datasettet. Disse verdiene kontrolleres delvis ved hjelp av data i en konfigurasjonsfil som leses før problemgeneratoren genererer datasettet. Verdiene i konfigurasjonsfilen fungerer som rammer for verdiene som genereres. I figur 48 under vises et utdrag fra konfigurasjonsfilen som styrer genereringen av oppdrag. Her defineres blant annet antallet oppdrag som skal genereres, grensene for intervallet for varigheten på oppdragene, hvor mange kompetansetyper et oppdrag maksimalt kan kreve med mer.

```
# ==== Assignment settings ====
assignmentCount=250

minDuration=30
maxDuration=90

maxCompetencesPerAssignment=3

percentageForHardTimeWindow=20
minimumDurationTimeWindow=1
maximumDurationTimeWindow=8
workHoursStart=8
workHoursEnd=18
```

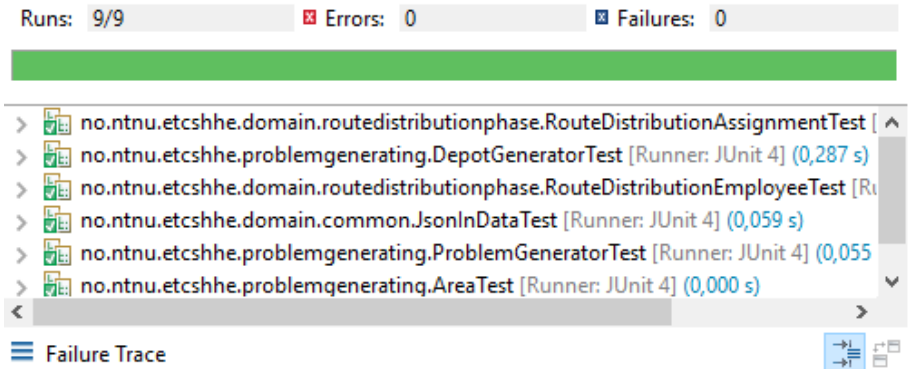
FIGUR 48: UTDRAG FRA KONFIGURASJONSFILEN PROBLEMSETTINGS.CFG

Utviklingen av programmet har foregått inkrementelt og parallelt med utviklingen av algoritmen. Utbedringer er gjort når gruppen selv har sett behov for det. For eksempel ble det nødvendig å generere fornuftige tidsvinduer for oppdrag først i rutefordelingsfasen da det var i denne fasen dette ble implementert i løsningen.

Programmet ble brukt blant annet til å genere datasett med varierende størrelser for å teste hvordan optimaliseringen skalerte med større datasett. Den ble også brukt for å generere datasett i lik skala som datasettene som benyttes av hjemmetjenesten i dag. Dette ble gjort ved å avtale nytt møte med hjemmetjenesten der det blant annet ble diskutert hvilke typiske verdier i datasettet hjemmetjenesten opererte med, se vedlegg F-møtereferater, møtereferat 2-hjemmetjenesten.

7.2 Enhetstesting og statisk testing

Enhetstester ble skrevet for individuelle biter av kildekoden for å sikre at den samsvarte med forventet oppførsel. Det ble ikke lagt fokus på testing av presentasjonsapplikasjonen da denne kun skulle benyttes av gruppen selv. Bruken av Optaplanner og eksterne biblioteker har gjort testingen av algoritmen til en utfordring. De testene som er skrevet av gruppen er derfor gjort på kodebiter som ikke har avhengigheter til slik utenomliggende kode. I figur 49 under vises det hvordan en gjennomkjøring av de implementerte testene ser ut i Eclipse. Det var planlagt å se nærmere på bruk av et rammeverk for mocking som ville tillate tester som avhenger av ekstern kode, ved å selv definere returdata fra metodekall som gjøres til slik kode. Tidshensyn førte imidlertid til at dette ikke ble gjennomført. Et eksempel på tester som vi helst skulle ha sett implementert er tester relatert til kjøring av DRL-reglene. Kjøringene av reglene er i Optaplanner tett knyttet opp til Solver-objektet som utfører optimaliseringen. Det ble gjort forsøk på å avfyre reglene ved å gå utenom Optaplanner, men dette viste seg å være vanskelig da alle reglene benytter seg av et scoreHolder-objekt som er knyttet opp mot en løsningsplans poengsum under optimaliseringen. Mangel på testing av disse reglene har skapt komplikasjoner, da hver regel kun utgjør et lite bidrag til løsningsplanens endelige poengsum.



FIGUR 49: GJENNOMKJØRING AV ENHETSTESTER I ECLIPSE.

SonarLint for Eclipse er en utvidelse som tillater kvalitetssikring av kildekode ved å kontrollere at koden følger et sett med regler. Reglene kontrollerer blant annet at koden følger standardkonvensjoner, er strukturert og lesbar og varsler også ved større sikkerhetsfeil. Hver gang en regel brytes, dukker regelbruddet opp som en oppføring i konsollvinduet som vist i figur 50. SonarLint ble kun benyttet for å oppdage regelbrudd i algoritmen og ikke i presentasjonsapplikasjonen. For det meste hjalp denne utvidelsen med å oppdage mindre problemer i koden. For eksempel ble importerte ressurser og variabler som aldri ble benyttet raskt påpekt. En større feil relatert til trådsynkronisering ble også oppdaget ved hjelp av SonarLint. Under kalkulering av avstander mellom oppdragene og depotenes beliggenheter, ble et Integer-objekt brukt for synkronisering. Integer-objekter er uforanderlige som vil si at dersom verdien i variabelen endres, blir en ny Integer-instans opprettet med den nye verdien. Fordi variabelen nå peker til et annet objekt enn det som

synkroniseres på kan dette føre til komplikasjoner. Dette ble løst ved å lage en egenimplementasjon av en indre klasse som det i stedet ble synkronisert på. SonarLint ble tatt i bruk sent i construction-fasen og kunne med fordel blitt benyttet tidligere. Dette kunne spart tid da gruppen ble nødt å sette seg inn i eldre kode for å fikse problemer rapportert av SonarLint lenge etter at koden var skrevet. Noen regelbrudd eksisterer fortsatt i prosjektet ved utgangen av prosjektperioden. Disse regelbruddene er for det meste relatert til kode som oppfattes som for kompleks og tunglest. Her tok gruppen en vurdering på de kodesnuttene det gjaldt og valgte at problemene ikke var store nok til at forandringer ble prioritert.

Description	Resource
Complete the task associated to this TODO comment.	RouteDistributionVehicle.java
Complete the task associated to this TODO comment.	RouteDistributionVehicle.java
Remove this unused import 'java.util.ListIterator'.	RouteDistributionVehicle.java
Remove this unused import 'no.ntnu.etcshhe.domain.common.Depot'.	RouteDistributionVehicle.java
Remove this unused import 'no.ntnu.etcshhe.domain.common.Employee'.	RouteDistributionVehicle.java
Rename this package name to match the regular expression '^[a-z]+(\.[a-z][a-z0-9]*)*$'.	RouteDistributionVehicle.java
Add the "@Override" annotation above this method signature	RouteDistributionVehicle.java
Refactor this code to not nest more than 3 if/for/while/switch/try statements.	RouteDistributionVehicle.java
This block of commented-out lines of code should be removed.	RouteDistributionVehicle.java

FIGUR 50: SONARLINT-RAPPORT FOR ROUTEDISTRIBUTIONVEHICLE.JAVA

7.3 Kommentering av kode

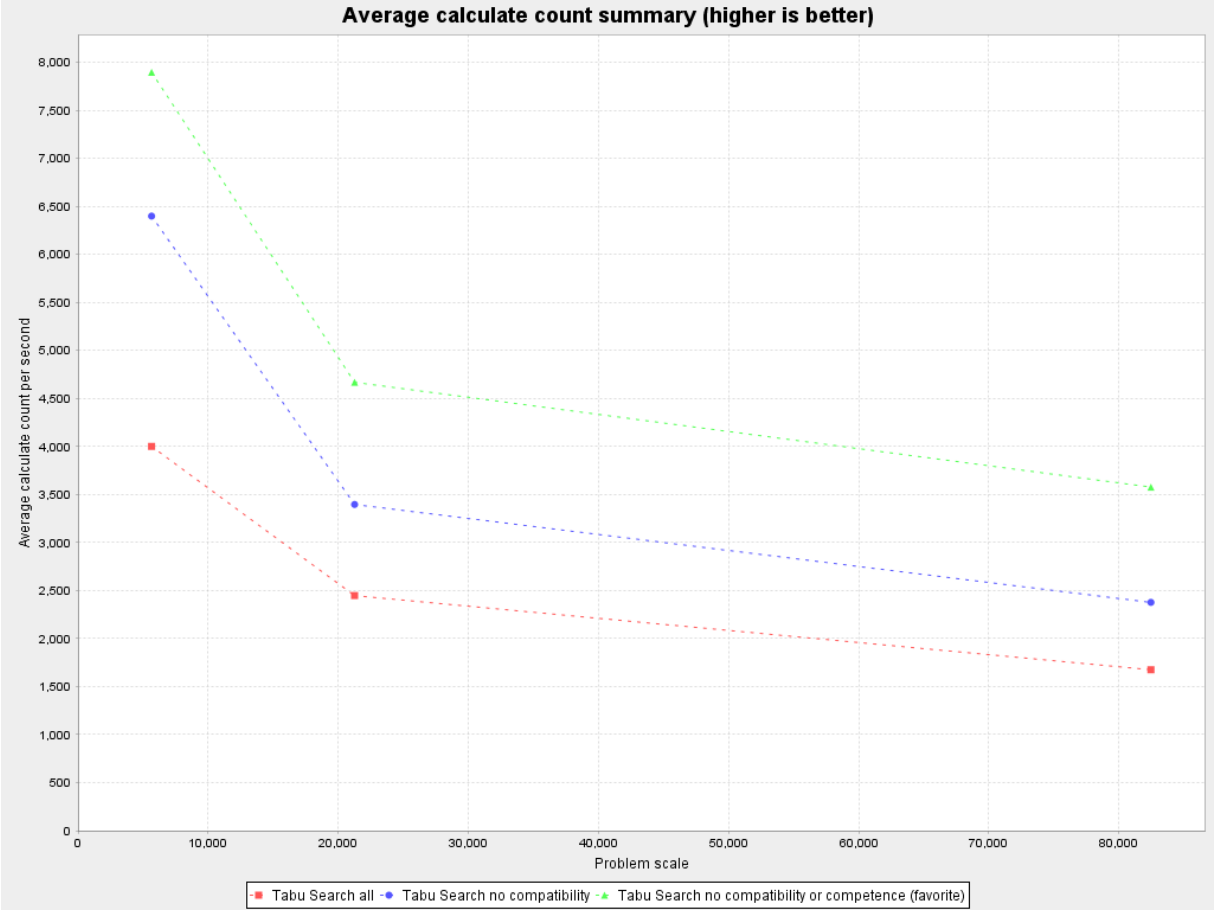
Javadoc er utført for all skreven kode, både i algoritmen og presentasjonsapplikasjonen. Dette er gjort både for gruppens egen del da dette gjør det enklere for gruppens medlemmer å sette seg inn i og benytte seg av hverandres kode. Klasse- og metodebeskrivelser kan lett sjekkes under arbeid med koden. Kommenteringen er også en viktig del av dokumentasjonen som kreves når ETC skal overta løsningen. Gruppens håp er at kommentarene skal gjøre det lettere for dem som eventuelt skal videreutvikle algoritmen å sette seg inn i algoritmens virkemåte. Kommenteringen er gjort på norsk for å følge kodestandarden satt hos ETC. Alle klasser, metoder og kode der gruppen mente det var nødvendig med ekstra beskrivelser er kommentert. For å sikre konsekvent formatering på kommentarene ble Eclipse konfigurert til å varsle dersom feil i javadoc-kommentarer fant sted. Dette ble som regel forårsaket ved at parametere for metoder ble endret uten å oppdatere javadoc tilsvarende.

7.4 Benchmarking

Optaplanner kommer med muligheten for å kjøre benchmark tester for å hente ut informasjon om forbedringspotensialer på forskjellige områder i algoritmen. Dette fungerer slik at man definerer en PlannerBenchmark og kjører igjennom en eller flere løsninger for å hente ut informasjon om hvordan Solvere løser problemsettene. Benchmarking har flere bruksområder. Det kan brukes for å sammenligne ulike Solver oppsett, altså ulike kombinasjoner av construction heuristikker, metaheuristikker og regelsett og det kan brukes for å se nyttig informasjon om spesifikke Solvere.

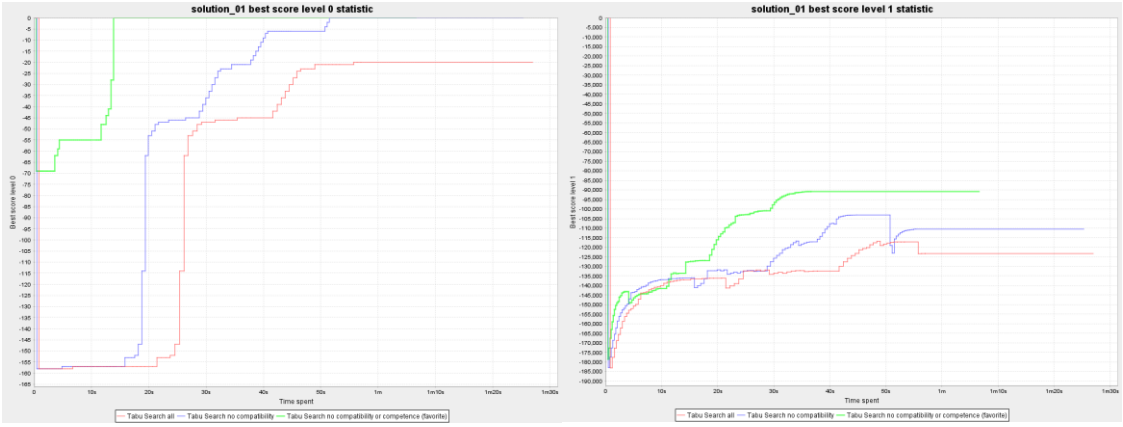
For å sammenligne ulike construction heuristikker og metaheuristikker kan man bruke en såkalt *blueprint* for standardiserte Solvere (Optaplanner Team, 2016, s. 265). Gruppen var derimot nødt til å endre på den standardiserte Solveren i implementasjonen. Dette medførte at det ikke fungerte å bruke en slik *blueprint* og at alle metaheuristikkene måtte defineres for seg selv. Dette var tidkrevende og lot seg ikke gjennomføre innenfor tidsrammen for kvalitetssikring. For å sammenligne ulike construction heuristikker må også vanskelighetsgraden av å plassere planentiteter defineres, dette ville sørget for en smartere oppbygging av den initiale løsningen og derfor raskere ført til gode løsninger.

Gruppen benyttet benchmarking for å hente ut informasjon om Solver-oppsettet som er konfigurert i vår implementasjon av algoritmen. Dette ga nyttig informasjon vedrørende flere områder av algoritmen slik som kostnaden ved å inkludere regler, informasjon om reglenes kvalitet og fornuftig størrelse på tabulisten til Solveren.



FIGUR 51: OVERSIKT OVER GJENNOMSNITTLIG ANTALL KALKULASJONER

Benchmarking av kalkulasjonskostnaden med og uten reglene "vehicleMustHaveCompetence" og "vehicleShouldHaveCompatibility" ble testet. Disse ble testet fordi gruppen antok at reglene var svært kostnadsfulle i form av antall kalkulasjoner i sekundet og dermed den endelige løsningen. Som vist i figur 51 over viste dette seg å stemme godt. Ved inkludering av begge reglene faller gjennomsnittlig antall kalkulasjoner med nesten 50%. Konsekvensen av dette er at algoritmen bruker to ganger så lang tid på å regne seg frem til en løsning med disse reglene enn uten. Dette påvirker igjen hvor godt sluttresultatet blir fordi Solveren ikke rekker å søke igjennom like mange løsninger. Slik Drools reglene er implementert må bilens kompetanse og kompatibilitet regnes ut på nytt hver gang ett oppdrag flyttes mellom biler. For å forbedre utregningen av disse reglene ønsket gruppen å implementere to skyggevariabler i Vehicle-klassen. Skyggevariablene skulle inneholde informasjonen om hvilken kompetanse og kompatibilitet som bilen samlet har tilgjengelig basert på de ansatte som er tilegnet bilen. Fordelen ved å benytte skyggevariabler til å holde på denne informasjonen fremfor å regne dette ut i DRL-reglene, er at utregningen av skyggevariablene kun skjer inkrementelt hver gang en ansatt flyttes mellom bilene. Denne utregningen vil derfor skje mye sjeldnere.



FIGUR 52: OVERSIKT OVER SCORE OVER TID FOR HARDE (VENSTRE) OG MYKE (HØYRE) FØRINGER

Figur 52 viser utregningen av beste Score over tid fra føringene implementert i Drools. På venstre side vises statistikk over resultatet av de harde føringene og på høyre de myke føringene. Statistikken viser at det er perioder med stor økning av score, fulgt av lengre perioder uten økning fulgt av store økning igjen hos begge føringene. Dette antyder ifølge Optaplanner dokumentasjonen (Optaplanner Team, 2016, s. 274) at noen trekk er heldige fordi de unnslipper lokale maksima. For å bøte på dette finnes det flere anbefalinger for hva som kan gjøres. Disse inkluderer å legge inn bedre moveSelectors for å gjøre mer fornuftige trekk eller endre DRL-reglene slik at påvirkningen av disse på en løsningsplans poengsum ikke er i form av hardkodede verdier.

Ved testing av Tabulistens størrelse ble det alltid den siste gjennomkjøringen av Solverene definert i Benchmark xml-filen som fikk det beste resultatet. Dette var ett kjent problem som følge av JIT (just in time) kompilering i Java og skulle løses ved å la benchmarkeren kjøre igjennom en oppvarmingsperiode før selve løsningen (Optaplanner Team, 2016, s. 265), dette løste allikevel ikke problemet hos gruppen. Testen ble derfor gjennomført ved å kjøre igjennom en og en Solver etter hverandre med avbrekk. Testens mål var å avgjøre hvilken tabuliste som gav det beste resultatet. Testingen viste at ut ifra de tre testverdiene på 10, 100 og 1000 på datasett med problemstørrelser på 5 655, 21 255 og 82 455 resulterte en tabuliste på 100 det beste resultat for harde føringer på alle datasettene. Dette gir en ide om hvor stor tabulisten burde være for datasett innenfor dette spennet. Det er også mulig å spesifisere en prosentandel av problemområdet som skal utgjøre størrelsen til tabulisten. Videre testing ville kunne avgjøre mer nøyaktig hvor stor denne prosentandelen burde være på.

8 Diskusjon og resultater

8.1 Diskusjon

Dette delkapittelet inneholder diskusjoner som ikke falt seg naturlig å diskutere andre steder i rapporten. Diskusjonen vil bære et selvkritisk preg der valgene som ble tatt vurderes i forhold til erfaring gruppen har tilegnet seg i løpet av prosjektperioden.

Avgrensninger som følge av tidsbegrensninger

Utover i prosjektperioden ble det nødvendig å avgrense oppgaven utover de avgrensningene foretatt i kapittel 1.7 Avgrensninger, da gruppen så at det ikke ville være mulig å implementere all funksjonalitet som var ønsket i løpet av den begrensede tiden tilgjengelig. Her følger en beskrivelse av tre ytterligere avgrensninger som ble foretatt på bakgrunn av tidsmangel.

Avgrensning 1: Gruppen var kun i stand til å designe og utvikle en av to ønskede moduser. Modusen som ble utelatt skulle være i stand til å kjøre over en lengre tidsperiode å forsøke å redusere antall ansatte og kjøretøy nødvendig for å utføre alle oppdrag. Selv om denne modusen aldri ble implementert var gruppen, i samarbeid med ETC i stand til å implementere en mellomløsning hvor fordelingsmodusen kunne utvides til å forsøke å redusere bruken av ansatte som var tilgjengelige ved å introdusere to nye føringer for algoritmen i form av DRL-reglene kalt "ReduceVehicleUsage" og "ReduceEmployeeUsage" som forsøker å unngå bruk av unødvendige ansatte og kjøretøy. Ved å inkludere disse føringene over en lengre periode vil en organisasjon være i stand til å danne seg et bilde angående områder der det er mulig å kutte ned på antall ansatte eller biler.

Avgrensning 2: Gruppen ble nødt til å innføre en avgrensing som innebar at en ansatt kun tildeles ett kjøretøy igjennom hele algoritmens kjøretid og derfor ikke kunne endre bil i løpet av tiden algoritmen optimaliserer innenfor. Dette medførte at kombinasjoner av ulike ansatte i kjøretøy ikke kunne byttes når et kjøretøy befinner seg på depotet. Dette hadde vært ønskelig da det kunne ført til mer optimale løsninger i flere tilfeller, for eksempel kunne en løsning der en ansatt var tilegnet en bil alene halve dagen for så å sitte på med en annen ansatt resten av dagen ført til bedre ressursutnyttelse. Dette vil ikke være mulig i gruppens implementasjon. Som følge av denne avgrensningen ønsket gruppen også å endre datasettet slik at en ansatt ikke har lister av skift over flere dager, men isteden representeres som unike objekter for hvert skift. Dette ville medført at ansatte kunne tilegnes forskjellige biler i forskjellige skift.

Avgrensning 3: Det var ikke tid til å implementere funksjonalitet for å sende inn et endret datasett som definert i 3.3 High-Level Use-Case beskrivelser. Dette ville for eksempel vært ønskelig dersom en ansatt ble syk i løpet av dagen og derfor ikke kunne gjennomføre sin rute. En slik funksjonalitet tror gruppen kunne vært implementert ganske enkelt dersom det hadde vært tid til dette fordi Optaplanner støtter muligheten til å fortsette optimalisering av ett datasett med endrede kriterier (Optaplanner Team, 2016, s. 295).

Bruk av distanse og tid

Algoritmens første fase som fordeler oppdrag på depoter benytter avstanden mellom hvert depot og alle oppdrag som beslutningsgrunnlag. I rutefordelingsfasen er det derimot nødvendig å benytte reisetiden mellom hvert oppdrag til alle andre oppdrag, pluss reisetiden mellom depotet det optimaliseres innenfor og alle oppdrag for beregning av ankomsttider. Avstand i meter benyttes altså

utelukkende i første fase, mens kun reisetid i millisekunder benyttes i andre fase. Dette medfører at man ikke enkelt kan finne data om total kjørelengde for de ferdig optimaliserte løsningsplanene den andre fasen resulterer i, men kun total reisetid. Total kjørelengde er en interessant verdi som de fleste bedrifter ville vært interessert i. Metodenavn relatert til henting av disse verdiene skaper også forvirring da de typisk er kalt getDistance uavhengig av om det er reisetiden eller avstanden som hentes ut. I retrospekt hadde det vært bedre å lagre både reisetid og distanse i et felles objekt. Dette hadde ikke hatt stor innflytelse på kalkuleringer av ruter mellom oppdrag og depoter fordi det er operasjonen å finne frem til ruten som er tidkrevende, ikke å hente ut dataen for ruten når den først er funnet. Det ville også hatt en nokså liten innflytelse på minnebruken for programmet. En klasse TripData kunne for eksempel lagre både reisetid og distanse. Hashtabellen som benyttes i Location-klassen kunne så mappe Location-objekter til slike TripData-objekter. På denne måten kunne også total kjørelengde enkelt blitt regnet ut.

