Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for ingeniørvitenskap
Institutt for bygg- og miljøteknikk

Reza Ariyan

Implementering av parametrisk optimalisering i praksis: dimensjonering og valg av bæresystemer i bygninger

Masteroppgave i Prosjektering av konstruksjoner
Veileder: Professor Anders Rønnquist
Juni 2019

NTNU
Kunnskap for en bedre verden

SWECO

Reza Ariyan

Implementering av parametrisk optimalisering i praksis: dimensjonering og valg av bæresystemer i bygninger

Masteroppgave i Prosjektering av konstruksjoner
Veileder: Professor Anders Rønnquist
Juni 2019

Norges teknisk-naturvitenskapelige universitet
Fakultet for ingeniørvitenskap
Institutt for bygg- og miljøteknikk

NTNU
Kunnskap for en bedre verden

Institutt for konstruksjonsteknikk
Fakultet for ingeniørvitenskap
NTNU- Norges teknisk- naturvitenskapelige universitet

TILGJENGELIGHET

Åpen

MASTEROPPGAVE 2019

FAGOMRÅDE:	DATO:	ANTALL SIDER:
Prosjektering av konstruksjoner	24.06.2019	69

TITTEL:

Implementering av parametrisk optimalisering i praksis: dimensjonering og valg av bæresystemer i bygninger

Implementation of parametric optimization in practice: design and selection of structural systems in buildings

UTFØRT AV:

Reza Ariyan

SAMMENDRAG:
Automatisering av arbeidsprosesser i byggebransjen har i de siste 10-15 årene vært i stor utvikling, der effektiviteten av gjenbruk av informasjon er enda et nytt begrep i industrien. Det viser seg at optimalisering av konstruksjoner i bygninger som oftest forekommer for sent for å ha noen real påvirkning på den endelige konstruksjonen.

Formålet med denne oppgaven er å demonstrere de praktiske mulighetene for automatisering og optimalisering av bygninger i et parametrisk miljø. Dette ved tre foreslåtte designprosessmetoder, som skal kunne brukes til utvikling av et mer intelligent design. Til dette benyttes det visuelle programmeringsverktøyet Dynamo, sammen med en supplerende konstruksjonsanalyse i Robot structural analysis.

De tre foreslåtte designprosessmetodene benyttes til utvikling av en beregningsmetode for optimalisering av et dekkesystem for et gitt rektangulært areal. Utarbeidelsen av denne optimaliseringsprosessen har bakgrunn i Swecos ønske om å kunne utforske flere optimale dekkesystem-alternativer, med begrenset ressursbruk.

Optimalisering av dekkesystemet benytter algoritmiske optimaliseringsteknikker, som arbeider mot å finne de mest optimale løsningene. Dekkets optimaliseringsgrad vurderes ut ifra blant annet konstruksjonsvekt, og mengde lokale armerigsforsterkninger i dekket.

Den utviklede parametriske beregningsmetoden er testet for to typer oppbygning av dekkesystemer. Basert på et Brute force search og en genetisk algoritmisk optimaliseringsprosess, har beregningsmetoden lykkes med å identifisere de optimale løsningene.

FAGLÆRER: Professor Anders Rønnquist

VEILEDER(E): Anders Rønnquist og Jan Ove Vinjevoll

UTFØRT VED: Institutt for konstruksjonsteknikk, Norges teknisk-naturvitenskapelige universitet

Fordord

Denne oppgaven er utarbeidet som siste ledd i gjennomføringen av min toårige mastergrad i Bygg- og miljøteknikk ved Norges teknisk-naturvitenskapelige universitet. Oppgaven utgjør 30 studiepoeng innen emnet TKT4198 - Prosjektering av konstruksjoner.

En stor takk rekkes til min veileder i Sweco, Jan Ove Vinjevoll, og avdelingsleder, Bård Hansen, for veiledning og tilrettelegging av nødvendig utstyr og verktøy for gjennomføring av oppgaven. I samme anledning takker jeg også hovedveileder ved NTNU, Anders Rønquist, for å veilede prosessen i avhandlingen og gi nye synsvinkler, samt konstruktiv tilbakemelding gjennom hele prosessen.

En spesiell takk til Autodesk University og Dieter Vermeulen for motiverende konferanser, som denne oppgaven er inspirert av.

Sammendrag

Automatisering av arbeidsprosesser i byggebransjen har i de siste 10-15 årene vært i stor utvikling, der effektiviteten av gjenbruk av informasjon er enda et nytt begrep i industrien. Det viser seg at optimalisering av konstruksjoner i bygninger som oftest forekommer for sent for å ha noen real påvirkning på den endelige konstruksjonen.