Valg av faseinndeling

Algoritmen ble designet og implementert med en faseinndeling i de to fasene depotfordeling og rutefordeling. Denne faseinndelingen ble bestemt tidlig i utviklingsfasen av to årsaker. Den første årsaken var at det tillot gruppen å dele problemet inn i mindre delproblemer som så kunne løses separat. Dette forenklet læringsprosessen av Optaplanner rammeverket betydelig. Den andre grunnen bak dette valget var at gruppen fryktet at dersom hele problemet skulle løses i en fase, ville søkeområdet blitt for stort og løsningen skalere dårlig. Som ett eksempel som viser denne utfordringen så gruppen på tiden det tar å beregne avstanden mellom de forskjellige lokasjonene som må beregnes. Antall beregninger som må utføres er gitt ved $n * (n - 1)$, der n er det totale antall noder. Disse distansene må regnes ut av routing-tjenesten ved hjelp av kartdata. Når n øker, øker også tiden utregningen tar kvadratisk. Ved å gjøre en forenkling og først dele opp nodene mellom depotene, vil utregningsmengden reduseres til $n * d$, der d er antall depoter. Da det i de fleste tilfeller vil være langt færre depoter enn noder vil dette redusere tiden utregningen av distanser tar drastisk.

Tabell 9 viser en oversikt over tidsbruken utregning av distanse mellom nodepar tar dersom antall noder øker på forskjellige oppsett.

TABELL 9: TIDSBRUK VED KALKULASJON AV DISTANSER MELLOM VARIERENDE ANTALL NODER

Prosessor	*100 noder*	*300 noder*	*650 noder*
i5-5200u 2.2GHz	11.97 sek	106.89 sek	508.29 sek
i7-4700MQ 2.4 GHz	6.93 sek	60.60 sek	292.53 sek
i7-4790K 4.0 GHz	5.03 sek	45.79 sek	220.07 sek

Slik gruppen så det var dette bare det første og mest åpenbare eksempelet på at søkeområdet måtte reduseres med en faseinndeling. Den endelige løsningens søkeområde vill også økes for hvert oppdrag som skal utføres og hver ansatte som må tilegnes en bil.

Utover i utviklingsperioden tilegnet gruppen seg informasjon som pekte mot at en slik faseinndeling kanskje ikke var nødvendig. Lastingen av kartdata ble for eksempel optimalisert betraktelig ved å benytte seg av tråder. I tabell 10 vises en oversikt over tidsbruken utregningen av distanser tar med trådberegning.

TABELL 10: TIDSBRUK VED TRÅDKALKULASJON AV DISTANSER MELLOM VARIERENDE ANTALL NODER

Prosessor	*100 noder*	*300 noder*	*650 noder*
i5-5200u 2.2 GHz	6.18 sek	39.82 sek	179.41 sek
i7-4700MQ 2.4 GHz	2.27 sek	14.07 sek	56.76 sek
i7-4790K 4.0 GHz	2.35 sek	9.27 sek	39.99 sek

Ved en faseinndeling var det også nødvendig å gjøre flere forenklinger som fører til sub-optimale løsninger. Det sjekkes for eksempel kun at en ansatt på depot har kompetanse til å gjennomføre ett oppdrag før det tilegnes, ikke at den ansatte er tilgjengelig. Ved faseinndelingen mister også algoritmen noe av muligheten til å oppdage de tilfellene der ressursbruken ikke er optimal fordi fordelingen av oppdrag ikke tar hensyn til nærliggende oppdrag som uansett må besøkes slik som nevnt i problemdefinisjonen.

Algoritmen kan endres slik at depotfordelingsfasen og rutefordelingsfasen løses i en fase. Dette kan gjøres ved å inkludere en regel som sier at ansatte kun kan tilegnes biler som er på samme depot som den ansatte. Dette kombinert med mindre endringer i opprettelsen av data objekter er alt som skal til. Det ble allikevel bestemt å ikke implementere denne løsningen fullt ut da utviklingsperioden var utløpt og det ikke var ønskelig å fullstendig fjerne muligheten for faseinndeling da det kan være nødvendig ved veldig store datasett.

Inputdatasettet

Når problemdatasettet importeres inn til algoritmen og serialiseres til java-objekter, inneholder hvert depot-objekt samlinger av alle ansatte og kjøretøy tilknyttet depotet. Dette er det i seg selv ikke noe problem med, men da systemet benytter Gson for å serialisere dataen direkte til Java-klasser blir denne dataen værende med samme struktur i hvert depot-objekt under optimaliseringen. Fordi Optaplanner i tillegg krever at planentiteter defineres i egne samlinger i løsningsplan-klassene, fører dette til mye redundant data og dårlig bruk av minne. For eksempel vil en instans av et løsningsplan-objekt av RouteDistributionPhaseSolution-klassen inneholde to lister bestående av kjøretøy og ansatte. Objektet vil også ha en referanse til depotet som det skal optimaliseres innenfor. Dette depotet inneholder selv data om kjøretøy og ansatte knyttet til seg. Problemet kan løses ved å endre datasettet som mottas av algoritmen og klassene denne dataen serialiseres til, eller skrive en mer tilpasset versjon av JsonProblemImporter som leser dataen fra JSON-filen mer systematisk. I begge tilfeller ville data om ansatte og kjøretøy fjernes fra depotklassedefinisjonen. Hvert ansatt- og kjøretøy-objekt ville i stedet inneholdt en referanse til sitt tilknyttede depot.

8.2 Måloppnåelse

I begynnelsen av prosjektet ble det utarbeidet et sett med resultat-, effekt og læringsmål i kapittel 1.4 Formål. I dette delkapittelet vil gruppen se på disse målene og vurdere i hvilken grad de er nådd.

Resultatmål

Resultatmålene gruppen satte seg var i stor grad knyttet opp til oppgavebeskrivelsen gitt fra oppdragsgiver. Målene inkluderte at algoritmen inneholdt en modul for innlesning av et problemdatasett, at algoritmen kunne løse slike datasett og generere tilnærmede optimale kjøreruter, at en modul eksisterte for å eksporter denne dataen på en fornuftig måte og at denne

eksporterte dataen kunne vises frem på en hensiktsmessig måte. Med unntak av at algoritmen skulle kunne generere tilnærmede optimale løsninger, mener gruppen at alle resultatmålene som ble satt også ble nådd. Målet relatert til tilnærmede optimale løsninger er vanskeligere å avgjøre fordi svaret på hva en slik løsning innebærer vil variere fra bedrift til bedrift. Dette er tatt hensyn til ved å implementere vekter som kan justere kriteriene det prioriteres etter. Modulene relatert til innlesning av problemdatasett og eksportering av løsningsdata tilsvarer henholdsvis klassene JsonProblemImporter og JsonObjectExporter som definert i 6.1 Algoritme, Klassebeskrivelser. Presentasjonsapplikasjonen fungerer som et bevis på at målet relatert til fremvisningen av denne dataen lar seg gjøre.

Effektmål

Gruppen har hatt stor fokus på dokumentasjon av kildekode og arbeidsprosess for å oppfylle effektmålet om at ETC så enkelt som mulig skal kunne ta over og benytte kodebasen. Gruppen mener derfor at dette effektmålet kan ansees som oppfylt. Effektmålene for selve algoritmen har derimot i løpet av prosjektets gjennomføring hatt varierende grad av oppnåelse. Flere av effektmålene ble tidlig i utviklingsfasen luket bort fordi gruppen så det nødvendig å begrense oppgavens omfang. Dette gjorde at fokuset på å redusere overflødige arbeidstimer og tilgjengelige biler ble fjernet. Som nevnt tidligere i dette kapittelet fant derimot gruppen, i samråd med oppdragsgiver, allikevel en metode for å oppfylle disse målene i slutten av utviklingsfasen. Effektmålene for algoritmen har også vist seg vanskelig å verifisere. Det er flere grunner til dette. Prosjektet avsluttes før løsningen blir tatt i bruk av reelle kunder. Kvalitative effektmål slik som å redusere stress hos de ansatte lar seg derfor ikke måle. Det lot seg heller ikke gjøre å bruke reelle data for å sammenligne disse med en løsning fra hjemmetjenesten. Gruppen ble derfor nødt til å basere seg utelukkende på genererte data. Dette medfører at de kvantitative effektmålene som å redusere svinn av ressurser, overflødige arbeidstimer og antall biler blir upålitelige. Gruppen har ikke inkludert totalt antall kilometer kjørt eller kjøretid i presentasjonsapplikasjonen. Dette medfører at gruppen kun visuelt kan bekrefte hvorvidt løsningen reduserer kjøreavstanden.

Læringsmål

Gruppen definerte i starten av prosjektet læringsmål vedrørende områder hvor de håpet å få ny eller fordypet erfaring. De ønsket å tilegne seg erfaring om god arbeidsmetodikk sammen med en reell oppdragsgiver på ett større prosjekt, ny kunnskap om utviklingen av optimaliseringsalgoritmer og bruk av kartdata.

Gruppen føler at de har fått god erfaring med prosjektstyring og arbeidsmetodikk da de startet prosjektet med å utarbeide en tidsplan i form av et Gantt-skjema og en arbeidsmetodikk basert på en forenkling av RUP med innslag av elementer fra Scrum. Metodikken som ble utviklet og erfaringene gruppens medlemmer tilegnet seg, har vært en medvirkende faktor til at prosjektet har blitt godt dokumentert og nådd et godt sluttresultat til tross for at flere avgrensninger ble gjort underveis. Definisjonene utarbeidet for faseinndelingen i RUP medførte blant annet at gruppen prioriterte lengre Inception- og Elaborationfaser før de begynte å implementere en løsning. Det viste seg å være heldig da gruppen i løpet av Elaboration-fasen oppdaget at de kunne oppfylle arbeidsgivers ønsker bedre dersom de benyttet seg av ett optimaliseringsrammeverk for å lage algoritmen fremfor å utvikle den fra bunnen av. Det var av denne grunn lite av arbeidet frem til dette punktet som måtte endres eller forkastes som følge av denne beslutningen.

Tett samarbeid med en oppdragsgiver og andre parter har også vært sentralt for læringsutbytte til gruppen. Det å arbeide med oppdragsgiver for å utarbeide en reell kravspesifikasjon har vært svært lærerikt. Å forsøke å realisere denne kravspesifikasjonen blant annet gjennom møte med representant fra hjemmetjenesten i Gjøvik for å definere tilgjengelig data i deres systemer har også bydd på interessante problemstillinger. Spesielt interessant har det vært å utarbeide hvilken data som ikke eksisterer i hjemmetjenestens systemer, slik som kompatibiliteten til ansatte og pasienter, og derfor definere disse. Gruppen tror at dette har gitt dem viktig erfaring som er direkte anvendbar i en fremtidig arbeidssituasjon.

Samarbeidet med ETC og hjemmetjenesten har også gitt gruppens medlemmer erfaring med viktigheten av kommunikasjon i ett slikt prosjekt. Det finnes flere eksempler hvor kommunikasjon, eller mangel på kommunikasjon har vært et sentralt tema. Et eksempel på manglende kommunikasjon mellom gruppens medlemmer og ETC var da gruppen så seg nødt til å avgrense oppgaven til kun å håndtere fordeling av oppdrag. Gruppen trodde de gjorde prioriteringen av oppdragsfordeling fremfor ressursreduksjon i lag med ETC, men det viste seg i etterkant at ETC hadde foretrukket at gruppen valgte ressursredusering isteden.

Det eneste området hvor gruppen føler at de har tilegnet seg mindre erfaring enn forventet er i bruken av kartdata. Kartdata viste seg ikke å være en like stor del av utviklingen av algoritmen som først antatt, da dette kun benyttes for å finne avstander og reisetid mellom to sett med bredde- og lengdegrader.

8.3 Videre arbeid

I dette delkapittelet vil ulike måter prosjektet kan videreutvikles diskuteres. Utvidelsene vil sees i sammenheng med den implementerte løsning slik at det også påpekes hvilke endringer av løsningen som må til for å kunne gjennomføre en slik videreutvikling.

Flere typer bedrifter som for eksempel restauranter med leveringsmuligheter baserer seg på at bestillinger kommer inn med høy frekvens. Slike bedrifter vil ikke finne nytte i å optimalisere sine ruter ved begynnelsen av en arbeidsdag fordi de ikke har fullstendig data tilgjengelig ved dette tidspunktet. Hver gang en endring i data registreres som for eksempel at en ny bestilling ankommer systemet, sendes et oppdatert datasett som skal løses til algoritmen. For å kunne implementere en slik modul i algoritmen ville det vært nødvendig å tilegne hvert oppdrag med et status-felt som indikerer om bestillingen er klar til å utføres eller påbegynt. Det ville også vært nødvendig å ha data om hvert kjøretøys koordinater når et nytt datasett innsendes da det ikke lengre kan antas at alle kjøretøy befinner seg på depotet ved optimaliseringens starttidspunkt. Kravene til algoritmen endrer seg også. Slike bedrifter trenger ofte rask tilbakemelding, og den dynamiske planleggingen bør fungere slik at resultater, med jevne intervaller eller etter en fastsatt forbedring av poengsummen for løsningen, sendes tilbake til deres systemer. Ved løsning av et endret problemdatasett burde også algoritmen bruke den beste løsningen for det forrige datasettet som utgangspunkt for optimaliseringen. Dette vil trolig redusere tiden det tar å komme frem til bedre løsninger da mindre endringer av datasettet har tilnærmede optimale løsninger i nærheten av de tidligere funnede løsningene.

Det blir i gruppens system ikke gjort noen verifisering av datasettet som kommer inn til algoritmen. Dersom en verdi som forventes ikke finnes i dette datasettet, importeres verdien som en null-verdi inn i datastrukturen benyttet av algoritmen. Det hadde vært ønskelig med en mer avansert versjon av denne modulen som avbryter dersom verdier som må være tilstede ikke finnes og gir

tilbakemelding om hva som gikk galt, hvor feilen oppstod og hvordan den kan rettes opp. Gruppen ser for seg to muligheter for at en slik utvidelse kan utføres. Funksjonaliteten kan bygges inn i klassen JsonProblemImporter, ved at innlesningen av JSON-datasettet ikke gjøres ved bruk av Gson, men foregår sekvensielt samtidig som verdier verifiseres. Denne fremgangsmåten gjør det lett å gi tilbakemeldinger skreddersydd til det innsendte datasettet ved at linjenummer og feilmeldinger kan samles opp og returneres ved feil. Den andre muligheten er å utvikle verifikasjonen som en egen selvstendig modul slik at det er det ferdig importerte objektet som sjekkes for gyldige verdier. Denne fremgangsmåten gjør systemet mer modulært ved at modulen ikke trenger å tenke på hvordan dette objektet ble opprettet. En fordel med dette er at grensesnittet inn mot algoritmen enklere kan endres ved at datasettet for eksempel kan komme inn på XML-format uten at dette har noe å si for verifiseringen. Ulempen er at feilmeldinger ikke like enkelt kan knyttes opp til konkrete plasseringer i et innsendt datasett.

Benchmark-testingen som blir gjennomført av gruppen er også kun en begynnelse på det som kan og burde testes og optimaliseres. Videre arbeid som kan benchmarkes inkluderer blant annet å spesifisere flere forskjellige metaheuristiske søkemetoder, slik som Simulated Annealing, som kan benyttes i Solveren. Det er også slik at noen optimaliseringsalgoritmen fungerer mer effektivt dersom det kan estimeres hvilke planentiteter som er vanskeligere å plassere i løsningen. For eksempel vil ansatte med få kompetansetyper være vanskeligere å plassere fordi de kan utføre et mindretall av oppdragene. Optaplanner definerer et interface - Comparator hvor man kan sammenligne planentiteter og avgjøre hvilke som burde plasseres først. Dersom dette interfacet implementeres utvides benchmarkingsmulighetene betydelig slik at ulike construction heuristikker og metaheuristikker kan sammenlignes og optimaliseres ved testing.

8.4 Evaluering av gruppens arbeid

Prosjektet ble først organisert ved å dele det inn i RUP-fasene som diskutert i 2 Arbeidsmetode og utviklingsprosess. De ulike fasene ble videre oppdelt gjennom å definere oppgaver som skulle gjennomføres i de ulike fasene i et Gantt-skjema. Disse oppgavene ble deretter utgangspunkt for mindre deloppgaver som gruppen delte mellom seg. Deloppgavene ble hovedsakelig holdt styr på gjennom verktøyet Trello. Gruppen har imidlertid utført det meste av prosjektet sammen på skolen. Dette har gjort behovet for et verktøy som Trello mindre fremtredende og har gjort at oppgaver til dels har blitt fordelt muntlig mellom gruppemedlemmene. Gruppen har forsøkt å holde en jevn arbeidsfordeling gjennom hele prosjektperioden med uker bestående av 30 arbeidstimer. Dette har for det meste blitt fulgt, med unntak av perioder tett opptil milepæler da ekstra arbeidstimer har måtte blitt innlagt for å nå disse i tide. Dette var spesielt tilfelle ved milepælene Initial Operation Capability der all funksjonalitet skulle være implementert og Product Release ved innlevering av prosjektet.

Gruppen har også forsøkt å fordele arbeidet jevnt mellom de to gruppemedlemmene. De har i stor grad vært delaktige i utviklingen av de fleste av algoritmens ulike moduler. Rollene definert i 1.9 Ansvarsforhold og Roller har blitt fulgt ved at Helge har fungert som kontaktperson og gruppeleder for gruppen, mens Sigurd har hatt ansvar for oppgavens nettside og for å ta notater under møter. I tillegg til de definerte rollene har Sigurd hatt hovedansvaret for å utvikle presentasjonsapplikasjonen, mens Helge har vært hovedansvarlig for å utføre Benchmark-tester av algoritmens resultater. Det tette samarbeide under utviklingen av algoritmen har gjort at gruppen har kunnet bistå hverandre ikke bare med kunnskap, men også motivasjon.

I stor grad mener gruppen at prosjektsamarbeidet har fungert godt. Det har ikke vært større konflikter som følge av uenigheter, ujevn arbeidsfordeling eller andre faktorer. Valg har blitt diskutert på en saklig måte og har i de fleste tilfeller ført til enighet innad i gruppen. I et fåtall av tilfellene ble problemet presentert sammen med hvert av gruppemedlemmenes løsningsforslag til oppdragsgiver og veileder der deres mening ble den avgjørende faktoren for valget som ble gjort.

9 Konklusjon

Å jobbe med en oppgave på denne størrelsen har gjort at gruppen har tilegnet seg verdifull erfaring som vil komme til stor nytte i videre studier eller i arbeidslivet fremover. Oppgaven som gikk ut på å lage en fordelingsalgoritme for oppdragsbasert transportnæring ble utlyst av ETC og har vist seg å være en svært omfattende oppgave som har gitt gruppen mye erfaring innen viktige områder slik som algoritmisk design og bruk av store omfattende rammeverk. En vesentlig del av oppgaven har også vært å identifisere hva som var mulig å oppnå av oppdragsgivers ønsker i løpet av den begrensede tidsperioden gruppen hadde tilgjengelig. Dette førte blant annet til ett skifte i gruppens oppgavefokus fra å utvikle en slik algoritme fra bunnen av til å ta i bruk rammeverket Optaplanner for å oppfylle så mange som mulig av disse ønskene. Skiftet har ført til at gruppen har fått kjennskap til det fantastiske Open Source miljøet som eksisterer og sitter igjen med mye positiv erfaring fra denne prosessen. Optaplanner har også gitt gruppen god erfaring med å sette seg inn i og ta i bruk ett stort og omfattende rammeverk for å løse spesifikke utviklingsoppgaver.

Ved hjelp av Optaplanner har gruppen vært i stand til å utvikle en fordelingsalgoritme som løser store deler av oppdragsgivers ønsker. Algoritmen er i stand til å ta imot ett datasett bestående av avdelings-, ansatt-, kjøretøy- og oppdragsinformasjon sammen med en definisjon av hva som skal prioriteres under optimaliseringen. Algoritmen kjører så igjennom to faser hvor oppdragene først blir fordelt på utkjøringspunkter før de deretter fordeler ruter med spesifikke ansatte til å utføre disse oppdragene etter spesifikke kriterier som kan velges av brukeren.

Det er allikevel flere områder der gruppen ser at algoritmen har muligheter for utvidelse eller kan forbedres. Dette gjelder blant annet muligheten til å utvide algoritmen til å støtte dynamisk planlegging og forbedring av visse regler som er implementert. Gruppen ser også ett behov for en lengre periode med kvalitetssikring og en mer omfattende bruk av Benchmarking for å virkelig oppnå de beste løsningene.

Alt i alt er gruppen godt fornøyd med sluttresultatet og håper at ETC vil ha stor nytte av det arbeidet som er gjort enten ved videreutvikling av deres kode eller som ett læringsverktøy for bruk av Optaplanner.

10 Referanser

Eppstein, D. (1996). *ICS 161: Design and Analysis of Algorithms*. Hentet fra University of California, Irvine: https://www.ics.uci.edu/~eppstein/161/960312.html

Google Maps for Work. (2016). Hentet fra Google: https://www.google.com/intx/en/work/mapsearth/

Green, B., & Seshadri, S. (2013). *AngularJS.* Sebastopol, Califorina: O'Reily Media. Hentet fra https://books.google.no/books?hl=no&lr=&id=eNExy_X1YYcC&oi=fnd&pg=PR2&dq=angularJS&ots=wz5aDYOfS7&sig=caGZGb0XtJfYc6bl42QisWtNkhI&redir_esc=y#v=onepage&q=angularJS&f=false

Here Company. (2016). *Here for Enterprise*. Hentet fra Here: https://company.here.com/enterprise/

JBoss Drools Team. (2016). *Drools Documentation.* Hentet fra JBoss: http://docs.jboss.org/drools/release/6.4.0.Final/drools-docs/pdf/drools-docs.pdf

Kartverket. (u.d.). *Om kartverket*. Hentet fra kartverket nettsted: http://www.kartverket.no/Om-Kartverket/

Luke, S. (2015). *Essentials of Metaheuristics.* Hentet fra George Mason University: https://cs.gmu.edu/~sean/book/metaheuristics/Essentials.pdf

Magic Web Solutions. (2013). *The benefits of web-based applications*. Hentet fra Magic Web Solutions: http://www.magicwebsolutions.co.uk/blog/the-benefits-of-web-based-applications.htm

Marshall, D. (1996). *Heuristic Search.* Hentet fra Cardiff School of Computer Science: http://www.cs.cf.ac.uk/Dave/AI2/node23.html

Open Street Map Wiki. (2016). *Kartverket import*. Hentet fra Open Street Map Wiki: http://wiki.openstreetmap.org/wiki/No:Kartverket_import

Open Street Map Wiki. (2016). *Using OpenStreetMap*. Hentet fra Open Street Map Wiki: http://wiki.openstreetmap.org/wiki/Using_OpenStreetMap

Optaplanner. (2016). *What is Optaplanner?* Hentet fra Optaplanner nettsted: http://www.optaplanner.org/

Optaplanner Team. (2016). *OptaPlanner User Guide.* Hentet fra JBoss: http://docs.jboss.org/optaplanner/release/6.4.0.Final/optaplanner-docs/pdf/optaplanner-docs.pdf

Personopplysningsloven. (2000). Hentet fra Lovdata: https://lovdata.no/dokument/NL/lov/2000-04-14-31

Poole, D., & Macworth, A. (2010). *Artificial Intelligence*. Hentet fra Constraint Satisfaction Problems: http://artint.info/html/ArtInt_76.html

Rational. (2005). *Rational Unified Process: Best Practices for Software Development Teams.* Hentet fra IBM nettsted:

https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

Rational Software Corporation. (2002). *Rational Unified Process: Artifacts*. Hentet fra University of Houston-Clear Lake: http://sce.uhcl.edu/helm/rationalunifiedprocess/process/artifact/ovu_arts.htm

Red Hat. (2016). *Overview*. Hentet fra Drools: http://www.drools.org/

Shuja, A. K., & Krebs, J. (2008). *Welcome to the IBM Rational Unified Process and Certification*. Hentet fra Informit: http://www.informit.com/articles/article.aspx?p=1155863&seqNum=2

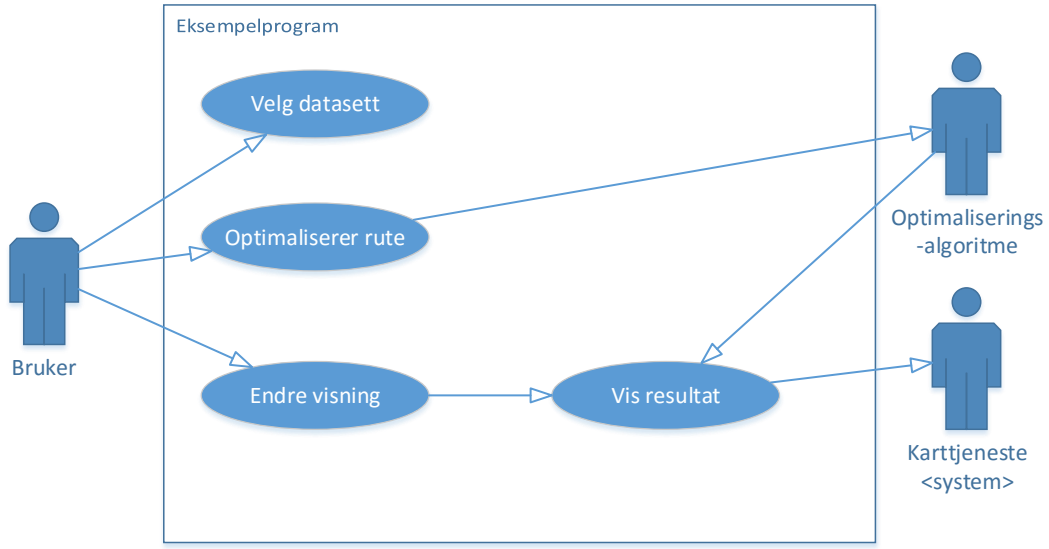
Sommerville, I. (2011). *Software Engineering 9th Edition.* Boston: Pearson Education.

Staffordshire University. (2008). *RUP summary*. Hentet fra Staffordshire University website: http://www.projects.staffs.ac.uk/suniwe/project/rup.html

Terrence, W. K. (2011). *Quantified Weighted Constraint Satisfaction.* Hentet fra Terrece W.K.MAK: http://www.tmak.info/papers/MasterThesis.pdf

11 Vedlegg

Vedlegg A - Utdrag fra kravspek, revisjon 1



Figur 1: Use-case diagram for eksempelprogram

High-Level Use-Case beskrivelser

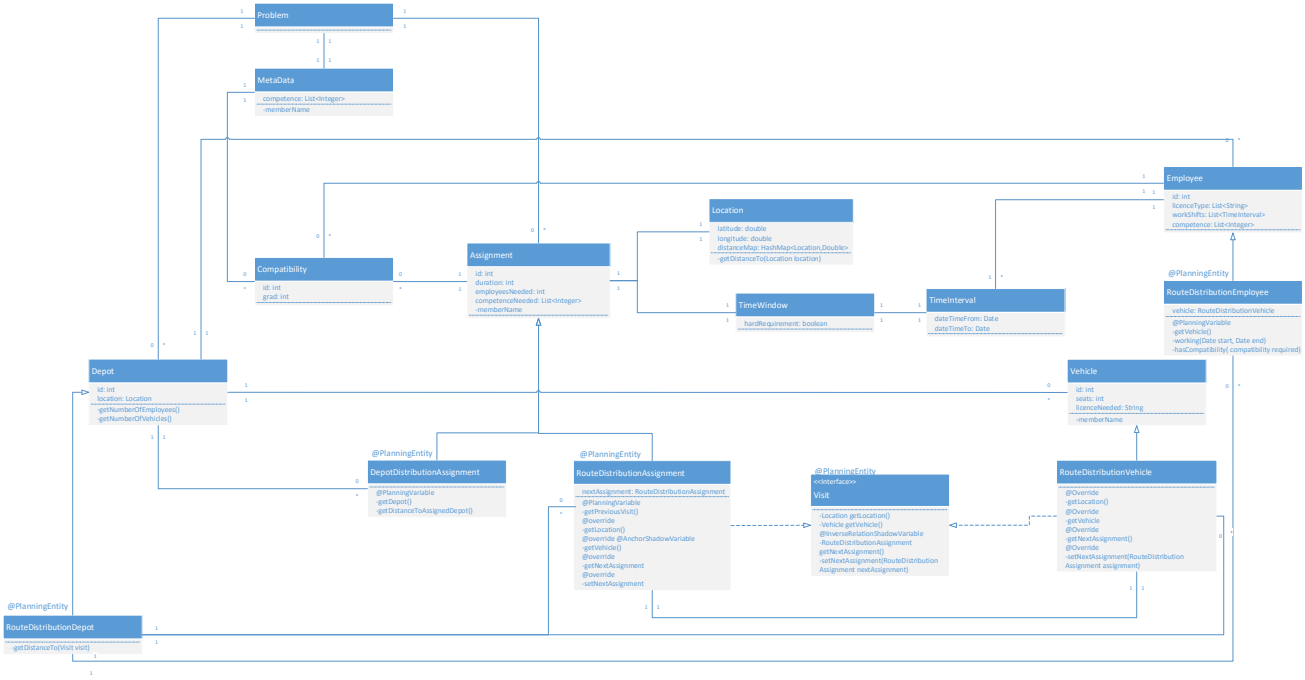
Use Case	*Velg datasett.*
Aktør	Bruker.
Mål	Velge et datasett som skal optimaliseres.
Beskrivelse	En bruker velger et datasett som skal optimaliseres fra en liste av forhåndsdefinerte sett.

Use Case	*Optimaliser rute.*
Aktør	Bruker, optimaliseringsalgoritme.
Mål	Få tilbake og vise frem en tilnærmet optimal rute.
Beskrivelse	En bruker skal kunne be om en optimalisert rute fra ett datasett og få dette returnert til systemet.

Use Case	*Endre visning.*
Aktør	Bruker.
Mål	Endre måten dataen fra algoritmen vises.
Beskrivelse	Brukeren skal kunne endre måten den tilnærmet optimaliserte dataen vises på.

Use Case	*Vis resultat.*
Aktør	Bruker, optimaliseringsalgoritme, karttjeneste
Mål	Vise frem resultatet fra optimaliseringsalgoritmen.
Beskrivelse	Når programmet mottar en tilnærmet optimal løsning skal denne vises frem på brukers skjerm.

Vedlegg B – Klassediagram


Problem
MetaData
competence: List<Integer>
-memberName
Employee
id: int
licenceType: List<String>
workShifts: List<TimeInterval>
competence: List<Integer>
Location
latitude: double
longitude: double
distanceMap: HashMap<Location,Double>
-getDistanceTo(Location location)
Compatibility
id: int
grad: int
Assignment
id: int
duration: int
employeesNeeded: int
competenceNeeded: List<Integer>
-memberName
TimeWindow
hardRequirement: boolean
TimeInterval
dateTimeFrom: Date
dateTimeTo: Date
@PlanningEntity
RouteDistributionEmployee
-vehicle: RouteDistributionVehicle
@PlanningVariable
-getVehicle()
-working(Date start, Date end)
-hasCompatibility(compatibility required)
Depot
id: int
location: Location
-getNumberOfEmployees()
-getNumberOfVehicles()
Vehicle
id: int
seats: int
licenceNeeded: String
-memberName
@PlanningEntity
DepotDistributionAssignment
@PlanningVariable
-getDepot()
-getDistanceToAssignedDepot()
@PlanningEntity
RouteDistributionAssignment
-nextAssignment: RouteDistributionAssignment
@PlanningVariable
-getPreviousVisit()
@override
-getLocation()
@override @AnchorShadowVariable
-getVehicle()
@override
-getNextAssignment
@override
-setNextAssignment
@PlanningEntity
<<interface>>
Visit
-Location getLocation()
-Vehicle getVehicle()
@InverseRelationShadowVariable
-RouteDistributionAssignment
getNextAssignment()
-setNextAssignment(RouteDistribution
Assignment nextAssignment)
RouteDistributionVehicle
@Override
-getLocation()
@Override
-getVehicle
@Override
-getNextAssignment()
@Override
-setNextAssignment(RouteDistribution
Assignment assignment)
@PlanningEntity
RouteDistributionDepot
-getDistanceTo(Visit visit)

Vedlegg C – Prosjektavtale

NTNU
Norges Teknisk-Naturvitenskapelige Universitet
NTNU i Gjøvik, Avd. Informatikk og Medieteknikk

PROSJEKTAVTALE

mellom NTNU v/Avd. Informatikk og Medieteknikk (NTNU/AIMT) (utdanningsinstitusjon), og

ETC, Electric Time Car AS

(oppdragsgiver), og

Helge Eriksen, Sigurd Molnes Harkjerr

(student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 08.01.2016 til 18.05.2016

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der AIMT yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra AIMT å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra Gjøvik/AIMT. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. AIMT står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av faglærer/veileder og sensor (intern og ekstern sensor). Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv hvis de har skriftlig karakter A, B eller C.

   Tilgjengeliggjøring i det åpen arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og dekan om de i løpet av prosjektet endrer syn på slik offentliggjøring.

   Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av AIMT til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU/AIMT og/eller studenter har interesser.

6. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

7. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

8. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i Fronter. I tillegg leveres et eksemplar til oppdragsgiver.

9. Denne avtalen utferdiges med et eksemplar til hver av partene. På vegne av AIMT er det dekan/prodekan som godkjenner avtalen.

10. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og AIMT som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten AIMT som partner.

11. Når NTNT/AIMT også opptrer som oppdragsgiver trer NTNU/AIMT inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

12. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene i mellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

13. Deltakende personer ved prosjektgjennomføringen:

NTNU/AIMTs veileder (navn): Ivar Farup

Oppdragsgivers kontaktperson (navn): Dag L Solhaug

Student(er) (signatur): Sigurd Holmes Torkjerr dato 19.01.2016

Helge Grimst. dato 19.01.2016

_____ dato _____

_____ dato _____

Oppdragsgiver (signatur): Dag L Solhaug dato 25/01-2016

Signert avtale leveres digitalt i Fronter
Godkjennes digitalt av AIMTs dekan

Om papirversjon med signatur er ønskelig, må papirversjon leveres til AIMT i tillegg.
Plass for evt sign:
AIMT Dekan/prodekan (signatur): _____ dato _____

Vedlegg D - Prosjektplan

PROSJEKTPLAN
FORDELINGSALGORITME

Helge Eriksen
Sigurd Molnes Harkjerr

1 MÅL OG RAMMER

1.1 Bakgrunn

ETC, Electiric Time Car AS er et IT-selskap sentrert i gjøvik. Selskapet ble etablert i 2003 og fokuserer på løsninger relatert til administrasjon av fellesbiler for bedrifter. Deres hovedprodukt, CarAdmin lar brukere holde daglig oppfølging av kjøretøy for en hel bedrifts bilpark. Daglig leder for ETC er Dag L. Solhaug.

I forbindelse med bacheloroppgavene som gjennomføres våren 2016, har ETC utlyst et oppdrag bestående av å utvikle en fordelingsalgoritme for tjenesteytende næringer. Et eksempel på en slik næring er hjemmetjenesten, der oppdrag fordeles på hjemmetjenestens ansatte innenfor en geografisk sone. Disse oppdragene spesifiserer type oppgave som skal utføres, lengden i tid oppdraget skal ta, antall personer som trengs for å utføre oppdraget og tidsrommet oppdraget skal utføres i.

Bachelorgruppen som skal utføre oppdraget utgjør to studenter fra NTNU i Gjøvik som studerer til dataingeniør. Studentene har liten erfaring utenom den erfaringen de har fått gjennom studier og har kun arbeidet på mindre prosjekter før dette. Prosjektstyring har tidligerer kun blitt gjort på teoretisk nivå. Studentene har fulgt standard utdanningsløp for Dataingeniører på Høgskolen i Gjøvik/ NTNU i Gjøvik. Femte semester valgte de begge å følge emnene programvareutvikling og mobil programvareutvikling, som tredje valgemne valgte Sigurd Matematikk 3 og Helge valgte Programvaresikkerhet.

1.2 Prosjektmål

1.2.1 Effektmål

Ved bruk av algoritmen er målet at følgene fordeler skal kunne være gjeldene. De faktiske målene vil variere fra bedrift til bedrift.

- Erstatte behovet for manuell ruteplanlegging.
- Redusere antall kilometer kjørt per bil per ansatt.
- Redusere svinn av ressurser i form av at ansatte må vente på tilgjengelig bil.
- Redusere antall biler som trengs for å utføre oppdragene.

- Redusere behovet for arbeidskraft ved å luke bort overflødige arbeidstimer brukt under transport.
- Øke samværtiden med kunde.
- Redusere stress hos de ansatte.

1.2.2 Resultatmål

Ved prosjektets slutt har gruppen følgende mål for systemet:

- Det skal finnes en enkel modul for å fore algoritmen med data.
- Algoritmen skal beregne en tilnærmet optimal oppdragsrute for ansatte.
- Systemet skal kunne eksportere sluttresultatet av utregningene på en fornuftig måte, slik at 3.parts programmer kan benytte seg av dette.
- Oppdragsrutene som er utregnet skal kunne vises frem på en hensiktsmessig måte.

1.2.3 Læringsmål

Ved å utføre prosjektet ønsker gruppen å få erfaring med eller tilegne seg kunnskap om:

- Å jobbe strukturert og systematisk ved hjelp av god prosjektstyring og arbeidsmetodikk på et større prosjekt.
- Å jobbe tett med en reell oppdragsgiver.
- Å arbeide med kartdata.
- Å utvikle optimaliseringsalgoritmer.
- Å utvikle prosjekter i C#.

1.3 Rammer

For å gjøre ruteberegninger skal algoritmen benytte seg av en ferdig utviklet routing- og karttjeneste. Disse tjenestene må kunne skiftes ut for å gi fleksibilitet dersom problemer med tjenestene skulle oppstå. Tjenestene må fungere uten tilgang til nett fordi algoritmen må være rask og flere spørringer over internett vil føre til for lang responstid.

Kildekoden til prosjekter kan heller ikke være åpent tilgjengelig i sin helhet, det kan derfor ikke benyttes kildekode under lisenser hvor dette er ett krav. Dersom en slik restriksjon finner seg gjeldende ved valg av kart- eller routingtjeneste har oppdragsgiver meddelt at de kan stille midler disponible for kjøp av nødvendig lisens.

Ved utviklingen av prosjektet gjelder prosjektavtalen mellom NTNU ved avdeling informatikk og medieteknikk, ETC og gruppens medlemmer. Avtalen fastsetter blant annet at studentene ikke kan publisere kode uten samtykke fra ETC, foruten den publiseringen som gjøres i regi av NTNU i Gjøvik.

2 OMFANG

2.1 Fagområde

Oppdragsbasert transportnæring opererer innenfor et geografisk område og kan være inndelt i ulike avdelinger. Hver avdeling kan ha ansvar for mindre delområder som dekkes av dem. Disse delområdene er ofte rigide slik at tjenester som skal leveres innenfor området alltid gjøres av det områdets avdeling, uavhengig av om ressurser kunne ha blitt brukt bedre ved å ta i bruk arbeidskraft fra andre avdelinger. To nabohus som ligger i skillet mellom slike delområder kunne derfor i værste fall bli dekket av to ulike avdelinger. Avdelingene har som regel varierende antall biler og ansatte tilgjengelige.

På toppen av dette kommer tilleggskrav som kan være gjeldene for ulike tjenesteytere. Hjemmetjenesten opererer foreksempel med vedtak som tildeles den hjelpetrengende. Vedtakene inkluderer hvilken type hjelp som skal utføres, en liste over konkrete oppgaver og kompetansekrav satt til den ansatte som utfører oppdraget. Vedtaket inneholder også et tidsrom oppgaven skal utføres i, hvor lenge hjemmebesøket skal vare og antall personer som trengs for å utføre oppgaven. Det blir også opprettet en primærkontakt for hver hjelpetrengende, der det er ønskelig at besøket forekommer av den ansatte som har blitt tildelt denne rollen.

Oppdragsbasert transportnæring, slik som hjemmesykepleien, benytter seg i stor grad av manuell ruteplanlegging i dag. Faktorer slik som beskrevet ovenfor har påvirkning på ruteplanleggingen. Dette medfører ofte mye arbeid fordi planleggingen fort kan bli kompleks og tidkrevende. Itillegg sitter ruteplanleggerne med informasjon som ikke registreres i datasystemet men som er avgjørende for planleggingen. Dette kan blant annet være hvilke ansatte som best håndterer vanskelige pasienter og derfor burde utføre oppdrag knyttet til dem.

2.2 Avgrensning

På grunn av algoritmens kompleksitet og den lille størrelsen på gruppen ble det avtalt med oppdragsgiver at gruppen skulle fokusere på utvikling av algoritmen.

Algoritmen skal bruke ett tredjeparts GIS-program for å finne veiruter. Dette valget ble tatt fordi det reduserer arbeidsbyrden dramatisk og fordi slik programvare allerede eksisterer og er i høy grad optimalisert.

Gruppen skal ikke arbeide med frontend løsninger som skal presenteres til klienter, men lage en arbeidsutgave for eget bruk.

2.3 Oppgavebeskrivelse

Det skal utvikles en automatisert algoritme som kalkulerer de beste arbeidsrutene ved å fordele oppdragene på en gunstig måte mellom de tilgjengelige kjøretøy, ansatte og institusjoner. Dette skal gjøres ved å følge ett sett av kriterier. Eksempler på slike kriterier kan være minst mulig kjørte kilometer per bil per ansatt eller mest mulig tid brukt hos kunden.

Algoritmen skal være en selvstendig modul som tar imot data på et bestemt format og eksporterer resultatet på et bestemt format. Input til algoritmen vil komme fra et tredjepartsprogram, som for eksempel et journal- eller bestillingssystem. Resultatet skal kunne eksporteres tilbake til et tredjepartsprogram gjennom et API.

Algoritmen skal typisk kjøres før en arbeidsdag, men det skal tas hensyn til unntakstilfeller som foreksempel at en ansatt blir syk i løpet av en dag. Algoritmen vil i et slikt tilfelle kjøres på nytt, men skal da ta hensyn til de oppdaterte dataene.

Det er viktig at algoritmen dekker krav satt av hjemmetjenesten i Gjøvik. En slik tjeneste inneholder et komplisert sett med krav. Ved å dekke Hjemmetjenestens behov antas det at algoritmen vil kunne generaliseres og brukes av en rekke andre transportbaserte tjenester, slik som post- eller matleveranse.

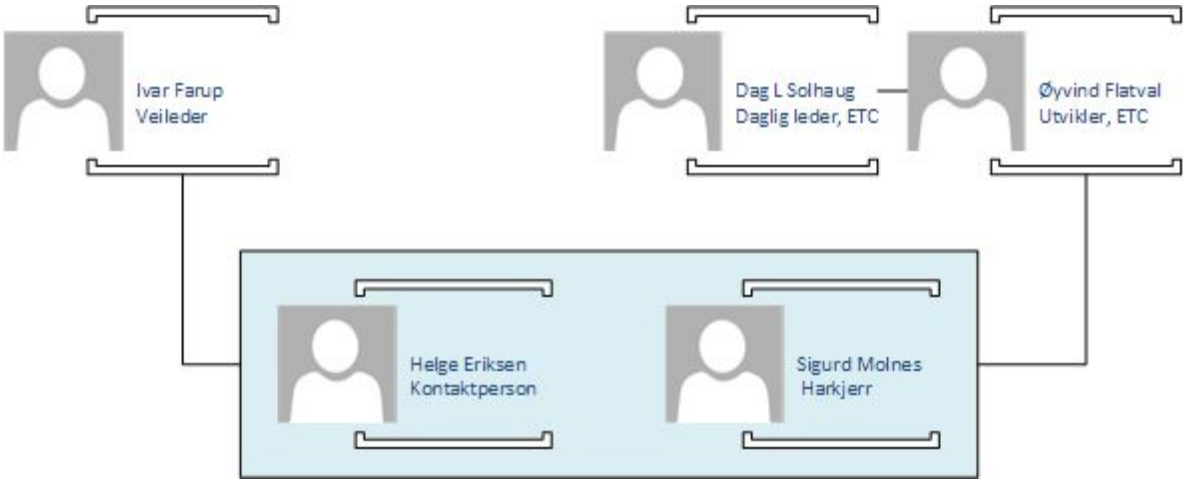
Kravene vil deles inn i to kategorier. Absolutte og relative krav. Absolutte krav vil brukes til å fullstendig utelukke løsninger, for eksempel vil et krav om kompetanse på ansatte som utfører oppdrag kunne ekskludere flere mulige løsninger. Relative krav vil gis en vekting avhengig av ønsket utfall for bedriften ved bruk av algoritmen. Bedriften skal selv kunne velge hvordan de selv vil vekte de relative kravene. Alle krav må også kunne utelukkes fullstendig dersom de ikke er relevante. Krav som kan kvantifiseres må kunne justeres. Et eksempel på et slikt krav vil være antall like oppgaver som maksimalt kan utføres etterhverandre av en ansatt.

Det skal velges en kart og routing tjeneste som oppfyller kravet om høy hastighet og være i stand til å benyttes uten internett. Tjenestene skal integreres i algoritmen på en slik måte at den enkelt kan byttes ut med konkurerende tjenester.

Det skal i tillegg utvikles et program som benytter seg av data fra algoritmen til å generere tidslinjer med oppdrag per bil, timelister per ansatt og rutevisning på kart.

3 PROSJEKTORGANISERING

3.1 Ansvarsforhold og roller



Figur 1: Rollediagram

Prosjektet er organisert med en flat ledelsesstruktur som skal fremme selvstendig arbeid og beslutningstagning fra hvert enkelt gruppemedlem. Helge Eriksen vil likevell fungere som Prosjektleder og kontaktperson for gruppen utad. Sigurd Molnes Harkjerr er ansvarlig for nettsiden og for å notere under møter.

Dag L. Solhaug og Øyvind Flatval representerer Electric Time Car AS som er oppdragsgiver, og vil bistå gruppen med teknisk og organisatorisk hjelp for å nå frem til ett best mulig sluttresultat. Dette vil de gjøre gjennom ukentlige møter og ved å være tilgjengelig for spørsmål over epost.