Formålet med denne oppgaven er å demonstrere de praktiske mulighetene for automatisering og optimalisering av bygninger i et parametrisk miljø. Dette ved tre foreslåtte designprosessmetoder, som skal kunne brukes til utvikling av et mer intelligent design. Til dette benyttes det visuelle programmeringsverktøyet Dynamo, sammen med en supplerende konstruksjonsanalyse i Robot structural analysis.

De tre foreslåtte designprosessmetodene benyttes til utvikling av en beregningsmetode for optimalisering av et dekkesystem for et gitt rektangulært areal. Utarbeidelsen av denne optimaliseringsprosessen har bakgrunn i Swecos ønske om å kunne utforske flere optimale dekkesystem-alternativer, med begrenset ressursbruk.

Optimalisering av dekkesystemet benytter algoritmiske optimaliseringsteknikker, som arbeider mot å finne de mest optimale løsningene. Dekkets optimaliseringsgrad vurderes ut ifra blant annet konstruksjonsvekt, og mengde lokale armerigsforsterkninger i dekket.

Den utviklede parametriske beregningsmetoden er testet for to typer oppbygning av dekkesystemer. Basert på et Brute force search og en genetisk algoritmisk optimaliseringsprosess, har beregningsmetoden lykkes med å identifisere de optimale løsningene.

Summery

Automation of the workflow in the construction industry has been in great development for the last 10 to 15 years. The efficiency of the re-utilization of information is one of many areas that are still not optimal in the industry. The optimization of the building design usually occurs too late, making it costly or impractical to implement some of the changes to the final design.

The purpose of this thesis is to demonstrate the possibilities for automation and optimization of building design, early, in a parametric environment. In this thesis, an exploration of three design processes is undergone, with the focus on more intelligent design. Dynamo, a visual programming software, along with Robot Structural analysis, a FEA program, are used as tools to aid the design processes.

Furthermore, a calculation method for optimizing the structural slab system, for a given area, is developed with the basis of the three design processes. The calculation proposal uses algorithmic optimization techniques, working towards a solution with the lowest structural weight and required reinforcement for each slab.

The proposed calculation method has proven efficient and is able to explore several slab systems with limited use of resources. Two types of slab systems have been successfully tested, indicating an optimal solution based on a Brute Forcesearch and agenetic algorithmic optimization process.

Innhold

Tabeller

Figurer

Forkortelser

BIM	=	Building Information Modeling
RSA	=	Robot Structural Analysis
VP	=	Visual Programming
FE	=	Finite Element
FEA	=	Finite Element Analysis
FEM	=	Finite Element Method
ULS	=	Ultimate Limit State
SLS	=	Serviceability Limit State
SOO	=	Single objective Optimization
MOO	=	Multi Objective Optimization
GA	=	Genetic Algorithm
MOEA	=	Multi Objective Evolutionary Algorithm
NSGA-II	=	Non-dominated Sorting Genetic Algorithm II
API	=	Application Programming Interface
RAM	=	Random Access Memory
Plugin	=	En programvareutvidelse (plugin eller addin) er en tilleggsmodul som er utviklet for å tilby ekstra funksjonalitet til et eller flere programmer.
COM-teknologi	=	Component Object Model-teknologi er en delingsmetode av moduler mellom to programmer utviklet av Microsoft i 1993.
.NET framwork	=	En programmingsplatform utviklet av Microsoft som kjører hovedsaklig på Microsoft Windows.
Open source	=	For at programmet skal være open source, må det inneholde kilde kodene og tillate distribusjon i kilde kodene.

Kapittel 1

Introduksjon

Ideen om å automatisere arbeidsprosesser mellom de ulike aktørene i byggebransjen har de siste 10-15 årene vært i stor utvikling gjennom Building Information Modeling (BIM). BIM har blitt introdusert på flere områder som et verktøy for automatisert datautveksling og gjenbruk av informasjon for aktørene i byggeprosjekter. Effektiviteten ved å kunne gjenbruke informasjon er enda et nytt begrep i industrien og ikke optimal, da det viser seg at analyser og optimalisering som oftest forekommer for sent for å få noen real påvirkning på konstruksjonens utforming (Cavieres et al., 2011). Dette kan føre til kostbare prosjektendringer og redusert antall ulike løsninger.

I forprosjekteringsfasen av et byggeprosjekt blir det i gjennomsnitt vurdert 2,7 forskjellige bygningsdesigner og bæresystemer. Dette skyldes blant annet prosjekters tidsbegrensninger og begrensede kostnadene til å utforske flere designalternativer (Biilmann, 2015). Ved innføring av automatisering i form av parametrisk design i tidlig prosjekteringsfase, vil det være mulig å analysere flere hundre designalternativer (Biilmann, 2015). Økningen av evalueringsmodeller gir mulighet til å kunne utarbeide en mer optimal modell i et tidlig prosjekteringsstadie, samtidig som risikoen for endringer blir redusert ved å ta tidlige beslutninger basert på mer informasjon.

Fokuset i denne oppgaven er å utvikle modeller og verktøy som kan synliggjøre praktisk bruk av parametrisk optimalisering for byggingeniører i en tidlig prosjekteringsfase. Dette gjøres ved å utvikle en parametrisk modell, med hensikt om å kunne automatisere en av hverdagslige oppgavene til en byggingeniør.

1.1 Mål

Raporten skal kunne bidra til utvikling av bærekraftige bygg, ved å belyse viktigheten av å benytte optimaliseringsprosesser i et tidlig prosjekteringsstadige.

Effekt mål

- Bevistgjøre mulighetene for praktsik bruk av parametrisk design for byggingeniører.
- Automatisering av en algoritmisk optimaliseringsprosess som kan benyttes av Sweco i en forprosjekteringsfasen, for indikering av optimale dekkesystemer.

Resultat mål

- Utvikling av en parametrisk analysemodell som inkluderer FE via Dynamo i RSA.
- Gjennomføring av FEA for en parametrisk modell via Dynamo i RSA.
- Bruk av forskjellige materialer i analysen, og parametrisk definering av tverrsnittet til FE.
- Automatisering av rangering og evalueringsprosesser, for modeller med forskjellige materialer og elementer.
- Genetisk algoritmisk optimalisering av et dekkesystem og visualisering av resultatene.
- Automatisering av prosessene slikt at det kan gjenbrukes.

Kapittel 2

Bakgrunn

Gjennom dette kapittelet presenters bakgrunnsteori for verktøyene og optimaliseringsteknikkene benyttet til utvikling av denne masteroppgaven. Kapittelet starter med en generell beskrivelse av alle verktøyene og samspillet mellom disse. Videre blir optimaliseringsmetodene benyttet ved bruk av disse verktøyene presentert. Avslutningsvis er oppbyggingen av, og bakgrunnen for, optimaliseringsmetodene i et parametrisk miljø som modellene i denne oppgavene baseres på vist og forklart.

2.1 Verktøy

Velkjente programvarer har blitt benyttet til gjennomføring av denne oppgaven. En oversikt over disse er vist i Tabell 2.1.

Programvarer	**Versjon**
Autodesk Robot Structural Analysis Professional	2019
Dynamo	2.0.2.6826
Microsoft Excel	1808
Python	2.7

Tabell 2.1: Programvare liste

2.1.1 Autodesk Robot Structural Analysis

Autodesk Robot Structural Analysis Professional (RSA) er et FEM-basert beregningsprogram utviklet av Autodesk co-operation. I tillegg er både Revit og Dynamo utviklet av Autodesk, noe som gjør at samhandlingen mellom disse tre programvarene blir enklere.

RSA benyttes hovedsakelig sammen med RSA application programming interface (API) til optimalisering og automatisering via funksjonalitetene som er tilgjengelige i RSA API. Tilgang til en åpen API gjør det mulig å automatisere en enkelt, eller flere repeterende oppgaver, og parametrisk styring av RSA. RSA API benytter Microsofts komponentobjektmodell (COM)-teknologi som gjør det mulig å lage en .NET Framework til å kunne styre RSA analysen (Fisher and Shorma, 2010).

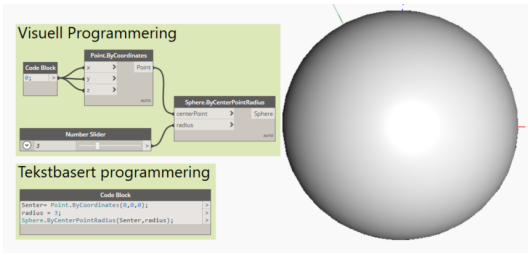
2.1.2 Dynamo

Det visuelle programmeringsverktøyet Dynamo ble i 2012 lansert som en tilleggspakke for BIM ved siden av Revit API og .NET Framework (Tammik, 2016). Siden lanseringen av Dynamo har antall brukere vokst kraftig, med hele 1,4 millioner installasjoner, 638 utviklere og 1 115 tilleggspakker. I dag følger Dynamo som en integrert del av Revit 2019. Dynamo finnes i to versjoner: Dynamo og Dynamo Studio. Dynamo Studio er en frittstående applikasjon uten tilkoblinger til Revit API (Autodesk, 2018a). Denne versjonen blir ikke benyttet i denne oppgaven.