Hjemmetjenesten vil bistå med generell informasjon om deres arbeidsmetoder gjennom et innledende møte. Deretter vil Monica Kristiansen, som er ansatt ved hjemmetjenesten i Gjøvik, være tilgjengelige for å besvare spørsmål over telefon.

Ivar Farup er veileder fra NTNU i Gjøvik og vil bistå gruppen med prosjektstyring og teknisk ekspertise. Dette vil primært gjøres ved ukentlige møter, men han vil også være tilgjengelig på epost og for drop-inn besøk når timeplanen hans tillater det..

3.2 Rutiner og regler i gruppa

3.2.1 Grupperegler

1. Ved uenighet presenteres forslagene til veileder og/eller oppdragsgiver der de kommer med innspill. Dersom det fortsatt er uenighet tas det endelige valget av veileder og/eller oppdragsgiver.

2. Dersom et gruppemedlem ikke utfører avtalt arbeid til avtalt tid over en lengre periode, kontaktes veileder for tvistløsning.
3. Møter med oppdragsgiver skal minst foregå annenhver uke.
4. Alle gruppemedlemmer har rett til å signere på vegne av gruppen.
5. Gruppeleder skal fungere som kontaktperson for gruppen og har ansvaret for å svare på forespørsler fra veileder og oppdragsgiver på vegne av gruppen.
6. Rollen som gruppeleder kan overføres til andre gruppemedlemmer dersom gruppeleder ikke lengre ønsker å ta på seg ansvaret.

3.2.2 Rutiner

1. Gruppen legger opp til en fast arbeidsuke på 30 timer. Dette gjennomføres med en fast arbeidstid for gruppen mandag til torsdag 09.00 - 16.00. Utover dette dekkes de resterende to timene som egenstudie.
2. Fast møte med veileder tirsdager klokken 10.15.
3. Fast møte med oppdragsgiver tirsdager klokken 13.00.
4. Gruppemedlemmene skriver notater over utført arbeid fortløpende under hele prosjektperioden.
5. Møtereferat skal nedskrives for alle møter. Dette inkluderer møter med veileder, oppdragsgiver og representant fra hjemmetjenesten. Møtereferater sendes til de tilstedeværende. Alle deltagere har tilsvarsrett.

4 PLANLEGGING, OPPFØLGING OG RAPPORTERING

4.1 Valg av SU-modell/prosessrammeverk

4.1.1 Karakteristikk ved prosjektet som er styrende for valg av modell

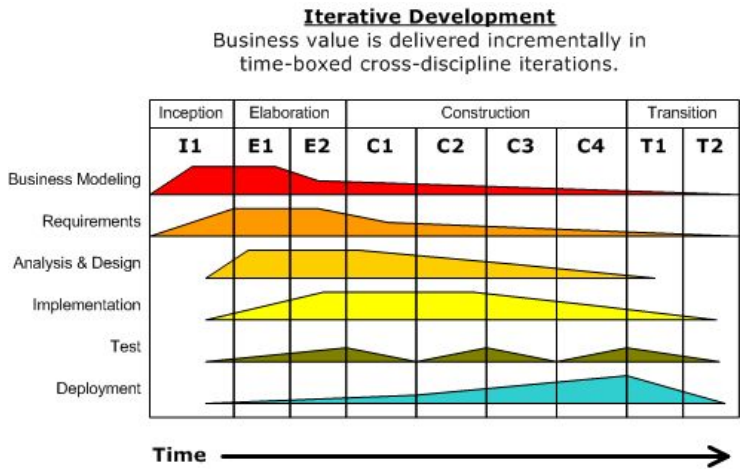
Følgende karakteristikker gjør seg gjeldende ved valg av systemutviklingsmodell som skal benyttes i prosjektet:

- Fast tidsfrist: bachelorprosjektet har en fastsatt dato for når oppgaven skal leveres.
- Få utviklere: prosjektet skal utføres av en liten gruppe bestående av kun to studenter.
- Ingen spisset kompetanse: utviklerne har det samme utgangspunktet og få spesialiseringen. Dette gjør at de lett kan ta over hverandres arbeid.
- Liten sannsynlighet for endring av kravspesifikasjoner: det forventes at hovedpunktene rundt kravspesifikasjonene vil være utarbeidet tidlig, og at disse med liten grad av sannsynlighet vil endre seg nevneverdig.
- Tilgang til kunde: utviklerne har ukentlige møter med oppdragsgiver.
- Høye dokumentasjonskrav: prosjektet skal utføres som del av en bacheloroppgave og dette legger høye krav til dokumentasjon av hele prosessen.

- Uerfarne utviklere: gruppens medlemmer har lite erfaring med større prosjekter i forhold til utvikling og prosjektstyring. En følge av dette kan være usikkerhet i estimering av tidsbruk.

Gruppen så originalt for seg at prosjektet skulle gjennomføres med Scrum da flere av disse karakteristikkene, som faste tidsfrister, få utviklere, ingen spisset kompetanse og tilgang til kunde, passer godt til denne metodikken. Scrum er også den utviklingsmodellen gruppens medlemmer er mest kjent med. Under utviklingen av gantt diagrammet og planleggingen av tidsbruk ble det derimot klart at det opplegget som var skissert hadde store likhetstrekk med RUP modellen. Det høye kravet til dokumentasjon er også en faktor som ikke er like godt egnet til Scrum metodikken, men oppfylles i høy grad av RUP via flere artifakter som sikrer god dokumentasjon og refleksjon rundt krav og design.

4.1.2 Beskrivelse av hvordan modellen skal anvendes i prosjektet



Figur 2: RUP-faser og fagdisipliner (tegning frigitt under Public Domain)

På grunn av gruppens og prosjektets størrelse vil en tynn variant av RUP bli benyttet. Faseindelingen inn i de 4 fasene Inception, Elaboration, Construction og Transition beholdes. Gruppen definerer RUP's seks fagdisiplinene på en slik måte at de avgrenses og finspisses, samtidig som de, etter gruppens mening, beholder hovedmomentene fra RUP's egne definisjoner:

- Business Modeling: kontakt med kunde.
- Requirements: forståelse av hva systemet skal gjøre.
- Analysis & Design: forståelse av hvordan systemet skal realiseres.
- Implementation: realisering av systemet.
- Test: kvalitetssikring, testing av enkeltstående komponenter og testing på tvers av algoritmens komponenter.
- Deployment: overføring av kodebase fra gruppen til oppdragsgiver for videreutvikling.

For å unngå overdreven dokumentasjonsfokus på bekostning av fremgang vil derimot ikke alle artifakter bli benyttet. Gruppen vil foreta en vurdering av de artifaktene gruppemedlemmene anser som best egnet til å styre og dokumentere arbeidet. Artifaktene

vil bli grunnlaget for tre hoveddokumenter: prosjektplan, kravspesifikasjonsdokument og designdokument. Dokumentene vil så danne grunnlaget for selve bachelorrapporten.

4.1.2.1 Forklaring av RUP-fasene brukt i prosjektet

Inception: I denne fasen vil prosjektet utredes. Dette vil gjøres gjennom utvikling av prosjektplan. Fagdisiplinene Business Modeling og Requirements vil ha høyt fokus. Dette vil gjøres ved å gjennomføre tidlige møter med oppdragsgiver for klargjøring av oppgaven. Life Cycle Objective er navnet på denne fasens milepæl og vil nås ved innlevering av prosjektplanen. På grunn av krav satt fra høgskolen om tidspunkt for innlevering av denne planen, vil milepælen ikke komme direkte etter avslutningen av Inception-fasen, men heller ut i Elaboration fasen. Dette er også gjort fordi gruppen ser det som sannsynlig at endringer til planen vil gjøres som følge av et senere møte med hjemmetjenesten.

Elaboration: Elaboration-fasen øker fokuset på prosjektets virkemåte og krav. Det er i denne fasen at kravspesifikasjonsdokumentet og designdokumentet for algoritmen tar form. Også her er Business Modeling og Requirements fagdisiplinene godt i bruk, blant annet gjennom møte med hjemmetjenesten for ytterligere utredelse av krav for oppgaven. Analysis & Design fagdisiplinen gjør seg også gjeldende under design av algoritmen. Fasen ender i milepælen Lifecycle Architecture, der de utviklede planene gjennomgås og sjekkes for eventuelle hull eller mangler som vil gjøre seg avgjørende i senere faser. Dersom ingen slike mangler oppdages, anses milepælen som nådd.

Construction: Selve utviklingen av algoritmen vil foregå i denne fasen. Gruppen vil fortsette å ha ukentlige møter med oppdragsgiver for å sikre fremdrift i prosjektet og korrigere kursen for ikke å havne i eventuelle fallgruver som kan oppstå. Business Modeling vil derfor ha et større fokus i denne fasen enn ved tradisjonell bruk av RUP. Algoritmedesignet vil fortsette å være en viktig faktor også i denne fasen. Det vil utvikles enhetstester parallelt med skriving av kode. Av disse grunnene, vil Business Modelling, Analysis & Design, Implementation og Testing være spesielt viktige fagdisipliner i denne fasen. Construction-fasen vil bære preg av Scrum-utviklingsmodellen ved at den inndeles i ukentlige sprints. Dette er gjort for å gjøre oss mer mottakelige for endringer som kan oppstå og for å sikre fremdrift da utviklingstiden er kort. Det vil også sikre at gruppen har noe å levere dersom algoritmen ikke blir fullstendig ferdigstilt. Som nevnt vil gruppen også ha ukentlige møter med oppdragsgiver. Oppdragsgiver vil allikevel ikke ha rollen som product owner som sett i Scrum, men vil kunne gi tips og ønsker avgjørende hvilke arbeidsoppgaver som skal ha fokus for ukens sprint. Construction-fasen ender i milepælen Initial Operation Capability der all funksjonalitet er ferdigutviklet med tilhørende testing.

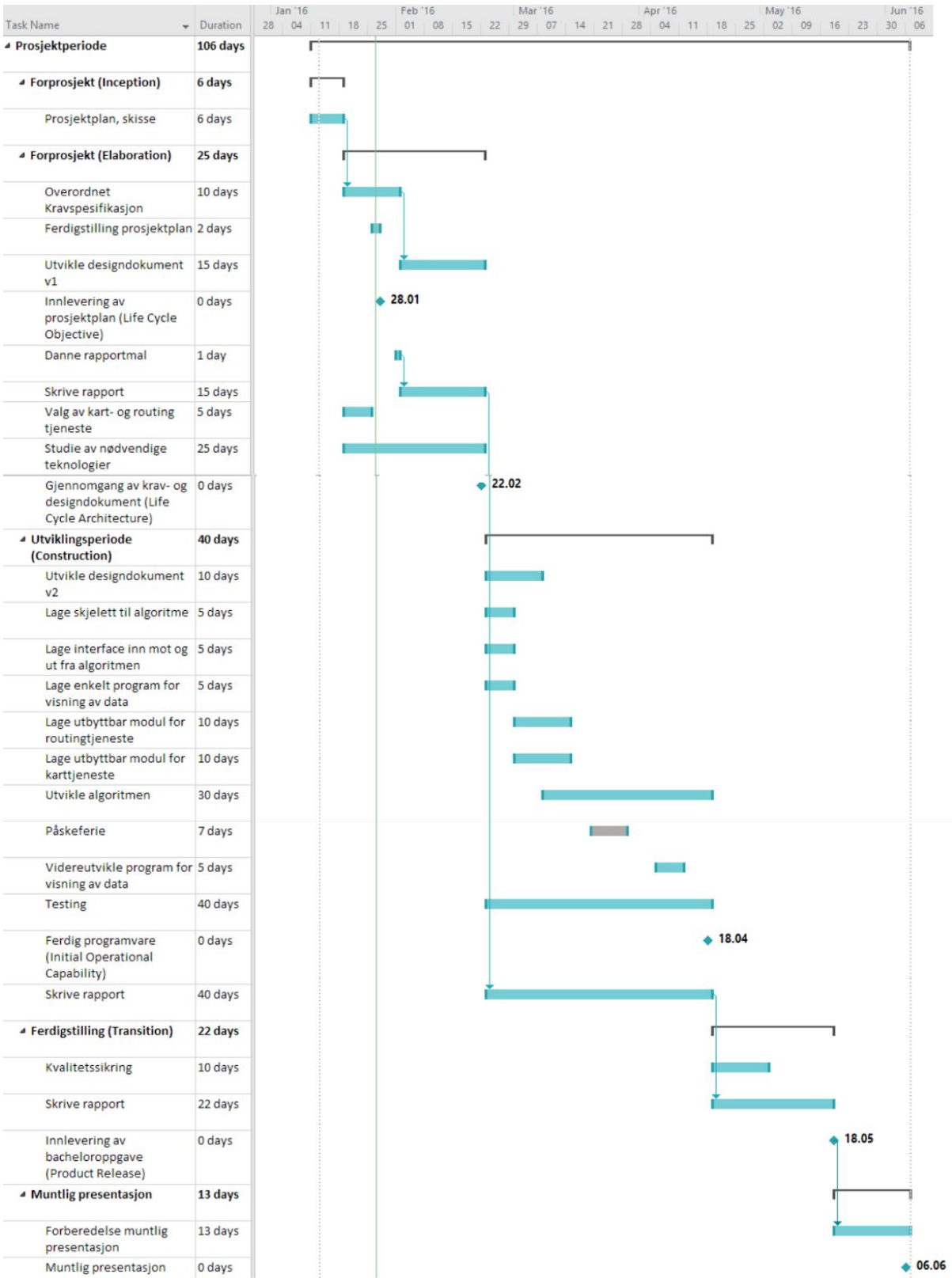
Transition: I denne fasen vil kvalitetssikring av algoritmen være fokuset i Testing fagdisiplinet. Algoritmen sammenlignes med resultatet fra hjemmesykepleiens arbeid for å avgjøre om den har akseptabel kvalitet. Fordi dette er en bacheloroppgave og sluttresultatet ikke skal direkte til forbruker vil Deployment fagdisiplinet få en liten rolle. Ferdigstilling av rapporten vil isteden være en viktig del av denne fasen og vil anses som Product Release milepælen i denne fasen.

4.1.2.2 Idealer i RUP

Vi vil også følge de seks idealene i RUP for å minimere risikoer og øke produktivitet.

- Utvikle iterativt: Gjennom sprintinndeling av construction-fasen for å minimere konsekvensene av endring av krav.
- Håndtere krav: Fokusere på brukere, i hovedsaklig hjemmetjenesten, og deres krav til algoritmen.
- Modularisere: Lage programvaren og algoritmen modulær slik at det kan utvikles i frittstående komponenter.
- Visuell modellering: Utvikle relevante diagrammer og modeller som foreksempel UML klasse diagram og use-case diagram.
- Kvalitetssikring: Ha testing i bakhodet under hele utviklingsperioden, ikke kun i transition-fasen.
- Kontroll av endringer: Benytte seg av versjonskontroll for å sikre at endringer synkroniseres mellom gruppemedlemmene.

4.2 Gantt-skjema



Task Name	Duration
Prosjektperiode	**106 days**
Forprosjekt (Inception)	**6 days**
Prosjektplan, skisse	6 days
Forprosjekt (Elaboration)	**25 days**
Overordnet Kravspesifikasjon	10 days
Ferdigstilling prosjektplan	2 days
Utvikle designdokument v1	15 days
Innlevering av prosjektplan (Life Cycle Objective)	0 days
Danne rapportmal	1 day
Skrive rapport	15 days
Valg av kart- og routing tjeneste	5 days
Studie av nødvendige teknologier	25 days
Gjennomgang av krav- og designdokument (Life Cycle Architecture)	0 days
Utviklingsperiode (Construction)	**40 days**
Utvikle designdokument v2	10 days
Lage skjelett til algoritme	5 days
Lage interface inn mot og ut fra algoritmen	5 days
Lage enkelt program for visning av data	5 days
Lage utbyttbar modul for routingtjeneste	10 days
Lage utbyttbar modul for karttjeneste	10 days
Utvikle algoritmen	30 days
Påskeferie	7 days
Videreutvikle program for visning av data	5 days
Testing	40 days
Ferdig programvare (Initial Operational Capability)	0 days
Skrive rapport	40 days
Ferdigstilling (Transition)	**22 days**
Kvalitetssikring	10 days
Skrive rapport	22 days
Innlevering av bacheloroppgave (Product Release)	0 days
Muntlig presentasjon	**13 days**
Forberedelse muntlig presentasjon	13 days
Muntlig presentasjon	0 days

Figur 3: Ganttdiagram

4.3 Hovedinndeling av prosjektet

Prosjektperioden er delt inn i fire faser, Forprosjekt, utvikling, rapport og muntlig presentasjon. Disse fasene, med unntak av muntlig presentasjon, er så fordelt på de fire fasene i RUP beskrevet over.

Forprosjektet er gitt en tidsramme på 31 arbeidsdager og er inndelt i Inception- og Elaboration-fasene fra RUP. Den lange planleggingsperioden er gjort med tanke på oppgavens art. Det er ikke store mengder kode som skal skrives, men kompleksiteten på koden er høy. Gruppen har derfor valgt å ta seg mer tid på kravspesifikasjon og design av algoritmen før det blir implementert kode. Prosjektplanen vil først utredes som en skisse og deretter ferdigstilles nærmere tidsfristen for innlevering. Dette er gjort fordi gruppen regner med å vite mer om oppgaven etter møte med hjemmetjenesten i uke 3. Valg av kart- og routingtjeneste vil gjøres tidlig fordi det er nødvendig for gruppen å lære seg å benytte tjenestene. Parallelt med planleggingen vil gruppen ha fokus på å lære nødvendige teknologier og programmeringsspråk som er vesentlige for å gjennomføre prosjektet.

Utviklingsperioden vil bli gjennomført i ukentlige sprinter der arbeidsoppgaver vil bli prioritert og utført av gruppens medlemmer. Gruppen har derfor satt opp grove inndelinger av arbeidet som skal gjennomføres, og omtrentlig når i utviklingsløpet det skal gjøres. Det forventes at oppgavene vil bli ytterligere oppdelt ettersom kravene til algoritmen blir klarere.

Etter 19. april skifter fokuset fra utvikling til rapportskriving. Dette betyr ikke at rapportskriving ikke skal finne sted før denne perioden eller at utviklingen stopper helt, men at primærfokuset skal ligge på rapporten og ikke lengre på produktet som skal utvikles.

Siste fase er knyttet til forberedelse og gjennomføring av den muntlige presentasjonen som skal finne sted i juni.

4.4 Plan for statusmøter og beslutningspunkter

På grunn av oppgavens kompleksitet ble det bestemt at gruppen skal ha ukentlige møter med veileder hver tirsdag morgen og med oppdragsgiver tirsdag ettermiddag. Ukentlige møter ble bestemt for at veileder og oppdragsgiver tidlig skal ha mulighet til å korrigere kursen til gruppen hvis de oppdager problemer som vil vanskeliggjøre eller hindre ett godt sluttprodukt.

Gruppen legger opp til å følge en arbeidsuke som spenner fra tirsdag til og med torsdag, pluss mandag uken etter. Dette valget ble tatt på grunn av at de viktige møtene med oppdragsgiver og veileder skjer på tirsdager som vil være styrende for fremtidig arbeid. Av denne grunn vil også deler av tirsdagen bli brukt til å planlegge arbeidsuken, spesielt under utviklingsperioden av prosjektet.

Viktige beslutninger knyttet til prosjektet, som foreksempel valg av karttjeneste, skal dokumenteres med begrunnelse for valget som er tatt. Beslutninger knyttet til hvilke

oppgaver som gjennomføres under utviklingsdelen av prosjektet vil tas av gruppemedlemmene selv, med innspill fra oppdragsgiver.

5. ORGANISERING AV KVALITETSSIKRING

5.1 Krav til dokumentasjon, standardbruk og kildekode

All dokumentasjon, inkludert planer, rapport, møtereferater, kommentarer i kode med mer, skal skrives på norsk. Etter samtale med oppdragsgiver var det ingen spesielle preferanser knyttet til dette, men det ble påpekt at det internt hos ETC ble benyttet norsk som dokumentasjonsspråk. Da dokumentasjon knyttet til prosjektet forhåpentligvis vil bli tatt i bruk for å videreutvikle prosjektet av ETC, ser gruppen det som hensiktsmessig å følge deres standarder.

Etter møte med veileder eller oppdragsgiver, vil gruppen bruke tid på å skrive om notater tatt under møtet til et møtereferat. Dette vil sendes til alle tilstedeværende. Formatet på referatet skal inkludere en liste av temaer som ble diskutert på møtet med bestemmelser eller oppklaringer diskusjonen førte frem til. Eventuelle kommentarer til møtereferatet fra tilstedeværende skal legges inn i referatet uendret merket i rød skrift.

All kildekode skal skrives på engelsk. Alle klasser, funksjoner og uklare variabler skal kommenteres. Kommentarene skal være på norsk. Dette gjøres for å opprettholde standarder som allerede er i bruk av ETC, slik at det skal være enklest mulig for dem å overta koden.

5.2 Verktøybruk

- Google Docs - En web-basert kontorpakke som gjør det enkelt å samskrive dokumenter. Google Docs vil bli brukt for utarbeiding av alle dokumenter, inkludert møtereferater, prosjektplaner og rapport.
- Bitbucket - Et web-basert vertssystem for prosjekter som benytter seg av GIT som system for versjonskontroll. All kildekode tilknyttet prosjektet vil lagres med tilhørende filhistorikk i et lukket repository der veileder og oppdragsgiver gis tilgang. BitBucket vil benyttes for å ha oversikt over endringer som gjøres i kildekoden, samt for å gi mulighet til å gå tilbake til tidligere revisjoner av koden.
- Visual Studio 2015 - En IDE utviklet av Microsoft primært brukt for utvikling for Microsoft Windows. Gruppen vil ta i bruk tilgjengelige verktøy som foreksempel kodeeditor, debugger og testdekning. Kildekode i C# vil skrives ved hjelp av Visual Studio.
- StyleCop - En plugin for Visual Studio for å forsikre at C#-kode følger et sett med angitte standarder.
- SonarQube - En plattform for inspeksjon av kode for å opprettholde høy kvalitet i koden. Verktøyet vil bli benyttet for å avdekke mangler i kode. Dette kan være at

koden ikke følger anbefalte standarder, er problematisk med tanke på sikkerhet og i noen tilfeller inneholder feil.

- Trello - Et web-basert prosjektstyrings verktøy. Trello vil bli benyttet for å holde oversikt over arbeidsoppgaver som skal gjøres, vise ansvarsfordelinger i gruppen og å vise fremgang i arbeidet.
- Microsoft Project Professional 2016 - Program utviklet av Microsoft som er laget for å lage og organisere en plan over prosjekter. Gruppen bruker programmet for å lage og vedlikeholde gantt-diagram.
- Microsoft Visio 2016 - Program for generering av diagrammer og vektorgrafikk. Programmet vil bli benyttet til å lage generelle diagrammer til bruk i rapport og prosjektplaner.

5.3 Risikoanalyse

5.3.1 Identifisere og analysere prosjektrisikoer

Skalainndeling for sannsynlighetsnivå består av lav, nokså lav, nokså høy og høy. Konsekvensnivået som følge av risikoene er inndelt i tolerabel, liten, problematisk og kritisk. Valget av en firedelt inndeling er gjort for å motvirke tendensen til å velge midt-nivået.