Analysene som blir utført i denne oppgaven benytter Dynamo 2.0, utgitt april 2018. Dynamo kan enten kjøres frittstående "*Sandbox* eller som en plugin i en annen programvare som Revit eller Maya. Sandbox er Dynamos kjernefunksjonalitet distribuert på en måte som ikke forstyrrer andre Dynamo-installasjoner (Autodesk, 2018b). Dynamo Sanbox startes fra:

```
C:\Program Files\Dynamo\Dynamo Core\2\DynamoSandbox.exe
```

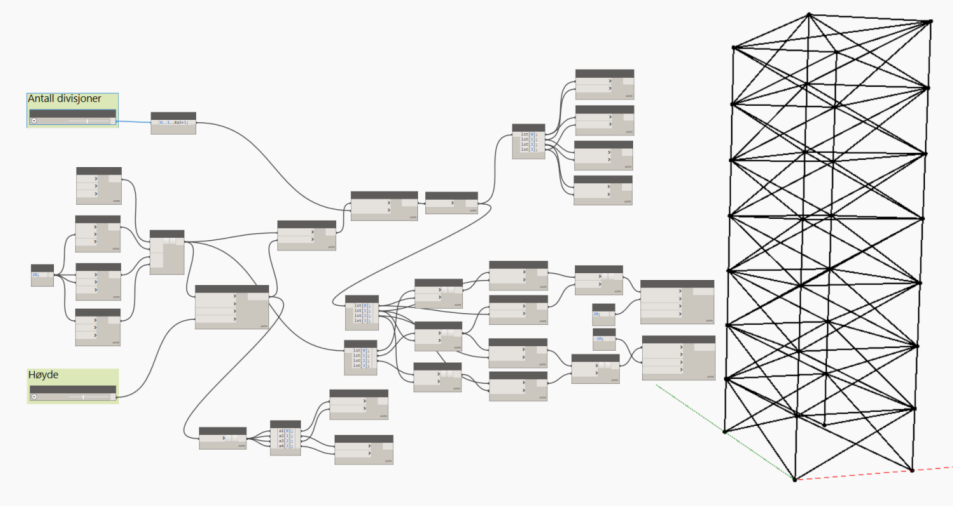
Dynamos utseende og funksjonalitet er preget av programvarer som MaxMSP, Maya Hypergraph og Sprinkhoppere (Kensek and Khan, 2013). Forskjellen mellom visuell programmering (VP) og standard tekstbasert programmering er tilkobling av kodebiter i motsetning til å skrive en kode. Et eksempel på dette kan ses i Figur 2.1, der resultatet av begge er vist til høyre.



Figur 2.1: Eksempel på visuell og tekstbasert programmering

Dynamo-grafer som er utviklet gjennom denne oppgaven utfører i hovedsak arbeidsprosesser som kartlegging av geometri, optimalisering, tilkobling og dataoverføring til andre programvarer. For få en generell beskrivelse og forståelse av Dynamo og VP, er det nyttig å undersøke The Dynamo Premier (Autodesk, 2018a).

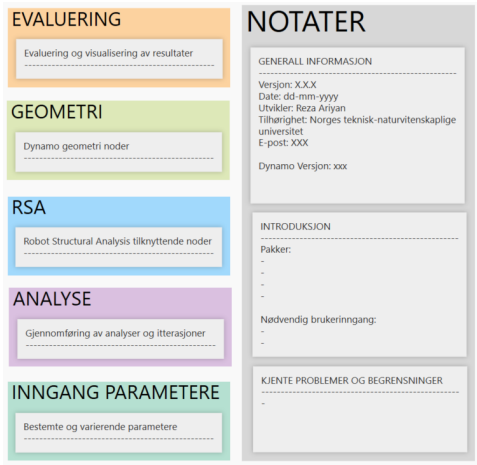
Generelt kan en bruke Dynamo på to forskjellige måter. Den ene er når geometrien blir definert i Dynamo. Slik kan geometrien bli benyttet til å utføre forskjellige analyser av deler eller hele konstruksjonen. Fordelen med denne metoden er evnen til å kunne lage parametriske modeller som kan benyttes til optimalisering. Når passende geometri er funnet, kan den overføres til Revit. Figur 2.2 illustrerer et eksempel av en parametrisk geometri. Konstruksjonen er modellert slik at brukeren enkelt kan justere høyden og antall diagonaler ved bruk av "*Number sliders*".



Figur 2.2: Eksempel på parametrisk geometri modellert i Dynamo

Den andre metoden skiller seg ved å hente Revit-familier og -elementer i Dynamo, fra en eksisterende Revit modell. Denne metoden blir benyttet til å utføre en handling i Dynamo, for deretter å oppdatere det tilbake til valgte elementer i Revit. Denne metoden er nyttig for å endre parametere i store BIM-modeller.

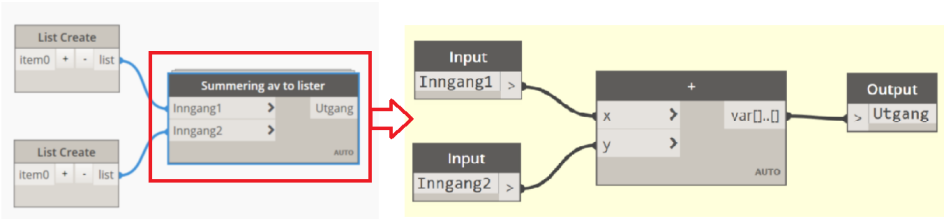
For å kunne organisere og lettere synliggjøre Dynamo-grafer, har Dynamo gitt muligheten til å kunne gruppere og fargekode en rekke noder. I denne oppgaven har funksjonen blitt benyttet til å organisere forskjellige arbeidsprosesser i Dynamo, for å skape en lettere forståelsen av grafene. Den generelle fargekodingen som har blitt brukt er vist i Figur 2.3.



Figur 2.3: Fargekoding og veiledning til Dynamo grafer

Custom nodes

En annen metode for å organisere og redusere en omfattende kompleks Dynamo-graf, er oppretting av egendefinerte noder, kalt "*custom nodes*". Egendefinerte noder fungerer som en mappe for et antall noder, og gjenkjennes av en gul bakgrunn. Et eksempel av en custom node er vist i Figur 2.4.



Figur 2.4: Eksempel på custom node i Dynamo

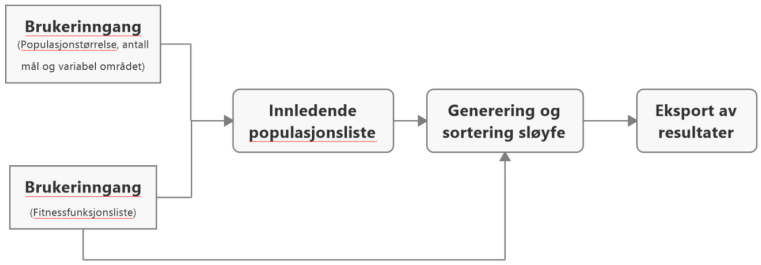
Egendefinerte noder gir muligheten til å kunne bruke flere kopier av en prosess, og lettere gjennomføre endringer som gjenspeiler seg i flere deler av grafen. Oppbyggingen av en egendefinert node kan gjøres på flere måter. For programmerere er det mulig å utvikle *zero touch*-noder fra bunnen av, ved bruk av programmeringsspråket "C#". Disse nodene er låst for endring utenfra. En annen metode som egner seg til ikke programmerere, er å jobbe direkte med Dynamo-brukergrensesnitt for å lage egendefinerte noder fra en samling av eksisterende noder (Autodesk, 2018a).

I denne oppgaven benyttes egendefinerte noder til å konvertere hele modeller til én node, slik at de kan brukes i en optimaliseringsprosess over et gitt antall iterasjoner.

2.1.3 Optimo

Optimo er en *open source*-tilleggspakke for Dynamo, og er et Multi Objektiv Optimaliseringsvektøy (MOO). Dette gjør det mulig for designere å optimalisere BIM-baserte problemer med ett og flere mål. Den nåværende versjonen av Optimo 0.1.2 er basert på den modifiserte MOO-algoritmen Non-dominated Sorting Genetic Algorithm-II (NSGA-II). Dette gjør det mulig å oppnå et sett med optimale løsninger.

Algoritmen fungerer ved å generere et sett av tilfeldige variabler innenfor et gitt område og størrelse av populasjonen, definert av brukeren. Populasjonen som blir laget kan bli kalt for "foreldre", der de optimale foreldrene brukes til å lage et nytt sett av populasjon kalt "barna". For hver generasjon blir de optimale barna valgt som foreldre ved dannelse av nye barn. Denne prosessen fortsetter så lenge stoppkriteriene ikke er nådd. Generelt kan arbeidsprosessen i Optimo beskrives i fem trinn, der disse er illustrert i Figur 2.5.