Nr	Risiko	Sannsynlighetnivå	Konsekvens
1	Feilestimering av tidsbruk på oppgaver over lengre tid	høy	problematisk
2	Algoritmen gir lite optimaliserte kjøreruter	høy	kritisk
3	Algoritmen er for treg	nokså høy	problematisk
4	Uklarhet rundt hvordan Algoritmen skal bedømme og veie ulike kriterier	nokså høy	problematisk
5	Dårlig kontakt med oppdragsgiver	nokså lav	problematisk
6	Dårlig kontakt med veileder	nokså lav	problematisk
7	Langtidssykdom hos Gruppemedlemmer	lav	kritisk
8	Tap av data	lav	kritisk
9	Integrasjon mellom GIS programvaren og algoritmen lar seg ikke gjennomføre	nokså lav	kritisk

5.3.2 Plan for håndtering av de viktigste risikoene

[1] Feilestimering av tidsbruk på oppgaver over lengre tid:
Det vil bli gjennomført hyppige møter med både veileder og oppdragsgiver for at de skal ha mulighet til å korrigere gruppen hvis de ser at dårlig tidsestimering vil føre til at arbeidet ikke blir gjennomført. Det er også lagt inn mye tid til å definere arbeidsmetodikk for hele prosjektet med den forventning at dette vil klargjøre eventuelle feilestimeringer tidlig og gi oss ett rammeverk for å håndtere dette. Dersom risikoen allikevel ser ut til å inntreffe vil ett virkemiddel for å motvirke konsekvensene være å nedjustere kravene på algoritmen.

[2] Algoritmen gir lite optimaliserte kjøreruter og [3] Algoritmen er for treg:
Det er lagt inn en lang periode i starten av prosjektperioden for kravspesifikasjon og design av algoritmen for å sikre at mulige faktorer som kan føre til dårlig optimalisering blir oppdaget tidligst mulig. Utviklingen av algoritmen vil utføres i sprinter. Dette gjør at man kan begynne å teste deler av løsningen tidlig og dermed oppdage eventuelle dårlige resultater med tid til å forbedre moduler. GIS programmet som skal beregne selve kjøreruten vil også testes tidlig og kunne byttes ut dersom det viser seg at den ikke gir tilstrekkelige resultater. Dersom løsningene som genereres er for dårlige eller implementasjonen er for treg, vil gruppen vurdere om tiden tillater å se på alternative implementasjonsmuligheter.

[4] Uklarhet rundt hvordan Algoritmen skal bedømme og veie ulike kriterier:
Det er avtalt ett møte med hjemmesykepleien tidlig i prosjektfasen slik at gruppen får klarhet i hvilke kriterier som skal bedømmes og hvor viktig de forskjellige kriteriene er opp imot hverandre. Gruppen vil forsøke å få kontaktinformasjon fra en representant fra hjemmetjenesten slik at spørsmål som dukker opp etter dette møtet kan besvares.

6 Terminologiliste

- Fordelingsalgoritme: dataprogram for fordeling av oppgaver over flere ressurser.
- GIS: geografiske informasjonssystem, programmer som har hovedfunksjonalitet knyttet til bruk av geografiske data.
- Front end: presentasjonslaget mellom brukeren og programmet.
- API : Application Programming Interface, applikasjonsprogrameringsgrensesnitt.
- RUP: Rational Unified Process, en utviklingsmodell som fokuserer på dokumentasjon samtidig som det skal være mulighet for endringer.
- Scrum: en utviklingsmodell som fokuserer på mye kundekontakt og mindre dokumentasjon.

7 Litteraturliste

Sommerville, I. (2011). Software Engineering - Ninth Edition. Boston: Pearson Education Inc.

Rational Software Corporation (udatert) Rational Unified Process: Overview. Hentet 26. januar 2016 fra http://sce.uhcl.edu/helm/rationalunifiedprocess/

Vedlegg E – Gruppeavtale

Grupperegler ved bacheloroppgave 2016

1. Ved uenighet presenteres forslagene til veileder og/eller oppdragsgiver der de kommer med innspill. Dersom det fortsatt er uenighet tas det endelige valget av veileder og/eller oppdragsgiver.
2. Dersom et gruppemedlem ikke utfører avtalt arbeid til avtalt tid over en lengre periode, kontaktes veileder for tvistløsning.
3. Møter med arbeidsgiver skal minst foregå annenhver uke.
4. Alle gruppemedlemmer har rett til å signere på vegne av gruppen.
5. Gruppen plikter å skrive notater over utført arbeid fortløpende under hele prosjektperioden.
6. Gruppeleder skal fungere som kontaktperson for gruppen og har ansvaret for å svare på forespørsler fra veileder og oppdragsgiver på vegne av gruppen.
7. Rollen som gruppeleder kan overføres til andre gruppemedlemmer dersom gruppeleder ikke lengre ønsker å ta på seg ansvaret.

Underskrifter

Sigurd Molnes Harkjerr _____ dato: _____

Helge Eriksen _____ dato: _____

Vedlegg F - Møtereferater

Møtereferat 1 - ETC

Dato: 12.01.2016
Tid: 13:00-14:30

Tilstede:

Dag L Solhaug
Øyvind Flatval
Helge Eriksen
Sigurd Molnes Harkjerr

Temaer diskutert

Tid og dag for ukentlige møter
Valg av dokumentasjonsspråk
Avgrensning av oppgaven
Generell utdypning av oppgaven
Selvutviklet routingalgoritme eller ekstern API
Kommunikasjon med hjemmetjenesten
Valg av systemutviklingsmodell
Kartdata tilgjengelig offline eller online
Valg av programmeringsspråk
Algoritmen som selvstendig modul eller integrert

Tid og dag for ukentlige møter
Det ble avtalt å ha ukentlig møte med oppdragsgiver tirsdager klokken 13.00. Det ble enighet om at hyppige, men kortere møter gagner både gruppen og oppdragsgiver.

Valg av dokumentasjonsspråk
Det ble spurt om oppdragsgiver hadde spesielle ønsker til hvilket språk rapporten og annen dokumentasjon skulle være skrevet på. Oppdragsgiver hadde ingen spesielle ønsker, men kommentert at de internt benyttet norsk som dokumenteringsspråk. Valget er derfor opp til gruppen selv.

Avgrensning av oppgaven
Da gruppen kun består av to personer ble det diskutert realistiske avgrensninger som følge av dette. Det ble enighet om at selve algoritmen er hovedfokus i oppgaven, men at utvikling av enkle moduler for fremvisning av generert data er hensiktsmessig. Dette i form av visning av tidslinjer per bil, timelister per person og rutevisning på kart.

Generell utdypning av oppgaven
Flere uklarheter ved oppgaven ble oppklart:

- Algoritmen skal typisk kjøres før en arbeidsdag, men at unntakstilfeller som at en person blir syk iløpet av en dag skal kunne håndteres. Algoritmen skal da kjøres på nytt, men skal kunne gjøre tilpasninger av ruten som allerede er generert.
- Det er mulig at tidsrommet et oppdrag skal utføres i kan være over lengre perioder, eksempelvis én uke.
- Algoritmen skal fungere for seg selv og ha et standard format for input og output.
- Type arbeidsoppgaver for en ansatt kan inkludere andre aktiviteter enn vedtaksoppdrag og kjøring, eksempelvis journalskriving og møter.
- Algoritmen skal fungere på ulike tjenesteytende gjerninger som feks leveranser og pizzabud.
- Hjemmetjenesten som eksempel på tjenesteytende næring:
    - Opererer innenfor et bestemt geografisk område med X ansatte og Y biler.
    - Oppdrag er definert gjennom at hjelpetrengende mottar et vedtak.
    - Et vedtak inneholder krav som type hjelp som trengs, tidsrom oppdraget skal gjennomføres, lengde i tid på oppdraget (ekskludert kjøretid), antall personer som trengs og bosted/adresse der hjelpen trengs.
    - Primærkontakt-kriteriet ble introdusert, det er ønskelig at samme ansatt besøker samme hjelpetrengende mest mulig.
- Routingdelen av algoritmen må kunne byttes ut, med tanke på lisensendringer eller at tjenesten ikke videreutvikles eller lignende.
- To prioriteter er viktige. At alle oppdrag utføres innen tidsperiode og å redusere kilometer kjørt per bil, per person. En vekting er trolig nødvendig for å tilfredstille disse prioritetene.
- Algoritmen bør fungere ved først å plassere urokkelige oppgaver og deretter prøve seg frem med de resterende.

Selvutviklet routingalgoritme eller ekstern API
Det ble bestemt at routing funksjonaliteten som er knyttet til algoritmen ikke skal utvikles av oss, men at en eksisterende løsning skal benyttes. Dette på grunn av oppgavens kompleksitet og fordi vi som studenter trolig hadde kommet opp med en dårligere løsning.

Kommunikasjon med hjemmetjenesten
Angående informasjonen som skal brukes som input til algoritmen ble det bestemt at det skulle gjøres en forespørsel til hjemmestjenessten om å få utlevert en vedtaksliste.

Det ble også bestemt at Dag L Solhaug skulle forsøke å sette opp et møte med hjemmetjenesten neste uke, uke 3, en gang mellom mandag og torsdag i tidsrommet 10:00-16:00 for å bedre kartlegge deres behov og krav til algoritmen.

Valg av systemutviklingsmodell
Det ble spurt om hvordan systemutviklingsmodell ETC følger. Oppdragsgiver svarte at de følger en avart av Scrum, hvor de brukte artifacter fra scrum, men utelot flere de ikke fant hensiktsmessig å følge, slik som daily standup meeting og tidsbestemte sprints. Det ble avtalt at gruppen selv velger hvilken modell de ønsker å følge. Vi ble tipset om å ikke bruke

en modell slavisk men å finne løsninger som fungerer for oss så lenge vi dokumenter valgene våre godt. Dersom Scrum velges som systemutviklingsmodell ble det enighet om at både Øyvind Flatval og Dag L Solhaug sammen skulle fungere som product owner.

Kartdata tilgjengelig offline eller online
Det ble diskutert forskjellige måter å benytte seg av kartdata. Oppdragsgiver og gruppen var enige om at det var nødvendig å ha kartdata lagret lokalt, da flere spørringer over internett ville føre til for lang responstid. Det kom frem at ETC hadde tidligere kunnskap med bruk av Google Maps og en annen karttjeneste men at disse trolig ikke var egnet da kartet må fungere offline. Marble, Osmsharp, ThinkGeo og Geoserver ble diskutert som andre mulige løsninger. Det kom også opp at det er viktig for ETC at koden ikke er tilgjengelig og åpen for alle. Det ble påpekt at dette kan være styrende for valg av karttjeneste.

Valg av programmeringsspråk
Gruppen spurte om arbeidsgiver hadde noen forslag eller preferanser til programmeringsspråk. C++ ble diskutert, men arbeidsgiver mente at det antaglig ikke ville være nødvendig å gå till ett så lavnivå språk. C# eller java ble diskutert som mulig språk fordi det ville være det eksterne routing APIet som vil ta seg av den tyngste oppgaven. JIT løsningene som disse språkene bruker har blitt veldig bra og er optimalisert til det miljøet som algoritmen skal kjøre i, dette resulterer ofte i raskere kode en selvlaget kode i C++. Oppdragsgiver bruker selv java og C# og siden de skal overta sluttresultatet ville det være fordelaktig og bruke en av disse, men valget faller til sist ned på gruppen.

Algoritmen som selvstendig modul eller integrert
Det ble spurt om hvordan oppdragsgiver så for seg at algoritmen skulle fungere sammen med 3.parts programmer. Oppdragsgiver forklarte at det ville være hensiktsmessig å ha algoritmen som en selvstendig modul som tok imot data på ett bestemt format og skrev data på et bestemt format.

Møtereferat 3 - ETC

Dato: 25.01.2016
Tid: 13:00-14:00

Tilstede:

Dag L Solhaug
Øyvind Flatval
Helge Eriksen
Sigurd Molnes Harkjerr

Temaer diskutert:

- Prosjektavtalen
- Prosjektplan
- Kravspesifikasjon
- Tilsvarsrett til møtereferat
- Møte med Hjemmesykepleiene
- Restriksjoner rundt valg av GIS-tjeneste?
- kart og routing-tjeneste.

Prosjektavtalen
Prosjektavtalen ble returnert fra ETC ble returnert til gruppen med underskrift fra Dag. Det er ønsket at avtalen blir underskrevet av dekan for så å sendes tilbake til ETC elektronisk.

Prosjektplan
Prosjektplan ble gjennomgått og endringer til denne ble foreslått.

Under effektmål ble det påpekt to andre viktige mål som bør være tilstede og muligens erstatte de nåværende: å redusere stress hos de ansatte, og å benytte mer tid hos pasient som følge av en optimalisering av ressurser.

Avsnitt 4, oppgavebeskrivelse: Det ble foreslått å reformulere avsnittet til å si at utviklingen skjer for hjemmetjenesten og skal fungere på andre tjenester istedet for motsatt. Det ble også satt fokus på at order 'justere' blir lagt litt mye i. Det bør tas en vurdering på om ikke man bør skille på å justere på tallverdier, som foreksempel antall liknende oppgaver som max skal utføres etter hverandre, og om kriterier skal utelukkes eller være gjeldende avhengig av bedrift.

Kravspesifikasjon
Det ble diskutert hva som bør inneholdes i en kravspek. Det ble satt fokus på at krav som er inne i systemet også bør dekkes på en måte. Samtidig ble gruppen rådet til å heller ta kontakt med veileder for å få et konkret svar på hvordan dette skal gjøres.

Tilsvarsrett til møtereferat

Siden møtereferatene vil bli publisert sammen med bacheloroppgaven ble det avtalt at oppdragsgiver skulle få tilsendt møtereferatene som skrives etter hvert møte. Hvis det skulle være uenigheter rundt det som er skrevet har alle som er tilstede rett til å få sitt standpunkt skrevet inn i møtereferatet. Dette vil da være uendret og merket i rødt

Møte med Hjemmesykepleien
Møtet med hjemmesykepleien torsdag 21.01.2016 ble kort oppsummert. Spesielt temaet rundt menneskefaktoren ble diskutert.

Det ble diskutert forskjellige metoder å implementere denne menneskefaktoren i avveiingene som algoritmen må foreta. Ett forslag var å tenke det i form av et RPG system hvor egenskaper tilegnes ansatte og behov/utfordringer tilegnes pasienter. Det ble også diskutert muligheten for ett kompatibilitetssystem hvor ansatte kan velge/velge bort pasienter. Disse faktorene og de etiske sidene vil være viktig å gripe fast ved i selve bacheloroppgaven.

Videre ble det presisert fra oppdragsgiver at programmet/algoritmen som skal utvikles vil ha ett høyere perspektiv enn det som ble forklart hos hjemmesykepleien. Algoritmen skal ta hensyn til flere avdelinger og finne den beste fordelingen av vedtak påtvers av ulike avdelinger. Det ble påpekt at den virkelige verdien av algoritmen kommer ved å inkorporere kompatibilitetssystem og ta hensyn til et større geografisk område med flere avdelinger. Et eksempel på et bruksområde som da gjør seg gjeldende er å se på hva virkningene kan bli ved å forflytte ressurser, foreksempel biler, over til andre avdelinger.

Restriksjoner rundt valg av GIS-tjeneste
Det ble forespurt hvordan ETC stilte seg til å betale en eventuell lisens for GIS-tjenester dersom dette skulle bli nødvendig. Gruppen ble bedt om å ikke ekskludere betalte løsninger og påpekte at open-source løsninger ofte også kom i en betalt versjon. De betalte lisensene har ofte studentlisenser der man kan utvikle uten restriksjon som kommer som følge av foreksempel GPL-lisensiert kode.

Kart og routing-tjeneste.
Tilslutt ble det diskutert litt om hvordan algoritmen skal bruke kart og routing-tjeneste for å finne en optimal rutefordelig. Gruppen presenterte noen tanker om hvordan de ser for seg at algoritmen kan designes og utfordringer rundt dette, slik som optimal fordeling mot bra nok fordeling, ble diskutert.

Møtereferat 4 - ETC

Dato: 02.02.2016
Tid: 13:00-14:00

Tilstede:

Dag L Solhaug
Øyvind Flatval
Helge Eriksen
Sigurd Molnes Harkjerr

Temaer diskutert:

- Geokoding
- Valg av kartdata
- Kravspesifikasjon

Geokoding
Det ble diskutert hvordan geokoding burde gjennomføres. Enten ved at algoritmen gjør det eller at det må gjøres før det sendes til algoritmen hos det eksterne programmet. Dag påpekte at det kan være lurt å la det eksterne programmet handtere dette, da det vil være lurt å ha feilhåndterings mekanismer klare hvis brukeren slår inn en adresse som systemet ikke kan geokode.

Valg av kartdata
Valg av routing-tjeneste og tilhørende kartdata ble diskutert. Gruppen har satt seg litt inn i en slik tjeneste kalt OsmSharp, denne tjenesten tilbyr routing basert på kartdata fra Open Street Map og har mulighet for lisensiering enten under GPL-lisens eller lukket ved å kjøpe en lisens for ca 3000 kroner i året. Gruppen spurte hvordan utvikling med GPL lisensen i bacheloroppgaven ville påvirke sluttproduktet som ETC skal overta. ETC mente at gruppen burde kunne utvikle prosjektet lukket med GPL-lisens også vil ETC kjøper lisens når de skal overta prosjektet.

Det ble også diskutert hvordan routing og optimalisering av kjøreruter burde hånteres. Gruppen har prøvd å implementere ett lite eksempel fra OsmSharp og merket at kartdata på pbf-format tok lang tid å benytte til routing med data over større geografiske områder, men at det i OsmSharp kunne optimaliseres ved å konvertere dette til routing-format. Allikevel ble det diskutert om ikke gruppen burde se på muligheter for å optimalisere kartdata, for eksempel ved å kun laste nødvendig data for det spesifikke geografiske området.

ETC var bekymret for at OsmSharps bruk av Open Street Maps karttda ikke var bra nok og ønsket at gruppen skulle se mer på de store kartdata tilbyderne. Disse vil antagelig ha mer nøyaktig og oppdatert informasjon. Det vil også være mindre sannsynlig at de slutter å

oppdatere sine data, da de ikke baserer seg på frivillighet. Kartdata tilbydere som Here (tidligere nokia maps) Tomtom og Garmin ble foreslått

Kravspesifikasjon
Det ble kommentert at use-caset ‘Endring av rute’ var noe uklar. Det kom ikke godt nok frem hva dette innebærte og det ble foreslått å konkretisere use-caset ved bruk av eksempler.

Det ble oppklart at når vi i ulike Use-Case snakket om bruk av JSON gjaldt dette kommunikasjon mellom tredjepartsprogrammet og algoritmen og ikke bestillingssystemet til bedriftskunden som benytter systemet. Gruppen burde tydeliggjøre at bruk av JSON som spesifisert i use-caset gjelder internt mellom tredjepartsprogrammet og algoritmen. Dag påpekte også at programmet vi omtaler som tredjepartsprogram eller presentasjonsapplikasjon egentlig blir et førstepartsprogram for ETC og at dette også bør komme tydeligere frem. Det ble diskutert om ikke et enkelt diagram kunne brukes til å få frem disse uklarhetene.

Møtereferat 6 - ETC

Dato: 19.02.2016
Tid: 14:00-15:00

Tilstede:

Dag L. Solhaug
Øyvind Flatval
Helge Eriksen
Sigurd Molnes Harkjerr

Temaer diskutert:

- Optaplanner
- Endring på oppgave som følge av bruk av Optaplanner

Optaplanner

Gruppen fortalte om hvordan deres arbeid med design dokumentet hadde gjort det klart at oppgavens kompleksitet og størrelse var av en slik grad at det ikke ville være tid til å gjennomføre alle use-case. Gruppen ser derfor for seg to mulige alternativer. Snevre inn oppgaven og utvikle en algoritme som løser dette mindre problemområdet, eller ta i bruk ett rammeverk slik som Optaplanner for å løse hele problemområdet. ETC stilte seg positive til å ta i bruk ett slikt rammeverk.

Endring av oppgave som følge av bruk av Optaplanner

Det ble diskutert hvordan oppgaven må endres som følge av bytte til et rammeverk slik som optaplanner. Kravet om at algoritmen skal fungere generelt økes og gruppen vil forsøke å dekke ulike moduser. En langsiktig modus som forsøker å se om besparelser kan gjøres ved å bytte om /redusere på biler og ansatte på tvers av depoter, og en kortsiktig modus der ansatte og biler er fastsatt sitt depot. Dag kommenterte at en mulig tilleggsutvidelse er større fokus på presentasjon. Algoritmen skal fortsatt fungere med et fastsatt inn-datasett og generere et resultat med hensyn på dette. Hvordan algoritmen løser oppgaven skal avgjøres hovedsaklig gjennom meta-data delen av dette datasettet. Videre ble det avtalt at gruppen skal sette seg inn i hvordan Optaplanner kan brukes for å løse dette.

Møtereferat 7 - ETC

Dato: 01.03.2016
Tid: 13:00-13:30

Tilstede:

Øyvind Flatval
Helge Eriksen
Sigurd Molnes Harkjerr

Temaer diskutert:

- Statusoppdatering
- Karttjeneste
- Til neste gang

Statusoppdatering
Gruppen fortalte litt om fremdriften siden forrige møte, spesifikt implementasjonen av JSON parsing, en enkel optaplanner implementasjon og problemgeneratoren som vil brukes for å generere datasett. Det ble også gått igjennom de to optimaliseringsmodusene, optimalisering med fastsatte ressurser og optimalisering med fordeling av ressurser for å være sikker på at ETC og gruppens forventninger til oppgaven er den samme.

Karttjeneste
Gruppen fortalte at valget av karttjeneste hadde falt på graphHopper med OSM. Dette vil antagelig fungere godt, men hvis det skulle vise seg at det ikke er godt nok er det implementert på en slik måte at det svært enkelt lar seg bytte ut.

Til neste gang
Det ble påpekt at gruppen må bli flinkere på å kommentere. Til neste gang ønsker ETC at Optaplanner eksempelet skal utbedres med flere DROOLS-regler og også få inn prosessering av distanse mellom lokasjoner inn i optimaliseringen.

Møtereferat 8 - ETC

Dato: 08.03.2016
Tid: 13:00-13:20

Tilstede:

Øyvind Flatval
Helge Eriksen
Sigurd Molnes Harkjerr

Temaer diskutert:

- Statusoppdatering
- Arbeid videre
- Kontrakt

Statusoppdatering

Gruppen presenterte fremgangen de har hatt siden forrige møte. De gikk kort igjennom de nye reglene som er implementert i fasen og fortalte om at utregning av distanser paralleliseres ved hjelp av tråder. Øyvind kommenterte at antall tråder som benyttes bør være konfigurerbart og heller bruke antall logiske prosessorer tilgjengelige, slik det nå gjøres, som en default-verdi. Dokumentet innsendt før møtet, bruk av Optaplanner, hadde Øyvind ingen spesielle kommentarer til. Øyvind kom på at utstyr tilgjengelig i biler kan være en faktor som gruppen ikke har tatt hensyn til i kravspesifikasjonen. Han påpekte at dette trolig ikke var viktig og ville komme tilbake til temaet ved et senere tidspunkt.