Figur 2.5: Generelle arbeidsprosessen av Optimo

Brukerinngang
Her skal brukeren definere populasjonsstørrelse, antall mål og variabelt område. Populasjonsstørrelsen er antall tilfeldige verdier opprettet mellom en øvre og nedre grense definert av brukeren, kalt "variabelt område". Presisjonen til resultatene er relatert til økt populasjonsstørrelse, og vil både øke presisjonen og beregningstiden. Antall mål definerer antall fitnessfunksjoner som vil inkluderes i optimaliseringsprosessn. Variablene er parametrenes øvre og nedre grense som definerer alternativene. Fitnessfunksjonsliste er en liste som inneholder alle fitnessfunksjoner laget som én eller flere egendefinerte noder i Dynamo.

Innledende populasjonsliste
Den innledende populasjonslisten er en tilfeldig generert liste fra variablenes øvre og nedre grenser. Denne listen brukes som den første populasjonen av alternativene.

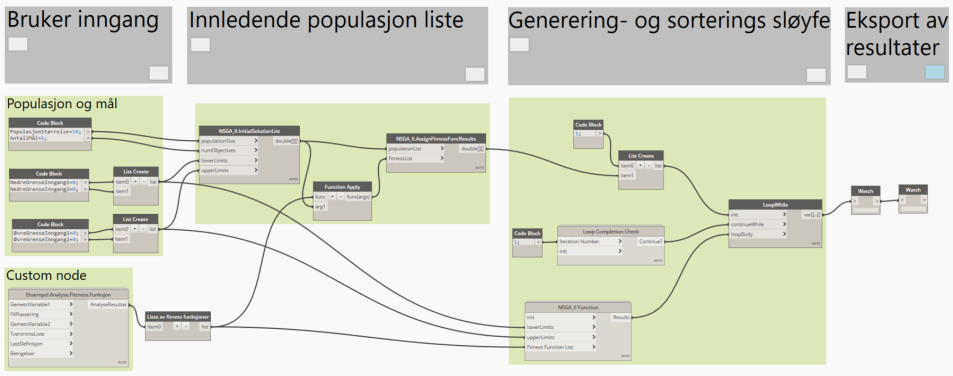
Generering- og sortering sløyfe
Dette er den sentrale delen av Optimo, der generering- og sorteringssløyfen vil lage et gitt antall generasjoner av populasjonen, som bestemtemmes av brukeren.

Eksport av resultater
Det utgående optimale resultatet skjer i henhold til Pareto front, se Kapittel 2.2.1. Resul-

tatene kan visualiseres i en parallellkoordinat-graf eller eksporteres direkte til en BIM-modell.

Den generelle oppbyggingen av Optimo i Dynamo er illustrert i Figur 2.6. Dette oppsettet vil være generelt likt i aller fleste tilfeller, der brukeren definerer inngangene "*populasjon og mål*"og "*fitnessfunksjoner*". Dynamo noden "*Function.Apply* fungerer ved å bruke variablenes verdi fra den første populasjonen i fitnessfunksjonen, for å finne deres målverdi. Denne listen blir kombinert sammen med variablenes verdi i "*AssignFitnessFuncResults*-noden. Alternativene fra innledende populasjonen sortereres etter resultatet fra målverdiene, der de optimale løsninger benyttes til å generere en ny populasjon. Dette vil fremgå i videre nodene, for et gitt antall generasjoner. Helt til slutt kan resultatene visualiseres.

Dette avsnittet er utarbeidet med validering og testing av Optimo (Rahmani et al., 2014), (Rahmani et al., 2015), (Rahmani, 2015) og (Varmeulen, 2017).



Figur 2.6: Et eksempel av oppbygging av Optimo i Dynamo

2.1.4 Python

Python er et *open source* programmeringsspråk, og er utbrett til bruk i ulike felt som matematikk, datavitenskap og webutvikling. Programmet er et tolket språk, noe som betyr at koden tolkes til andre språk via et "interpreterprogram", uten å kreve sammenstiling. Python-tilknyttingen som brukes i Dynamo fungerer via IronPython. IronPython er en implementering av Python laget av Microsoft i C#, og gjør det mulig for Python å kommunisere med .NET framwork (Talerico, 2017).

Python er nøkkelen til å kunne overvinne begrensninger der Dynamo ikke er tilstrekkelig. Via Python får brukeren mulighet til å få direkte tilgang til RSA via RSA API, noe som ikke kan nås med Dynamo. Da det ikke er utarbeidet noen dokumentasjon for RSA API til programmering i Phython, vil det være nødvendig å bruke Dynamo community og Autodesk support til dette.

2.1.5 Begrensninger

Implementering av grafer i Dynamo vil i utgangspunktet være begrenset til brukerens bakgrunn i programmering. Som beskrevet i Kappitel 2.1.2, er Dynamo et VP-verktøy som gjør det lettere for ikke-programmerere å kunne automatisere arbeidet via VP. I slike tilfeller vil arbeidet være begrenset til de ferdiglagde nodene og tilleggspakkene til Dynamo. Dette kommer spesielt til syne når en skal benytte VP til optimalisering med sofistikerte krav og betingelser, ved bruk av noder som for", "if"og "while loop". Disse begrensningene kan overkommes ved å innføre Python-skripter til grafen.

En annen utfordring er betydelig maskinvarekrav og minnebruk, ved håndtering av komplekse geometrier i Dynamo. I komplekse grafer som bearbeider flere hundre objekter og geometrier, vil oppbygning av geometrien være avgjørende for å minske tids- og minnebruken. I Dynamo er hendelsene ved å skape, kontra å visualisere en geometri helt forskjellige. Generelt er å skape en geometri mye raskere og bruker mindre minne enn å realisere objektet (Tierney, 2015). Geometrier som blir realisert kalles for "*tesselleted* geometri, og kan sammenlignes med å skrive en kakeoppskrift kontra å lage selve kaken, henholdsvis for untesseleted og tesseleted geometri. Der alle geometri-noder i Dynamo er tesseleted geometri, vil brukeren kun ha to muligheter for å øke hastigheten til grafen: Python-noder og ZeroTouch-noder. Dette er igjen en begrensning som avgjøres ut ifra utgangspunktet til brukeren.

Gjennomføring av optimalisering med tilleggspakken Optimo sammen med RSA er begrenset av maskinvarens tilgjengelige RAM. Det har blitt observert følgende korrelasjoner: 30 GB RAM for ca. gjennomkjøring av 1500 modeller. Den eneste løsningen for gjennomføring av flere iterasjoner er å lagre gjeldende befolkning, og bruke det som initialbefolkning etter gjenåpning av Dynamo og RSA.

2.2 Optimaliseringstyper og metoder

Enkelt beskrevet består et optimaliseringsproblem av å maksimere eller minimere en funksjon, som velger initialverdiene fra et gitt område og beregner utkomstverdien av funksjonen. Ut fra dette kan det konkluderes med at optimalisering handler om å finne den beste løsningen innenfor et sett med alternativer. Optimalisering av en hendelse kan være med hensyn på ett eller flere mål, der disse blir henholdsvis kalt Single Objective Optimization (SOO) og Multi Objective Optimization (MOO).

SOO er prosessen med å finne den beste løsningen av et problem, ved å enten ignorere andre faktorer av et problem, eller ved å kombinere disse til et mål. I byggeindustrien er som regel ikke et enkelt mål av interesse, og det er sjeldent mulig eller ønskelig om å finne en optimal løsning. Da optimalisering i praksis oftest vil innebære flere objektiver, vil optimalisering være å finne den optimale avstanden mellom disse målene, i stedet for å finne den beste løsningen (Vermeulen, 2018).

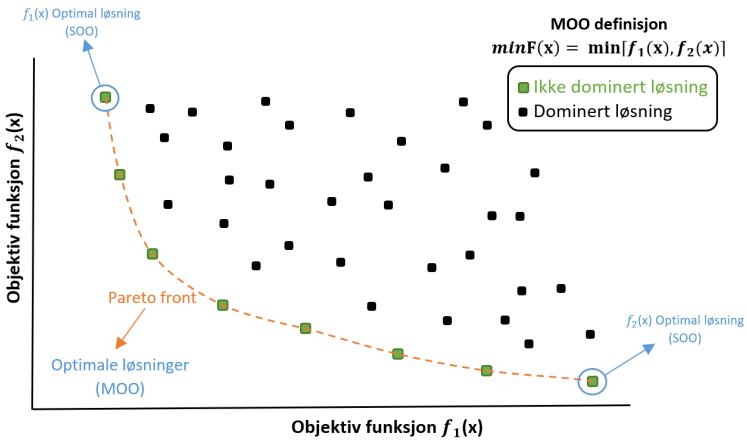
MOO er et område med flere kriterier for beslutningsprosesser, som er berørt av matematiske optimaliseringsproblemer, der det involveres mer enn én objektiv funksjon som skal optimaliseres. Å løse MOO-problemer betyr vanligvis å hjelpe en menneskelig beslutningsprosess å finne den mest optimale løsningen (Deb, 2014) og (Ruiz et al., 2008).