Arbeid videre

Det ble enighet om at gruppen skulle jobbe videre med designdokumentet og implementasjon av fasen: dele oppdrag i ruter per depot. Gruppen skal også se på en enkel visning av dataen, i denne sammenheng sa Øyvind at han hadde en link til ett veldig enkelt verktøy for å visualisere plotting av punkter på ett kart som han skulle sende gruppen. Han hadde også noen linker han kunne sende oss angående ett VRP eksempel i optaplanner som arbeidet med kartdata over hele Belgia og en VRP datasett generator.

Kontrakt

Kontrakten med underskrift fra dekan ble overlevert til ETC.

Møtereferat 9 - ETC

Dato: 15.03.2016
Tid: 13:00-13:20

Tilstede:

Øyvind Flatval
Helge Eriksen
Sigurd Molnes Harkjerr

Temaer diskutert:

- Statusoppdatering
- Arbeid videre
- Avgrensning

Statusoppdatering
Gruppen beskrev svært kort hvordan de har implementert fase 2 av løsningen, hvordan de originalt så for seg å implementere dette, og hvordan de ønsker å utvide det de har implementert.

Avgrensning
Gruppen informerte Øyvind om at de begynner å se at oppgaven må begrenses enda mer enn de originalt så for seg for at det skal være mulighet å gjennomføre innenfor tidsfristen. Dette medfører at optimalisering av hvem ansatt som skal i hvilken bil blir fjernet og isteden blir bestemt på forhånd. Gruppen ser seg også nødt til å utelate implementering av en modul for å redusere antall ansatte og biler som trengs på hvert depot for å gjennomføre oppdragene.

Øyvind sa seg enig i at det var fornuftig å avgrense oppgaven slik at vi ender opp med noe som kan presenteres ved bachelorfremføring. For ETC sin del blir det svært viktig at implementeringen er godt dokumentert og laget slik at det er enkelt å sette seg inn i løsningen for videre utvikling fra deres side.

Arbeid videre
Gruppen meddelte at arbeidet videre vil fokusere på dokumentasjon, inkludert dokumentering av det som mangler av løsningen med begrunnelser for hvorfor det ikke blir implementert. Videre skal gruppen jobbe med å utvide fase to til å ta høyde for flere regler fra kravspesifikasjonen. Deretter ønsker de å implementere en løsning uten faseoppdeling.

Møtereferat 10 - ETC

Dato: 05.04.2016
Tid: 13:25-13:50

Tilstede:

Øyvind Flatval
Helge Eriksen
Sigurd Molnes Harkjerr

Temaer diskutert:

- Statusoppdatering
- Arbeid videre
- Avgrensning

Statusoppdatering
Gruppen beskrev kort hva de har gjort. Dette inkluderer depotrepresentasjon som en boolean verdi i datamodellen, regler for å unngå for sen utføring av oppdrag, og for tidlig ankomst til oppdrag, progresjon på presentasjonsapplikasjonen og dokumentering av de ulike fasene til algoritmen.

Arbeid videre
Gruppen fortalte om hovedfokuset de kommende to ukene. Utbedring og implementering av regler, få datasett fra algoritmen inn i presentasjonsapplikasjonen og lage en modul som leser innstillinger fra datasettets metadata og velger hvilke regler som skal være med under løsningen.

Øyvind understrekte at det blir viktig å dokumentere det vi ikke valgte å ta med og tanker om hva som burde bli gjort videre. Dette innebærer å utføre design av fasene vi ikke velger å ta med for at det skal bli enklest mulig for etc å videreutvikle prosjektet.

Avgrensning
Gruppen fortalte at slik de ser det vil det ikke være fornuftig å implementere en ny løsning uten faseinndeling slik som de ønsket, men isteden fokusere på å få en fullverdig løsning slik det er satt opp i dag.

Møtereferat 11 - ETC

Dato: 12.04.2016
Tid: 13:00-14:15

Tilstede:

Øyvind Flatval
Dag L. Solhaug
Helge Eriksen
Sigurd Molnes Harkjerr

Temaer diskutert:

- Statusoppdatering
- Oppgavens retning
- Arbeid videre

Statusoppdatering
Gruppen gikk igjennom hvilke oppgaver som var blitt gjennomført siden forrige møte. Dette inkluderte presentasjonsapplikasjonen som nå tar imot løsningssett fra algoritmen, problemgeneratoren som har fått utbedret funksjonalitet og flere regel-filer.

Dag ble også oppdatert på retningen oppgaven har tatt og en gjennomgang av hvilke regler som så langt er implementert ble utført.

Oppgavens retning
Etter gjennomgangen av oppgaven slik gruppen har implementert den hadde Dag noen spørsmål rundt hvorfor algoritmen ikke forsøker å redusere antall ansatte og biler, da han mente at dette var en del av optimaliseringsløsningen forespurt. Gruppen svarte at dette var noe de originalt hadde ønsket å ha med som en egen modus ved siden av den som er implementert, men at de hadde sett seg nødt til å begrense oppgaven slik at kun modusen relatert til fastsatte ressurser ble realisert. Dag argumenterte for at reduksjon av ansatte og biler også var vesentlig for den realiserte modusen

Arbeid videre
Gruppen meddelte at de vil se på muligheten til å redusere kjøretøybruk ved å inkludere regler som underbygger dette kravet, og fjerne eller justere reglen for jevn fordeling av oppdrag i den andre fasen så lenge det ikke fører til endringer av datamodellen i bunn.

Gruppen fortalte også at de vil fortsette arbeidet med å inkludere regler som fortsatt mangler.

ETC anbefalte gruppen å ta kontakt med hjemmetjenesten for et møte angående om en anonymisert liste satt opp for en periode kan overrekkes gruppen for sammenligning med algoritmens resultat med det samme datasettet.

Møtereferat 12 - ETC

Dato: 19.04.2016
Tid: 13:00-13:40

Tilstede:

Dag L. Solhaug
Helge Eriksen
Sigurd Molnes Harkjerr

Temaer diskutert:

- Statusoppdatering
- Møte med hjemmetjeneste
- Arbeid videre

Statusoppdatering
Gruppen orienterte Dag om regler som ble implementert som følge av ønsker fra forrige møte. Det ble også informert om at gruppen har avtalt et møte med hjemmetjenesten mandag 25. april.

Møte med hjemmetjenesten
Det ble diskutert hva gruppen skulle ta opp med Monica hos hjemmetjenesten. Gruppen ønsker å få tilgang til reelle data med adresser, oppgaver og ansatte for en hel dag hos hjemmetjenesten, hvis dette ikke er mulig ønsker de å få en oversikt over hvordan en typisk arbeidsag ser ut slik at de kan generere eksempeldata som ligner på reelle data. Dag anbefalte også å ta med ett datasett som gruppen har generert og spørre hvordan Monica ville fordelt oppdragene.

Arbeid videre
Gruppen fortalte om fokuset videre og gjorde det klart at rapporten blir hovedfokuset fremover mot prosjektinnlevering. Det ble også påpekt at kvalitetssikring også vil jobbes med parallelt med dette.

Møtereferat 1 - Veileder

Dato: 19.01.2016
Tid: 10:00-11:00

Tilstede:

Ivar Farup
Helge Eriksen
Sigurd Molnes Harkjerr

Temaer diskutert

- Generell fremdrift
- Tilbakemelding på prosjektplan
- Diskusjon rundt tidplan
- Statusrapport og statusmøte
- Råd angående kravspek
- RUP artifakter

Generell fremdrift:
Gruppen startet med å fortelle hvordan de ligger ann på prosjektplanen. Vi nærmer oss ferdig, men har mangler på punktene rammer og valg av SU-modell. Gruppen planlegger å begynne å se på kravspesifikasjon denne uken

Tilbakemelding på prosjektplan
Gruppen sendte sitt nåværende utkast av prosjektplanen, som Ivar skal lese gjennom før neste veiledningsmøte, når gruppen sender mail om at det er klart for gjennomgang.

Gruppen godtok også å gi Ivar lesetilgang til kildekoden når et repository opprettes. Han ble også tildelt lesetilgang til trello-boarded som brukes av gruppen.

Det ble diskutert litt hva som ligger i begrepet standardbruk og kildekode. Ivar foreslo å legge inn noen ord i prosjektplanen angående kodestandard. Hvilken avhenger av hvilket språk vi skal programere i, C# eller java.

Ivar gjorde det også klart at dersom noe konkret skulle diskuteres på veiledningsmøter som han bør sette seg inn i, så burde materialet som skal diskuteres sendes til han på forskudd.

Det ble også klargjort hvilke ekstrakriterier som blir satt som oss som ingeniørstudenter. Etter innleveringen av prosjektplanen skal gruppen ha en muntlig fremføring av prosjektplanen på engelsk for veileder.

Diskusjon rundt tidsplan
Ivar pressiserte at det er svært viktig at det blir dokumentert hvilke valg som blir gjort underveis og at rapporten burde kontinuerlig arbeides på. Dette passer godt med planene

gruppen har gjort, men må komme bedre frem i gantt-diagrammet eller på annen måte dokumenteres

statusrapport og statusmøte
Krav til statusrapporter og statusmøter ble diskutert. Ivar meddelte at det ikke lengre er noen krav til slikt i bacheloroppgaven lengre, men at det kan være lurt å benytte seg av dem i henhold til milepælene for å se i hvilken grad de er nådd.

Råd angående kravspek
Ivar anbefalte å holde en tett dialog med oppdragsgiver. Han anbefalte å tenke oss at vi skal være i stand til å begynne å kode algoritmen. Han påpekte også at oppdragsgiver trolig har en klar idé om hvordan algoritmen skal fungere, og at vi bør presse dem mest mulig for informasjon.

Han mente også gruppen burde høre med oppdragsgiver hvordan det er med henvendelser til dem utenfor møtetidspunkter.

RUP artifakter:
Gruppen fortalte om problemer rundt utnyttelse av RUP's artifakter knyttet opp til ønske om å bruke den metodikk og lærdom som gruppen har tilegnet seg i faget Systemutvikling. Gruppen ønsker å bruke de artifaktene som ligner på det de har lært i SU, men vet ikke helt hvordan det best skal forklares i dokumentet. Ivar sa seg enig i at det ikke er hensiktsmessig for en gruppe på to, som skal kjøre en oppgave på ett halvt år, å følge full RUP og at det derfor relativt enkelt bør la seg gjøre å argumentere for mye mindre artifakt bruk.

Møtereferat 4 - Veileder

Dato: 10.02.2016
Tid: 10:15-11:15

Tilstede:

Ivar Farup
Helge Eriksen
Sigurd Molnes Harkjerr

Temaer diskutert

- Kravspesifikasjon
    - Datasett
    - Resultatmål og lovgitte krav
    - Sikkerhet og personver
    - Etiske hensyn
- Designdokument
- Presentasjon av prosjektplan

Datasett
Gruppen var nyskjerring på å vite om hva Ivar syntes om punktet 5.3 Datasett og om JSON-eksempel var en bra måte å forklare dette på. Ivar svarte at dette var godt forståelig og ville hjelpe en leser godt med å sette seg inn i oppgaven. Han foreslo å eventuelt liste opp alle dataer med datatyper som kan oppstå.

Videre gjennomgang av datasettet viste at gruppen burde forklare i teksten at Kapasitetsberegninger er ett hensyn som ikke skal implementeres i vår løsning, men at det skal bygges inn slik at algoritmen enkelt skal kunne ta det med i beregninger. Dette burde stå i både prosjektrapporten, under avgrensninger og i kravspek som kommentarer til JSON.

Resultatmål og lovgitte krav
Under resultatmål mente Ivar det burde komme klarere frem at gruppen går for å finne en "tilnærmet god" løsning og ikke den optimale. Han påpekte, som gruppen også har vurdert, å sammenligne resultatet av algoritmen på et senere stadie, med et uttømmende søk for å se hvor godt resultatet faktisk ble. Dette vil være med på å styrke rapporten.

Gruppen ba Ivar se over punktet lovgitte krav fordi vi hadde beskrevet det så kort og tenkte det kanskje krevde mer utdypning. Fordi informasjonen som meddeles der er såpass selvsagt, mente Ivar at punktet ikke trengte videre utdypning.

Sikkerhet og personvern
Iforhold til usikkerhet om lover rundt personvern knyttet til både lagring av personinformasjon og kompatibilitetsdata ble gruppen anbefalt å se på datatilsynets "personvern på 1-2-3" - en forklaring av personvernsloven for dem som ikke studerer jus.

Etiske hensyn
Kompatibilitetslister er ett krav fra ETC, men punktet etiske hensyn tar for seg fremgangsmåter for å samle inn data om kompatibilitet, det at dette er utenfor oppgavens område burde kommenteres. Det er ikke gruppen som skal foreta innsamlingen, men kun benytte den.

Designdokument
Det ble spurt om Ivar hadde noen tips før gruppen går igang med designdokumentet. Ivar påpekte at datastrukturen vil trolig bli viktig, fordi dataen må aksesseres raskt for at algoritmen skal fungere raskt. Ivar påpekte at design av en algoritme er annerledes enn design av et ordinært program. Det blir viktig å få klart frem algoritmenivåene og de konkrete fremgangsmåtene i hver algoritmefase som benyttes.

Presentasjon av porsjektplan
Gruppen holdt en uformell presentasjon av prosjektplanen på norsk-engelsk for Ivar.

Møtereferat 1 - Hjemmetjenesten

Dato: 21.01.2016
Tid: 10:00-12:00

Tilstede:

Øyvind Flatval
Monica Kristiansen
Helge Eriksen
Sigurd Molnes Harkjerr

Temaer diskutert

- **En normal arbeidsdag**
- **Spesifisering av vedtak**
- **Avveiinger mellom de ulike faktorene**
- **Roller blant de ansatte**
- **Tid på ruteplanlegging idag**
- **Hvordan gjøres endringer i timeplanen**
- **rapporteringssystem for status på oppgaver**
- **Videre kontakt**

En normal arbeidsdag

Saksbehandling av et vedtak tar ofte litt tid med utfylling og saksbehandling. Det opprettes et vedtak først ved at en beskjed kommer inn på telefon til "tildeling og koordinering"-avdelingen hos Gjøvik Kommune. Søknad kommer deretter via dem til avdelingen.

Man forsøker å bruke 30% av arbeidstiden blant de ansatte på aministrative oppgaver. Dette inkluderer rapportering, møter og kjøretid blant annet. Rapportering kan foreksempel være blodsukkermålinger hos pasient. Rapporteringen kan gjøres via vanlig pc eller gjennom PDA-ene som brukes av de ansatte.

Det forsøkes å planlegge arbeidsdagene slik at alle er inne ca klokka 12 for en spisepause. Dette fungerer da som et felles møtepunkt hvor man kan bytte oppgaver eller avtale å hjelper hverandre.

Hjemmesykepleien opererer med to skift, dag og kveld. Natt håndteres av en annen avdeling/instutisjon fordi det er få faste vedtak som må følges opp på natt, der er det mes alarmbesøk.

Spesifisering av vedtak

Ett vedtak inneholder oppgaver som skal gjennomføres hos pasienten. Arbeidsoppgaver er klart definert i datasystemet og har ett tidsestimat på hvor lang oppgaven burde ta. En arbeidsoppgave kan for eksempel være å gi pasienten en dusj, til dette kan det da være definert at det skal gjøres på 10 minutter. Tidsestiamtet er derimot ikke noe absolutt krav, pleieren kan gå inn i systemet og endre tildelt tid hvis de ser at denne pasienten trenger f.eks 20 minutter for å fullføre dusj. Ingen vedtak har en fast tid på når de må gjennomføres, men må fordeles gjennom dagen til fornuftige tider. Det finnes derimot faktorer som kan spesifisere når ett besøk må gjøres, slik som for eksempel medikamenter som må tas til fastsatte tider.

Når ett vedtak kommer til hjemmesykepleien plasseres det så på en av flere ulike lister ved å registrere dato og eller klokkeslett det skal gjennomføres innenfor. Nye vedtak kan komme inn hver dag. Flesteparten av vedtakene er over lang tid men også noen kortvarige, slik som ved armbrudd. Vedtak eller arbeidsoppgaver kan også fjernes ved en endringsmelding.

Listene genereres hver dag og oppgavene må så inspiseres og eventuelt flyttes for at det skal gå opp. At listene genereres daglig er ett valg hjemmesykepleien har gjort, de kunne også blitt generert over lengre tid. Listene vises så på alle de ansattes PDAer. Ulike lister har ulike krav til kompetanse slik at ett vedtak som har arbeidsoppgaver som kun kan gjennomføres av en sykepleier må plasseres i en liste for sykepleiere. Det finnes også en liste kalt tungtransport hvor det er to personer som kjører sammen og utfører de vedtakene hvor det kreves flere personer for å gjennomføre, slik som stell av tunge pasienter. Ved unntakstilfeller hvor ett vedtak som ikke ligger under tungtrafikk lista allikevel trenger to, slik som at en tung pasient har falt og pleieren klarer ikke løfte personen opp igjen alene, ringer pleieren etter hjelp fra andre pleiere i nærheten som har tid.

Vedtak som inneholder arbeidsoppgaver som har krav til spesiell kompetanse og har oppgaver uten slike krav gjennomføres uansett fult ut av pleieren. Det er altså ikke slik at en sykepleier reiser rundt å setter sprøyter, men lar husvasken stå igjen til noen andre.

Avveiinger mellom de ulike faktorene
Det første som vurderes er geografiske problemstillinger. Man forsøker å redusere kjøreavstand mest mulig ved å sette opp hensiktsmessige ruter.

De ansatte deles uformelt i ulike roller basert på hvordan de håndterer pasienter. Under listegenerering reflekteres det på hvilke ansatte som passer for å håndtere de ulike vedtakene. Disse valgene gjøres manuelt av de som genererer listene.

Det ble diskutert hvor viktig det var at kontaktperson besøker samme pasient. Behovet for at kontaktperson besøker samme bruker er ikke nødvendigvis et krav. Brukere er forskjellige og det er ikke like viktig for alle. Dessuten kan vanskelige pasienter bli en påkjenning for kontaktperson hvis det blir slik at han/hun alltid må besøke pasienten.

At en ansatt må utføre samme type oppgave etter hverandre er heller ikke en stor faktor. Vedtakene inneholder som oftest ulike oppgaver som skal gjennomføres som er med å sørge for at det blir variasjon uansett.

Roller blant de ansatte
De ulike ansatte har forskjellig kompetanse. Eksempler inkluderer ufaglærte, assistenter og sykepleiere. Noen vedtak krever en gitt kompetanse. Listene er gjerne delt på kompetanse. Foreksempel kan en liste være forbeholdt assistenter med enklere oppgaver. Dersom et vedtak inneholder flere oppgaver der en av oppgavene krever en spesiell kompetansegrad, utføres også de andre oppgavene av den samme ansatte når de først er der.

De ansatte har enten turnus inne på avdelingen eller ute på kjøring. Alle som jobber på kjøring, har lappen. Det hender at de som kjører "tungtransporten" kan ta med ansatt med inne-turnus på grunn av frafall ved foreksempel sykdom.

Hvert bruker har også en kontaktperson knyttet til sitt vedtak. Mål er at kontaktperson skal danne seg et bilde av hva brukeren trenger og at brukeren vet hvem kontaktpersonen er. Kontaktpersonen har kompetansenivå i forhold til brukerens behov. Kontaktpersonen skal ha totaloversikt for sitt vedtak. Dette innebærer blant annet og oppdatere tiltaksplaner, sørge for at hjelpemidler er tilgjengelig for pasient og være en del av førstegangssamtale med nye brukere der deres vedtak blir gjennomgått.

Tid på ruteplanlegging idag
Det ble anslått at det brukes mellom 15 og 30 minutter på ruteplanlegging idag. Dette inkluderer generering av lister, og deretter redigere disse listene til ønsket resultat. Det ble også påpekt at prosessen er lengre på fredager da dette da gjøres for både lørdag og søndag i tillegg.

Det ble påpekt at hvis vedtaksdataene hadde blitt mistet hadde det tatt lang tid å føre disse inn igjen i systemet.

Hvordan gjøres endringer i timeplanen
Dersom et nytt vedtak kommer inn midt på dagen, må listene genereres på nytt for å inkludere det nye vedtaket. Det er ofte tilfelle at hjelpen gjør seg trengende umiddelbart. Et nytt vedtak kan foreksempel komme klokken 14.00 og kreve hjelp allerede samme kveld.

Dersom en ansatt ser at det blir en forsinkelse på et vedtak ringes brukeren for å informere om dette. Er forsinkelsen stor nok, kan den ansatte utlyse vedtaket. Utlyste vedtak kan sees av alle kjørende og kan tas av alle ansatte som ser de har tid til å gjennomføre vedtaket.

Oppdrag kan også kanselleres, enten av hjemmetjenesten eller av pasienten selv. Mulige grunner for kansellering inkluderer at veier ikke er måket og at det er umulig å komme frem, eller at pasient har en legeavtale i perioden hjelpen skulle funnet sted. Dersom kanselleringen gjøres fra avdelingen. Dersom kanselleringen finner sted kort tid før et besøk skal finne sted, informeres den ansatte som skal utføre besøket over telefon.

rapporteringssystem for status på oppgaver
De ansatte som er ute på kjøring kviterer fortløpende statusen på sine kjøringer.

Oppdragene kan ha følgende status: ferdig utførte, påbegynte, kanselerte eller ikke påbegynte. En oversikt kan vises i deres systemet der oppdragenes status er representert ved farger.

Videre kontakt
Det ble forespurt om det var mulig å opprette kontakt med hjemmetjenesten på et senere tidspunkt dersom flere spørsmål skulle dukke opp. Dette var helt greit. Monica Kristiansen kan nås på telefon 91837100. Bente og May-Britt kan også svare på spørsmål på 61158845

Møtereferat 2 - Hjemmetjenesten

Dato: 25.04.2016
Tid: 13:00-14:15

Tilstede:

Monica Kristiansen
Helge Eriksen
Sigurd Molnes Harkjerr

Temaer diskutert

- Data om ansatte
- Data om biler
- Data om oppdrag
- Data om depot
- annet

Data om ansatte

Avdelingen i Hunndalen opererer med 10 lister med oppdrag som fordeles på de ansatte. 7 på dagen og 3 på kvelden i ukedager og 7 i helgen med 4 på dag og 3 på kveld..Listene fordeles med hensyn på hvilke kompetanse som kreves for å gjennomføre oppdragene på listen. To av listene utføres sammen, såkalt tungtransport, som krever mer enn en person for å gjennomføre. Hjemmetjenesten opererer i hovedsak med to kompetanser, Sykepleier og ikke-sykepleier. Sykepleier har oppgaver som kun de kan utføre, slik som visse typer sårstell og de har visse oppgaver som de ikke utfører, slik som praktisk bistand. Avdelingen har hovedsakelig minst 3 sykepleiere på dagtid og 1 på kveldstid. Resten av listene fylles av hjelpepleiere og assistenter som begge kan utføre hovedsaklig de samme arbeidsoppgavene. Ansatte som jobber dag har typisk arbeidsdag fra 07.30 til 15.00 og Kveldsskift er typisk delt inn i to hvor to av de ansatte begynner klokken 15.30 og en begynner 18.30 og arbeider typisk til 23.30

Data om biler

Avdelingen i Hunndalen har 6 kjøretøy.

Data om oppdrag

Oppdrag kan oppstå mellom 08:00 til ca 23:30. Antall oppdrag per liste varierer typisk mellom 10 og 20. Tidsvinduene er omtrentlige. Oppgavenes art vurderer når ca de blir plassert. Morgenstell skjer typisk fra 08:00-12:00. Det tas hensyn til pasientenes ønsker om mulig. Medisinering og injeksjon må skje innenfor fastsatte tider. Disse tidsvinduene er rundt 2 timer lange der oppgaven MÅ utføres. Kveldsstell tar kortere tid enn morgenstell og bør også utføres innenfor fastsatte tidsvinduer. Brukere forventer å bli besøkt til ca samme tid hver gang.

Oppdragenes lengde styres av oppgavetypen. Praktisk bistand som renhold tar typisk 90 minutter, morgenstell typisk 15 min med et tilegg på 30 min dersom dusj er inkludert. Oppdrag varierer omtrentlig mellom 5 - 120 minutter.

Dersom to ansatte utfører oppdrag sammen, blir ulike oppgaver innad i oppdraget registrert på de ulike ansatte. Oppgaver som er felles, registreres hos begge de ansatte. Lengden oppdraget tar å utføre blir dermed redusert, men ikke halvert.

Oppdragene slik det registreres idag, inneholder ingen data om antall ansatte som bør utføre oppdraget eller hvilke kompetansenivåer som må utføre oppdraget.

Kravet om kompatibilitet, at krevende pasienter besøkes av ansatte som mestrer å håndtere dem, gjelder kun 3-4 vedtak. Resten er somregel medgjørlige.

Data om depot
Området avdelingen opererer i er gjengitt grovt i følgende kart:

De andre avdelingene i området, er av ca lik størrelse som denne.

Annet
Gruppen ga en kort forklaring på hvordan programmet er ment å fungere ved at alle oppdrag for en periode, typisk en dag fordeles på de tilgjengelige ansatte og biler.
Monica kommenterte at løsningen tillsynelatende ikke tar hensyn til primærkontakter.
Gruppen svarte at det stemmer, men at det er et kriterie som relativt enkelt kan legges inn.

Vedlegg G – Timelogger

Timelogg – Sigurd Molnes Harkjerr

Dato	timer	Klokkeslett	Beskrivelse
fre 08.01.2016	3	12:00-15:00	Nedskrevet grupperegler, møteforberedelse
man 11.01.2016	7	10:00-12:00	Lynkurs ved Tom Røise.
		12:00-13:00	Møte med veileder angående vår plan for arbeidet fremover.
		13:00-15:00	Revidering av spørsmål til oppdragsgiver. Forespurt møte med oppdragsgiver. Smått sett på ulike valg for karttjenester tilsendt fra oppdragsgiver på mail.
		20:00-21:30	Lesing av tidligere bacheloroppgave.
		21:30-22:00	Utbedring av grupperegler ved å ta hensyn til rollen som grupperegler.
tir 12.01.2016	7	09:00-13:00	Skriving av Prosjektplan
		13:00-14:30	Møte med oppdragsgiver, Dag L Solhaug, Øyvind Flatval
		14:30-16:00	Oppsummering og skriving av møtereferat
ons 13.01.2016	7	09:00-11:00	Skriving av Prosjektplan
		11:00-14:30	Lage Gantt-diagram
		14:30-15:00	Skriving av Prosjektplan
		19:00-20:00	Studie: Programering i C#
tor 14.01.2016	7	09:00-15:30	Skriving av Prosjektplan
		15:30-16:00	Forberedelse møte med hjemmetjenesten
fre 15.01.2016	2.5	12:00-14:30	Diskusjon og studie for å utføre valg av SU-modell
man 18.01.2016	7	09:00-16:00	Skriving av Prosjektplan, med fokus på valg av SU-modell
tir 19.01.2016	7	09:00-10:00	Forberedelse til møte med veileder.
		10:00-11:00	Møte veileder
		11:00- 11:30	Skriving av møtereferat, Veileder
		11:30-13:00	Forberedelse til møte med oppdragsgiver
		13:00-14:00	Møte veileder
		14:00-14:30	Skriving av møtereferat, ETC
		14:30-16:00	Forbedring av gantt-diagram
ons 20.01.2016	7	09:00-13:00	Gjennomgang av prosjektplan, lage terminologiliste
		13:00-14:30	Forberedelse møte med hjemmetjenesten
		14:30-16:00	Sette opp mal for kravspesifikasjon. Arbeid med Use-Case diagram
tor 21.01.2016		09:00-10:00	Forberedelse til møte med hjemmetjenesten + reisetid til møte
		10:00-12:00	Møte med hjemmetjenesten
		12:00-14:00	Skrive møtereferat
		14:00-16:00	Skriving av kravspesifikasjon
fre 22.01.2016	2	13:00-15:00	Innledende studie på vehicle routing problem with time windows
man 25.01.2016	7	09:00-13:00	Skriving av kravspesifikasjon
		13:00-14:00	Møte med ETC
		14:00-15:00	Skrive møtereferat

		15:00-16:00	Algoritme-research
tir 26.01.2016	5	09:00-10:00	Forberedelse møte veileder
		10:00-11:30	Møte med veileder + skriving av møtereferat
		11:30-13:30	Karrieredag
		13:30-16:00	Ferdigstilling prosjektplan
ons 27.01.2016	8	09:00-12:00	Ferdigstilling og levering av prosjektplan + prosjektavtale
		13:00-16:00	Prototype algoritmedesign
		18:00-20:00	Research: Traveling Salesman og Vehicle Routing Problem teori.
tor 28.01.2016	7	09:00-16:00	Lage domenemodell og utbedring av krav.
lør 30.01.2016	2	16:00-18:00	Research: Variable Neighbourhood Search og Sweep heuristics
man 01.02.2016	9	09:00-13:00	Kravspesifikasjon, ikke-funksjonelle krav
		13:00-16:00	Valg av kart- og routingtjeneste
		18:00-20:00	Gjøre seg kjent med C# og Windows Forms
tir 02.02.2016	7.5	09:00-12:00	Kravspesifikasjon.
		12:00-13:00	Foreberedelse, møte ETC
		13:00-14:00	Møte, ETC
		14:00-15:30	Møtereferat / lage "Deploymentdiagram"
		18:00-19:00	Modulariseringsprototype Karttjeneste
ons, 03.02.2016	8	09:00-10:00	Forberedelse møte med veileder
		10:00-11:00	Møte, Veileder
		11:00-16:00	Sette opp git repo og research: unit tests og code coverage i visual studio 2015.
		19:00-20:00	Dokumentering av diskusjoner som er blitt gjort ved valg av karttjeneste.
tor, 04.02.2016	6	09:00-15:00	Detaljert kravspesifikasjon, lagde datasett til algoritmen i JSON-format
søn, 07.02.2016	3	15:00-18:00	Lagde skisse til JSON interface for algoritmen.
man, 08.02.2016	10	10:00-17:00	Kravspesifikasjon, overblikk over systemet, gå over use-cases, input til algoritmen
		18:00-20:00	Fortsatte på skisse til JSON interface, opprettet wordpress blogg for hjemmeside
		21:00-22:00	La til noe innhold og satt meg inn i wordpress.
tir, 09.02.2016	7	09:00-12:00	Kravspesifikasjon, absolutte/relative kriterier og ikke-funksjonelle krav
		12:00-13:00	Møteforberedelse, ETC
		13:00-13:30	Møte, ETC
		13:30-16:00	Møtereferat / Fikse på kravspek /sette opp designdokumentmal
ons, 10.02.2016	7	09:00-10:00	Forberedelse møte veileder
		10:00-11:00	Møte, veileder
		11:00-16:00	Fikse kravspesifikasjon, lage deploymentdiagram.
tor, 11.02.2016	5	09:00-10:00	Algoritmedesign
		10:00-12:00	Valg av kart- og routingtjeneste, Here og Bing maps
		13:00-15:00	Algoritmedesign, Savings Algorithm
fre, 12.02.2016	6	13:00-15:00	Algoritmedesign, Savings Algorithm
		16:00-20:00	Algoritmedesign, Savings Algorithm

søn, 14.02.2016	2	20:00-22:00	Algoritmedesign, Savings Algoritme
man, 15.02.2016	8	10:00-16:00	Algoritmedesign, Savings Algoritme + Tabu Search
		19:00-21:00	Algoritmedesign, Tabu Search
tir, 16.02.2016	8	09:00-10:00	Forberedelse, møte veileder
		10:00-11:00	Møte veileder
		11:00-13:00	Diskusjon og forberedelse møte med ETC
		13:00-13:30	Møte avlyst
		13:30-16:00	Se på muligheter for planleggingsrammeverk: Optaplanner
		19:00-20:00	Lese mer om optaplanner
ons, 17.02.2016	7	09:00-16:00	Fordypning optaplanner
tor, 18.02.2016	2	12:00-14:00	Fordypning optaplanner
fre, 19.02.2016	2	13:30-14:00	Forberedelse, møte ETC
		14:00-15:00	Møte, ETC
		15:00-15:30	Skriving av møtereferat
lør, 20.02.2016	2	12:00-14:00	Implementasjon av offline routingtjeneste - GraphHopper
man, 22.02.2016	7.5	09:00-12:00	Modellering av problemet knyttet opp mot optaplanner.
		14:00-16:30	Teori: optaplanner
		18:30-20:30	Teori: optaplanner
tir, 23.02.2016	7	09:00-10:00	Møteforberedelse Veileder + planlegging av dagen
		10:00-11:00	Møte, veileder
		11:00-13:30	Klassediagram over første problemstilling: fordeling på depoter
		18:00-20:30	Teori: Solution og Solver, klasser og interfaces
ons, 24.02.2016	7	09:00-12:00	Optaplanner implementasjon av fordeling av noder på depoter.
		12:00-13:00	Teori: Score Calculation i optaplanner
		14:00-15:00	Teori: Score Calculation i optaplanner
		15:30-17:30	Begynte å lage et lite program som genererer et problem med noder i gjøvik
tor, 25.02.2016	6	09:00-12:00	Skrive designdokument
		12:00-14:00	Skrive program som genererer problem med noder i Gjøvik
		14:00-15:00	Skrive designdokument: arkitekturmodell
fre, 26.02.2016	2	12:30-14:30	Skrive program som genererer problem med noder i Gjøvik
søn, 28.02.2016	2	10:00-12:00	Forbedret problemgeneratoren og den skriver nå problemet til fil.
man, 29.02.2016	7	09:00-16:00	Skrive designdokument: Algoritmens ulike moduser.
tir, 01.02.2016	8	09:00-12:00	Designdokument: Algoritmens ulike moduser.
		12:00-13:00	Forberedelse møte ETC
		13:00-13:30	Møte, ETC
		13:30-14:00	Skrive møtereferat
		14:00-15:30	Kommentering av kode.
		19:30-21:00	Koding og javadoc
ons, 02.03.2016	7	09:00-16:00	Designdokument og kommentering/endring av domenet til prosjektet.

tor, 03.03.2016	9	09:00-13:30	JsonInterfaceManager klasse og Logging
		14:30-16:00	Implementerte Logging i flere klasser.
		18:00-21:00	Lagde en factoryklasse for Solution objekter
fre, 04.03.2016	3	12:00-15:00	Paralellisering av routing mellom noder.
man, 07.03.2016	3	07:00-08:00	optaplanner teori: moves og neighborhoods
		19:00-21:00	Leste om rendering bibliotek-muligheter for java, Google Maps og Osm
tir,08.03.2016	8	09:00-10:00	Forberedelse møte veileder, planla arbeid fremmover
		10:00-11:00	Møte, veileder
		11:00-11:30	Skrev møtereferat og foreberedelse møte, ETC
		11:30-13:00	Optaplanner teori: chained variables, shadow variables
		13:00-13:30	Møte, ETC
		13:30-14:00	Skrive møtereferat
		14:30-16:00	Skrive ut solution i punkter og se resultatet via gps visualizer
		18:30-19:00	Optaplanner teori: chained variables, shadow variables
		20:00-21:00	Enkel nettvisning med google maps
ons, 09.03.2016	7	09:00-12:00	Design fase 2: fordeling av oppdrag i ruter
		12:00-13:00	Lynkurs ved Frode Haug
		13:00-16:00	Design fase 2: fordeling av oppdrag i ruter
tor, 10.03.2016	7	09:00-12:00	Design fase 2: fordeling av oppdrag i ruter
		12:00-16:00	Utførte endringer i prosjektstruktur
fre, 11.03.2016	3	12:00-15:00	Implementasjon av domeneklasser for fase 2
man, 14.03.2016	8	09:00-16:00	implementasjon av algoritmefase 2
		20:00-21:00	Satt meg mer inn i Drools regeler
tir, 15.03.2015	9	09:00-10:00	Forberedelse, møte med veileder
		10:00-11:00	Møte Veileder
		11:00-13:00	Skrive møtereferat, forberedelse møte med ETC og kommentering av kode
		13:00-13:30	Møte ETC
		13:30-14:00	Skrive møtereferat
		14:00-15:30	Skrive designdokument + lage diagram som viser hvordan chaining fungerer
		20:30-23:00	Skrive designdokument + utdypning dokument: valg av karttjeneste
ons, 16.03.2016	7.5	09:00-12:00	Skrive dokument 'Valg av karttjeneste'
		12:00-14:00	X-session
		14:00-14:30	Skrive dokument 'Valg av karttjeneste'
		20:00-22:00	Skrive dokument 'Valg av karttjeneste'
tor, 17.03.2016	5	09:00-10:00	Leste optaplanner teori: Moves og MoveSelectors
		10:00-11:00	La inn en construction heuristic fase til for å fikse en bug i builden
		11:00-13:00	Rapportskriving: fase 2 komplisert versjon
		13:00-14:00	Installering og testing av SonarLint og EclEmma code coverage plug-ins
fre, 18.03.2016	2	12:00-14:00	Lære om AngularJS for bruk i presentasjonsapplikasjonen.

fre, 25.03.2016	2	18:00-20:00	Lese om custom moves, optaplanner teori
lør, 26.03.2016	6	12:00-18:00	lære AngularJS og implementasjon av presentasjonsapplikasjon
man, 28.03.2016	2	12:00-14:00	implementasjon av presentasjonsapplikasjon
tir, 29.03.2016	4	12:00-16:00	Implementasjon av presentasjonsapplikasjon
ons, 30.03.2016	7	10:00-15:00	Implementerte støtte for depotbesøk under ruter.
		21:00-23:00	Begynne å jobbe med rekursiv oppdatering av ankomsttider.
tor, 31.03.2016	8	10-00:13:00	Implementasjon av rekursiv oppdatering av ankomsttider.
		13:00-17:00	Utbedre ProblemGenerator for å lage tidsvinduer med vettuge verdier.
		18:00-19:00	Utbedre ProblemGenerator slik at oppdragenes tidsvinduer tar hensyn til arbeidstider
fre, 01.04.2016	5	12:00-14:00	Gjennomgang av nylig implementert kode, og planlegging av den neste uken.
		21:00-00:00	Javadoc og opprydning i ProblemGenerator relaterte klasser.
lør, 02.04.2016	2	12:00-14:00	Lagde en testklasse for Area-klassen.
søn, 03.04.2016	2	15:00-17:00	Opprydning i AssignmentGenerator klassen.
man, 04.04.2016	8	10:00-13:30	Introduserte regler for å redusere tidlig ankomst og for sen ankomst til oppdrag
		13:30-14:00	Sette opp liste over oppgaver som gjenstår for algoritmen og presentasjonsprogram.
		15:00-17:00	Begynte å lage en json-exporter fil som eksporterer løsning som json.
		21:00-23:00	Lagde en ny metode i jsonExporter-klassen som manuelt konverterer til json.
tir, 05.04.2016	7	09:00-10:00	Gjennomgang av oppgaver som gjenstår og forberedelse møte veileder
		10:00-11:00	Møte, veileder
		11:00-11:30	Møtereferat, veileder
		11:30-12:00	Forberedelse møte ETC
		12:00-13:00	Sette opp repo for presentasjonsapplikasjon
		13:00-14:00	Møte, ETC
		14:00-14:10	Møtereferat, ETC
		14:10-15:00	Fiksing av repo etter misslykket mergefix.
		15:00-16:00	Få datamodell fra algoritme inn i presentasjonsapplikasjon
ons,06.04.2016	9	09:00-16:00	Få datamodell fra algoritme inn i presentasjonsapplikasjon
		20:00-22:00	Feilsjekking av format på json-løsningsfiler.
tor,07.04.2016	6.5	09:30-14:00	Diverse fikser av presentasjonsapplikasjon
		14:00-15:00	Lage en SolverConfigBuilder klasse, og refactore diverse kode.
		15:00-16:00	Endre struktur i MetaData klasse og introdusere ny klasse MetaDataSettings.
fre,08.04.2016	2	19:00-21:00	Delte opp drl-filer i mindre deler og skrev mer på SolverConfigBuilder-klassen.
lør, 09.04.2016	1	15:00-16:00	Begynte på implementasjonsdokument for presentasjonsapplikasjon.
man, 11.04.2016	7	09:00-12:00	Splitte opp regler i mindre deler som blir lagt til i fordelingen basert på metadata-innstillinger.
		12:00-16:00	La til vekting av regler fra Weights-objektet i MetaData.
tir, 12.04.2016	8	09:00-10:00	Forberedelse møte veileder
		10:00-10:50	Møte veileder
		10:50- 11:30	Møtereferat veileder
		11:30-12:30	Fiksing av bug som førte til at biler kunne reise tilbake i tid.

		12:30-13:00	Forberedelse møte, ETC
		13:00-14:30	Møte ETC
		14:30-15:30	Møtereferat ETC, og diskusjon om arbeid videre.
		15:30-16:00	Fiksing av bug som førte til at biler kunne reise tilbake i tid.
		19:00-20:00	Skriving av implementasjonsdokument for presentasjonsapplikasjon.
ons, 13.04.2016	10	09:00-10:00	Endret TimeWindow til å være en subklasse av TimeInterval
		10:00-12:00	Implementert regel: møter ansatte førerkortkrav
		12:00-13:00	Refactor, flytte getDistanceTo funksjon til Visit interface-et.
		13:00-16:00	Fikset bug i presentasjonsapp slik at ruter vises i rett rekkefølge
		19:00-20:00	Ordnet opp i dupliserende kode i presentasjonsapplikasjon.
		20:00-22:00	Utvided views i presentasjonsapp til å vise data om ansatte og biler.
tor, 14.04.2016	7	09:00-10:00	La til nye regelfil med vekt og 2 regler. reduserbruk av ansatte, og reduser bruk av kjøretøy.
		10:00-11:00	Fikset applikasjonsapp til å godta employees med vehicleid-er satt til null.
		11:00-12:00	La til competence og compatibility til løsnings-json filen.
		12:00-13:00	Gjenspeilet competence og compatibility i presentasjonsapplikasjon.
		13:00-16:00	Dokument intro til fordelingsalgoritmer.
fre, 15.04.2016	1	12:00-13:00	Dokument intro til fordelingsalgoritmer.
lør, 16.04.2016	3	13:00-15:00	Javadoc, sonarLint fiksing og skriving intro til fordelingsalgoritmer.
		19:00-20:00	Lagde test case for getDepartureTime funksjon.
man, 18.04.2016	7	09:00-16:00	Dokument intro til fordelingsalgoritmer.
tir, 19.04.2016	9	09:00-10:00	Forberedelse møte veileder
		10:00-11:00	Møte veileder
		11:00-12:30	Nedlasting av office 360 + konvertering av rapport mal.
		12:30-13:00	Forberedelse møte ETC
		13:00-14:00	Møte med ETC
		14:00-16:00	Utbedre rapport-skjelett
		18:00-20:00	Ordne format rapport, legge inn tekst fra skrevne dokumenter
ons, 20.04.2016	7	09:00-14:00	Arbeid med strukturering av rapport, diskusjon rundt oppsett.
		20:00-22:00	Arbeid med introduksjonskapittel i rapport.
tor, 21.04.2016	7	09:00-11:00	Arbeid med introduksjonskapittel i rapport.
		11:00-15:00	Renskrivning av designdok presentasjonsapp
		18:00-19:00	Så på muligheter for enhetstesting av drools-regler.
fre, 22.04.2016	3	12:00-15:00	Forberedelse møte med hjemmetjenesten
man, 25.04.2016	8	09:00-13:00	Skriving av kravspek som følge av avgrensninger gjort under utvikling.
		13:00-14:30	Møte med hjemmetjenesten
		14:30-15:00	Skriving av møtereferat hjemmetjenesten.
		20:00-22:00	Javadoc-forbedring av diverse klasser.
tir, 26.04.2016	7	09:00-12:30	Skriving av optaplanner delkapittel.
		12:30-13:00	Forberedelse møte ETC

		13:00-13:40	Møtereferat ETC, og diskusjon om arbeid videre.
		13:40-14:00	Skriving av møtereferat.
		19:00-21:00	skriving på intro til constraint satisfaction problemer + optaplanner delkapittel.
ons, 27.04.2016	7	09:00-14:30	Skriving av optaplanner delkapittel.
		18:30-20:00	Skriving av optaplanner delkapittel.
tor, 28.04.2016	7.5	09:00-12:00	Skriving av optaplanner delkapittel.
		12:00-15:00	Omskrivning kravspesifikasjon.
		21:00-22:30	Javadoc av diverse klasser.
fre, 29.04.2016	1.5	12:00-13:30	Skriving om shadow variabler.
lør, 30.04.2016	1.5	21:30-23:00	Javadoc av diverse klasser.
søn, 01.05.2016	1	22:00-23:00	Skriving av Teknologi, Angular
man, 02.05.2016	7	09:00-12:00	Skriving av Teknologi, Angular
		12:00-15:00	Skriving av Design - presentasjonsapp
		20:00-21:00	Skriving av Design - presentasjonsapp
tir, 03.05.2016	9	09:00-10:00	Diskusjon og forberedelse møte med veileder.
		10:00-11:00	Møte med veileder.
		11:00-15:00	Skriving av implemetasjonskapittel.
		20:00-23:00	Javadoc av diverse klasser.
ons, 04.05.2016	7	09:00-14:30	Implementasjonsdokument, Klassebeskrivelser
		17:00-18:30	Implementasjonsdokument, Klassebeskrivelser
tor, 05.05.2016	6	09:00-15:00	Implementasjonsdokument, Klassebeskrivelser + disposisjon
fre, 06.05.2016	2	12:00-14:00	Implementasjonsdokument, Klassebeskrivelser
søn, 08.05.2016	1	11:00-12:00	Javadoc av gjenværende klasser.
man, 09.05.2016	8	9:00-10:00	Implementasjonsdokument, Klassebeskrivelser
		10:00-15:00	Korrektur, kilde-, figur-, tabell- og kapittelhenvisning
		17:00-19:00	Kildehenvisning og korrektur.
tir, 10.05.2016	7	09:00-10:00	Kildehenvisning og korrektur.
		10:00-10:30	Møte, veileder
		10:30-13:30	Kildehenvisning og korrektur.
		17:00-19:30	Design presentasjonsapplikasjon
ons, 11.05.2016	7	09:00-12:00	Implementasjon av presentasjonsapplikasjon.
		15:00-19:00	Implementasjon av presentasjonsapplikasjon.
tor, 12.05.2016		09:00-12:00	QA problemgenerator + realistiske datasett
		13:00-16:00	javadoc + sonarLint
fre, 13.05.2016	5	09:00-10:00	Møte med veileder.
		12:00-16:00	Måloppnåelse + videre arbeidd
lør, 14.05.2016	9	09:00-12:00	Videre arbeid + QA
		12:00-16:00	QA + gruppeevaluering
		21:00-23:00	Alternativer til Optaplanner + korrektur

søn, 15.05.2016	8	09:00-16:00	Diverse Rapport
		21:00-22:00	Bruk av tid og distanse

Timelogg – Helge Eriksen

Uke 0

				fredag 8-1/16: 12.00 - 15.00 skrev ut og signerte prosjektavtale snakkett litt overordnet om hvordan vi ser for oss at oppgaven skal løses, hvordan skal algoritmen kalles av tredjepartsprogrammer		

Uke 1

Mandag 11.01.16 10.00 - 12.00 skrivekurs 12.00 - 13.00 snakket med Ivar om omfanget av oppgaven og avtalte at vi skal møtes tirsdager klokken 9 han anbefalte gira som planleggings verktøy. 13.00 - 15.00 sendte mail til ETC om møte. Diskuterte hvordan algoritmen skal utvikles begynte så smått å se på hvilke GIS verktøy vi skal benytte, - så på OsmSharp og ThinkGeo 17.00 - 19.00 Leste igjennom SU-oppgaven vår og Ruteplanlegger bacheloroppgaven	Tirsdag 12.01.16: 09.00 - 13.00 startet arbeidet på prosjektplan. har sett på prosjektmål,fagområde, avgrensning, oppgavebeskrivelse, ansvarsforhold og roller, rutiner og regler i gruppa 13.00 - 14.30 møte med etc 14.30-16.00 lagde/fylte ut møtereferat	Onsdag 13.01.16 09.00 - 11.00 arbeidet videre på prosjektplan skummleste litt av Contract Manager bacheloroppgaven fra i fjor. 11.00 - 14.30 Så på hvilke verktøy vi skulle bruke for å lage gantt diagram, valgte project professional 2016 fra microsoft. Lagde gantt diagram. 14.30-15.00 arbeidet videre på prosjektplan	Torsdag 14.01.16 09.00 - 15.30 arbeidet på prosjektplan lurer på RUP vs Scrum 15.30-16.00 forberedet oss til møte med hjemmesykepleien.	Fredag 15.01.16 12.00 - 14.00 Rup vs Scrum RUP 14.00-15.00 leste på C#		

Uke 2

Mandag 18.01.16 09.00 - 16.00 jobbet videre med prosjektplan, mer spesifikt valg av SU-modell og hvordan den skal brukes	Tirsdag 19.01.16 09.00-10.00 forberedet oss til møte med veileder 10.00-11.00 møte med veileder delte trello og prosjektplan, skal dele bitbucket greit at vi ville bruke arbeidsmetodikken til RUP og beskrive lett variant av RUP 11.00 - 11.30 skrev møtereferat fra møte med veileder. skal vi levere dette med bachelor oppgaven? i så fall må vi begynne å sende referatene til alle som har vært med på møtet. 11.30 - 13.00 planla møte med oppdragsgiver skrev ut og signerte prosjektavtale 13.00 - 14.00 møte med oppdragsgiver 14.00-14.30 skrev møtereferat 14.30 - 16.00 oppdaterte gantt diagrammet.	Onsdag 20.01.16 09.00-13.00 fullførte prosjektplan kladd klar til gjennomgang av veileder og oppdragsgiver 13.00-14.30 planla møte med hjemmesykepleien 14.30-16.00 begynte på kravspek, use-case diagram	Torsdag 21.01.16 09.00-10.00 forberede møte og reise til møtet 10.00-12.00 møte med hjemmesykepleien. se møtereferat. 12.00-14.00 skrive møtereferat 14.00-16.00 kravspek	Fredag 22.01.16 13.00 - 15.00 leste meg opp på routing vehicle problem wirth time windows begynte å tenke på vår egen.		

Uke 3

Mandag	Tirsdag	Onsdag	Torsdag			Søndag
25.01.16 09.00-13.00 Arbeidet videre med kravspek 13.00-14.00 møte med ETC 14.00-15.00 skrev møtereferat 15.00-16.00 algoritme resech	26.01.16 09.00-10.00 forberede møte med veileder og jobbet med retting av prosjektrapport 10.00-11.30 Møte med veileder og skrev møtereferat 11.30-13.30 karrieredag 13.30-16.00 jobbet med å rette på prosjektplan Fullførete prosjektplan	27.01.16 09.00 - 12.00 Ferdigstilte og leverte prosjektplan og prosjektavtale 14.30 - 18.30 Husk å nevne i oppgaven: Hva er NP - hard problems definer hva problemet er heuristics	28.01.16 09-00-16.00 laget Domenemodell og utbedring av krav.			21.01.16 19.00 - 23.30 gjorde små C# oppgaver jeg fant på internett, tilslutt lagde jeg Tic Tac Toe. Og så noen videoer på youtube

Uke 4

Mandag	Tirsdag	Onsdag	Torsdag			
01.02.16 09.00-13.00 arbeidet med ikke-funksjonelle krav og XML-data inn til algoritmen 13.00-16.00 Valg av kart og routing tjeneste. prøvde ut og fikk osmsharp til å/ delvis til å fungere	02.02.16 09.00-13.00 hvordan geokode? osmsharp geokoding og routing gmap 13.00-14.00 møte med ETC 14.00 -15.30 Tegne dopliyment view og skrive møtereferat 17.30 - 23.00 skrev om et traveling salesman problem eksempelet i java til C# fra NB! javakoden er feil, den gir ikke riktig svar, har skrevet om C# koden så den gir riktig svar.	09.00 - 10.00 forberedet møte med veileder 10.00 - 11.00 møte med veileder 14.30 - 16.30 prøvde å få osmsharp kartfunksjonen til å funke, mislyktes. 18.30 - 20.30 skrev på kravspek	04.02.16 09.00 - 15.00 Detaljert kravspesifikasjon lagde datasett til algoritmen i JSON			

Uke 5

Mandag		Onsdag	Torsdag			Søndag
08.02.16 14.00 - 20.00 skrev kravspek, absolutte og relative kriterier, datasett og forbedret det jeg har skrevet på ikke-funksjonelle kravfaTirsdag 09.02.16 08.00-09.00 satt meg inn i sigurdkode 09.00- 12.00 skrev mer kravspek 12.00-13.00 forberedet møte med etc 13.00-13.30 møte med ETC 13.30-14.00 skrev møtereferat. 14.00-16.00 rettet på kravspekk etter forslag fra etc. sette opp designdokument-mal		10.02.16 09.00-10.00 forberede møte med veileder 10.00-11.00 møte med veileder 11.00-16.00 skrev små endringer på kravspekk etter forslag fra Ivar. Lagde deployment View og endret litt på oversiktsdiagram for at de skal stemme overens.	11.02.16 09.00 - 10.00 algoritmedesign 10.00-12.00 valg av kart og routing tjeneste. fant info om here og mer om OSM 15.00 - 17.00 leste på løsningsmetoder			21.02.16 13.00 - 18.00 Jobbet med ymse

Uke 6

Mandag	Tirsdag	Onsdag	Torsdag	Fredag		Søndag
15.02.16 11.00 - 19.00 prøvde å sette meg inn i ANT. skrev ned endel av tankene jeg kom på rundt problemstillingen.	16.02.16 09.00 - 10.00 forberedelse møte veilder. 10.00-11.00 møte med veileder 11.00-13.00 forberedelse møte ETC 13.00 - 13.30 gikk til ETC men møte avlyst 13.30-16.00 satt oss inn optaplanner eksempler	17.02.16 09.00 - 16.00 leste seg opp på optaplanner, så en video på youtube og prøvde å sette seg inn i kode. Det ser veldig lovende ut for at dette kan opfylle alle våre krav.	18.02.16 12.00-16.00 optaplanner lab	19.02.16 13.00 - 14.00 forberedet møte med etc 14.00-15.00 møte med etc 15.00-15.30 skrev møtereferat.		21.02.16 18.00-19.30 gjorde lab 201-203

Uke 7

Mandag	Tirsdag	Onsdag	Torsdag	Fredag		
22.02.16 09.00-12.00 jobber med å modelere designdokument og satt opp java repo. 13.00-15.30 leste kapittel 4 i optaplanner og prøvde å lage utkast til designmodell 15.30-19.30 så på JSON parsing i java og fikset gitignore.	23.02.16 09.00-10.00 planlegge møte med veileder og dagen videre. 10.00-11.00 møte med veileder og skrev møtereferat 11.00-13.00 lagde klassediagram for initiell fordeling 13.00-16.00 jobbet med å forstå gson og lage POJOs 16.00 - 17.00 så ferdig en video på youtube 17.00 - 18.00 leste mer optaplanner dokumentasjon, se nærmere på chapter 2 quick start. 18.00-19.00 lagde ferdig POJOene NB! assignments with tasks with competance requirement might be overkill. CompetenceList and CompetanceType as the same or is there a need for two different? 19.00-19.30 la inn mer optaplanner relevant stoff i koden vår	24.02.16 11.00-12.00 jobbet videre med kode	25.02.16 09.00 - 12.00 skrevet på designdokument. 12.00-14.00 skrev generateDepot() kode 14.00 - 15.00 skrev mer på designdokument.	26.02.16 09.00 - 10.00 leste på optaplanner og fant ASSERT_FULL 12.30-15.00 jobbet med JsonInDataTest		

Uke 8

Mandag	Tirsdag	Onsdag	Torsdag	Fredag		
29.02.16 09.00-16.00 Skrive designdokument: algoritmens ulike moduser	01.03.16 09.00-12.00 skrev på designdokument/optaplanner 12.00-13.00 forberedet møte med ETC 13.00-13.30 møte ETC 13.30-14.00 skrev møtereferat 15.30-18.30 javadoc	02.03.16 06.00-07.30 skrev javadoc 09.00 - 12.00 skrev på designdokumentet 12.00-13.30 Endret JSON data og tilsvarende klasser 13.30-16.00 skrev JsonTest og jobbet litt med problem generator	03.03.16 08.00-13.30 implimenterte problemgenerator sammen med optaplanner og skrev drools rules 15.00-18.30 parallellisering av avstandskalkulering	04.03.16 12.00-15.00 jobbet med å samle kode og skrev videre på kode		

Uke 9

Mandag	Tirsdag	Onsdag	Torsdag	Fredag		
07.03.16 09.00-12.00 skrev NoCompetenceAvailable rule 12.00 - 14.00 leste litt om vrp	08.03.16 09.00-10.00 forberedet møte med veileder og planla uka 10.00-11.00 møte med veileder. 11.00-11.30 skrev møtereferat og møtekladd til ETC møtet. 11.30-13.00 leste om shadow variables 13.00-13.30 møte med ETC 13.30-14.00 skrev møtereferat 17.00-19.00 shadow variables chaining eksempelet	09.03.16 09.00-12.00 jobbet med design av Assignment distribution phase 12.00 - 13.00 skrivekurs med Frode Haug 13.00 - 16.00 jobbet videre med design av route distribution phase	10.03.16 09.00 - 12.00 jobbet videre med design av route distribution phase 12.00 - 16.00 gjorde store endringer på pakkenavn og struktur	11.03.16 12.00-15.00 jobbet med koding av Routedistribution phase		

Uke 10

Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	
14.03.16 09.00-13.30 jobbet med koding av Routedistribution phase 19.30 - 20.30 skrev om fairness rule i Routedistribution phase	15.03.16 09.00-10.00 planla uka og forberedet møte med veileder 10.00-11.00 møte med veileder. 11.00-13.00 skrev møtereferat, javadoc og forberedet møte med ETC 13.00-13.30 møte med ETC 13.30-14.00 skrev møtereferat fra møte med ETC 14.00 - 16.00 jobbet på fase 2 klassediagram og implementerte RouteDistributionEmployee	16.03.16 09.00 - 12.00 jobbet på fase 2 klassediagram og skrev om dette i designdokument 12.00-14.00 X-Session 14.00-14.30 litt mer design, jobbet litt på valg av karttjeneste.	17.03.16 09.00-10.00 skrev ferdig om kartverket 10.00-10-30 skrev videre på designdokument av fase to regler 10.30-12.00 tegnet diagram over arvestruktur 12.00-14.00 jobbet mer med tegning av utvidet arvestruktur med representasjoner av klassemedlemmer	18.03.16 10.00-11.00 Jobbet med ymse	19.03.16 10.00-12.00 prøvde å få inn depot som planningvariable range fant mye som ikke kan gjøres, var ikke så heldig med hva som kan gjøres 12.00 - 13.00 så på optaplanner planning value og value range	

Uke 11
21.03.16 - 28.03.16
Påskeferie

Uke 12

	Tirsdag	Onsdag	Torsdag	Fredag		
	29.03.16 09.00-12.00 Javadoc 12.00 - 16.00 så på hvordan planlegge variable turer til depoet	30.03.16 dødsfall i familien	31.03.16 Jobbintervju	01.04.16 12.00-14.00 gikk igjennom endringene Sigurd har gjort og diskuterte de.		

Uke 13

Mandag	Tirsdag	Onsdag	Torsdag			Søndag
04.04.16 Syk	05.04.16 09.00-10.00 oppdaterte meg på hva sigurd har gjort og planla møte med veileder. 10.00 - 11.00 møte med veiled 11.00-11.30 møtereferat 11.30-12.00 forberede møte med ETC 12.00-13.00 arbeide med repo til presentasjonsapp 13.00-14.00 møte med ETC 14.00-14.10 skre møtereferat 14.10 - 15.00 gjenoprettet drl rules mistet fra dårlig mergfix 15.00-16.00 jobbet med å legge hasDepotVisit inn i Visit interface	06.04.16 09.00-11.00 endret på avstandskalkuleringene slik at de ikke lenger kalkulerer unødvendige avstander. 11.00 - 16.00 skrev drools rules. VehicleMustHaveCompetence tar enda ikke høyde for at ulike ansatte i samme bil sammen kan oppfylle kompetansekravet eller at ansatte kanskje ikke er på jobb. begge disse har problemer med at de ligger i lister	07.04.16 09.00-10.30 jobbet videre med VehicleMustHaveCompetence la til sjekk om de er på jobb. 10.30-16.30 jobbet med å forbedre depotGenerator 19.30 - 21.00 fikset bug i depoGenerator hvor bare første ansatt fikk førerkort. 21.00 - 24.00 la til flere skift per ansatt idetpotGenerator fikset flere bugs i depotGenerator Jobbet videre med depotGeneratorTest			10.04.16 16.00 - 23.00 skrev en VehicleMustHaveCompetence regel som sjekker alle ansatte på jobb i bilen. skrev en regel VehiclesMustHaveLegale-NumberOfPassangers regel som sjekker at det ikke er for mange ansatte i en bil av gangen.

Uke 14

Mandag	Tirsdag	Onsdag	Torsdag			Søndag
11.04.16 09.00 - 12.00 endret på VehicleMustHaveCompetence regelen slik at den muligens fungerer som den skal, og tilegner - score tilsvarende manglende kompetanse 12.00 - 12.30 fikset feil i depotGenerator hvor en ansatt kunne være på jobb samtidig med seg selv 12.30 - 15.30 fikset en bugg i VehicleMustHaveCompetence som fjernet assignments sine competence requirements. Tror også vehicleMustHaveLegalNumberOfPassangers rule skal fungere nå 15.30 - 16.00 la inn en tom routeDistributionPhaseCompatibility.drl fil og så på alt det harde arbeide sigurd har gjort. 18.30 - 20.00 laget routeDistributionPhaseCompatibility rule	12.04.16 09.00 - 10.00 oppsummering av tidligere arbeid og planlegge møte med veileder. 10.00-10.50 møte med veileder 10.50 - 11.30 skrev møtereferat 11.30 -12.30 la inn routeDistributionPhaseEmployeesRequiredPerAssignmentScoreRules.drl fil 12.30 - 13.00 forberedet møte med ETC 13.00-14.30 møte med ETC 14.30 - 15.30 skrev møtereferat fra møtet med ETC og diskuterte implikasjonene av møtet og arbeidet videre. 15.30 - 16.00 forsøkt å løse et problem med ssh key i bitbucket hos sigurd	13.04.16 09.00-11.00 la til mulighet for endring av variabler i depot Generator 11.00 -11.20 funderte på shadow variable i bil med ansatte, gjerne i hashmap med tid som key, men dropper det intill videre 11.20 -11.30 implementerte en EnoughEmployeesPerAssignment rule. 11.30 -12.00 begynte så smått på qa 12.00-16.00 så litt på og troubleshoota ymse bugs	14.04.16 09.00-10.30 la inn funksjonalitet for å redusere antall ansatte og biler brukt. 10.30 - 11.00 troubleshoota presentasjonen 11.00 - 15.00 forbedret competence og compatibility reglene 15.00-16.30 forsøkte å se på om reglene faktisk har blitt bedre			17.04.16 18.00 - 19.00 Ymse jobb

Uke 15

Mandag	Tirsdag	Onsdag	Torsdag	Fredag		
18.04.16 09.00- 11.00 diskuterte fremgang, avtalte møte med hjemmesykepleien og så på dukumentasjon. 11.00 - 15.00 skrev på designdokument Fordele oppdrag i ruter per depot forenklet 15.00 - 15.30 så på å endre vehicleShouldHaveCompatibility regelen for å unngå score trap, må se videre. 20.00 - 23.30 sliter med å finne ut av vehicleShouldHaveCompatibility og vehicleMustHaveCompetence hvordan fungerer from i drools ?	19.04.16 09.00-10.00 statusoppdatering, planla uken og møtet med veileder 10.00-11.15 møte med veileder 11.15 - 13.00 skrev møtereferat, installerte og ordnet word og forberedet møte med ETC 13.00 - 14.00 møte med etc 14.00 - 16.00 ordnet med samskriving i word og begynte arbeidet med å oprette skjellettet.	20.04.16 09.00-13.30 arbeid med overblikk av rapporten og la inn endel på innledning 13.30 -15.00 la inn endel på kapittel 1.7 18.00 - 20.00 endret endel på kapittel 1.7 20.00-21.00 eclipse vil ikke åpne seg så da blir det ett nytt ett.	21.04.16 08.30-12.00 skrev om RUP fasene 12.00 - 13.00 sturet med delkapittelet kalt hovedinndelingen av prosjektet 13.00 -16.00 forsøkte å fiksen drools reglene for compatibility og competence er inne på noe men mangler litt 16.00 -18.00 laget en funksjon for å skrive ut competence needed og vehicle competence link med veldig god beskrivelse for hvordan operasjoner som kan gjøres i drools competence var aldri gal problemet var at ansatte ikke er på jobb når noen av oppgavene skulle gjennomføres etter forslag fra optaplanner. 18.00 - 20.00 la inn en stygg funksjon for å se hvilken compatibilities som mangler, funksjonen ser ut til å fungere som den skal.	22.04.16 12.00 - 15.00 planla møte med hjemmetjeneste n, gikk igjennom arbeid som må gjøres og så på å lage en egendefinert rute som hjemmetjeneste n kan løse		

Uke 16

Mandag	Tirsdag	Onsdag	Torsdag		Lørdag	Søndag
25.04.16 09.00 - 13.00 arbeidet med dokumentet, forberedet møte med hjemmetjenesten. 13.00 - 14.30 møte med hjemmetjenesten 14.30-15.00 skrev møtereferat	26.04.16 06.00-07.30 skrev ned beslutningspunkter 09.00 - 12.30 skrev på oppgaven 12.30 -13.00 forberedet møte med ETC 13.00 - 13.40 møte med etc 13.40-14.00 skrev møtereferat	27.04.16 07.00-09.00 skrev litt på oppgaven og oppdaterte diagrammer 09.00-12.00 endret på diagrammer og prøvde å få orden i design 12.00-13.00 oprettet diagram for pipeandfilter 13.00 - 15.00 så på muligheten for å implimentere bare en fase 15.00 - 16.00 testet ny implementasjon	28.04.16 09.00 - 13.00 design og disusjon. 13.00-15.00 videre diskusjon		30.04.16 22.00 - 24.00 så på hvordan lage en branch i git og hvordan lage en PlannerBenchemark. Skal prøve å implimentere i morgen.	12.00 - 17.30 jobber med å implementere benchmark. simulated annealing er gal. alle andre krever også spesielle move operations. 17.30-20.00 begynte å skrive litt på QA 20.00-22.00 jobbet med å få solvers lagret, kan ikke bruke gson automatisk, skal prøve å gjøre den serializable

Uke 17

Mandag	Tirsdag	Onsdag	Torsdag	Fredag		
02.05.16 09.00 - 15.00 skrev på oppgaven, hovedsakelig kapittel 7, men også litt på diskjusjon 18.00 . 21.00 laget en fornuftig BenchsoulutionIO og la inn 3 datasett og 3 solvers.	03.05.16 09.00 - 10.00 diskuterte fremgang med sigurd og planla møte med veileder 10.00 - 11.00 møte med veileder. 11.00 - 15.00 skrev på imlementasjonskapittelet 17.00 - 18.30 fikset DepotMustHaveCompetence rule og lastet opp dette og benchmark opplegget. 18.30 - 20.00 jobbet videre med implementasjon	04.05.16 09.00 - 15.00 skrev på implementasjonskapittelet 20.30 - 21.30 fjernet ubrukte funksjoner i kodebasen	05.05.16 09.00-14.30 skrev på implementasjon 16.00 - 19.00 Benchmark	18.00-19.00 bechmark, tabu gir alltid best til den siste som kjører.		

Uke 18

Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag
09.05.16 09.00 - 16.00 godkjent fravær jobbintervju 18.00 - 19.00 dokumentasjon og så på arbeidet sigurd har gjort	10.05.16 09.00-10.00 statusoppdatering, planlegging av møte med veileder og fremdrift. 10.00 - 10.30 lite møte med veileder hvor vi avtalte nytt møte til fredag. 10.30 . 14.00 skrev videre på dokumentasjon.	11.05.16 07.00 - 09.00 tok igjen tapt tid fra i går. jobbet med å forbedre design 09.00 - 12.30 skrev på design 12.30-17.00 urnenedsettelse 17.00-22.00 design og kravsapek og leser igjennom hele dokumentet	12.05.16 09.00-16.00 implementasjon og benchmark	13.05.16 09.00 - 10.00 møte med veileder. 12.00 - 18.00 skrev på måloppnåelse ymse endringer. 19.00 - 20.00 skrev videre på måloppnåelse og gikk igjennom noen av kommentarene til veileder	14.05.16 09.00-14.30 gikk igjennom hele dokumentet og forbedret leslighet. 15.30 - 20.00 fortsatte gjennomgangen av dokumentet	15.05.16 09.00-16.30 gjennomgang og forbedring av dokument 19.00-20.300 javadoc og så på timebruk

Uke 19

Mandag						
16.05.16 09.00-15.00 ferdigstilte rapport 17.00 -22.00 fortsatte og ferdigstille rapport						