De oppnådde løsningene i et ikke-konvekst optimaliseringsproblem kan kun garanteres dersom hele domenet for definisjonen av objektive funksjoner utforskes. Dette vil i mange tilfeller være uhåndterlig, og forutsetter problemet til å benytte matematiske optimaliseringsmetoder som genetiske algoritmer til å oppnå løsningene (Giagkiozis and Fleming, 2015). Dette, sammen med beslutningsprosesser av MOO, blir forklart nærmere i dette kapittelet.

2.2.1 Pareto optimale løsninger

Pareto Optimal er en av de vanligste tilnærminger til en MOO-beslutningsprosess, utviklet av Vilfredo Pareto (Kenbur, 2005). Metoden fungerer slik at besvarelsene stilles opp etter en Pareto-front, som er en grense, opprettet i løsningsrommet til alle alternativene. Denne grensen representerer et sett av løsninger som ikke er dominert av andre løsninger. En slik ikke-dominert løsning refereres til de løsningene der det ikke finnes andre alternativer som kan forbedres uten å svekke de andre målene (Deb, 2002).

Pareto Optimale løsninger sammen med Pareto-fronten av et MOO-eksempel med to objektive funksjoner er illustrert i Figur 2.7. Da MOO overstiger to objektive funksjoner, vil det være utforende og visualisere Pareto-fronten nøyaktig.

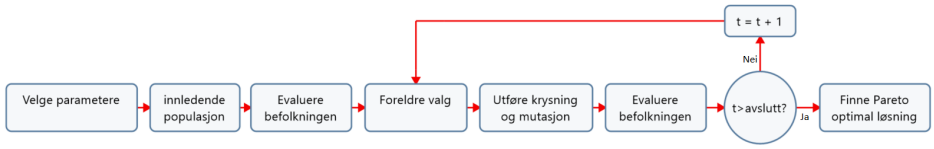


Figur 2.7: Pareto optimal løsning av en MOO med to objektive funksjoner som har mål om å minimeres

2.2.2 Genetisk optimalisering

Genetisk optimalisering er en optimaliseringsteknikk som er inspirert av Darwins evolusjonsteori, og benytter seg av genetiske algoritmer (GA). GA er en søkebasert algoritme, basert på naturlig valg og overlevelse av de sterkeste. GA har vist seg å være en rask og enkel metode innenfor globale optimaliseringsalgoritmer, og benyttes til et bredt spekter av problemer, deriblant funksjonsoptimalisering. GA er en god måte å løse MOO problemer på, med tanke på at metoden effektivt kan lokalisere flere Pareto optimale løsninger i et stort løsningsrom. En kan tenke seg å gjette det riktige tallet mellom én og en million. Når problemene er omfattende, vil det ta for lang tid å sjekke hver eneste løsning. Pareto optimale løsninger gir oss mulighet til å vite hvor bra eller dårlige svarene er, og på denne måten benyttes de gode svarene til å komme nærmere det riktige tallet.

Den generelle prosessen for en genetisk optimaliserings arbeid med GA er illustrert i Figur 2.8. GA begynner i utgangspunktet med en initial løsningsliste, som refererer til den opprinnelige populasjonen. Størrelsen på populasjonen er avhengig av problemets natur og kompleksitet, som oftest er tilfeldig generert innenfor et spesifikt område. Fra hver populasjon blir enkelte befolkninger som refereres til foreldre valgt til å avle en ny generasjon. Foreldrene blir utpekt i populasjonen gjennom en beslutningsprosess som Pareto-fronten, hvor befolkningene blir målt via en eller flere fitnessfunksjoner. Som nevnt tidligere kommer GA fra naturlig utvalg, hvor den opprinnelige befolkningen blir evaluert og krysset med hverandre til å produsere avkom. Hver iterasjon kan sees som en generasjon, hvor avkommene arver informasjonen fra foreldre med mulighet for tilfeldig mutasjon. Algoritmen kan enten avsluttes etter at et bestemt antall iterasjoner er nådd, eller et spesifikt resultat er oppnådd (Chukwuchekwa, 2011) og (Eiben et al., 1994).

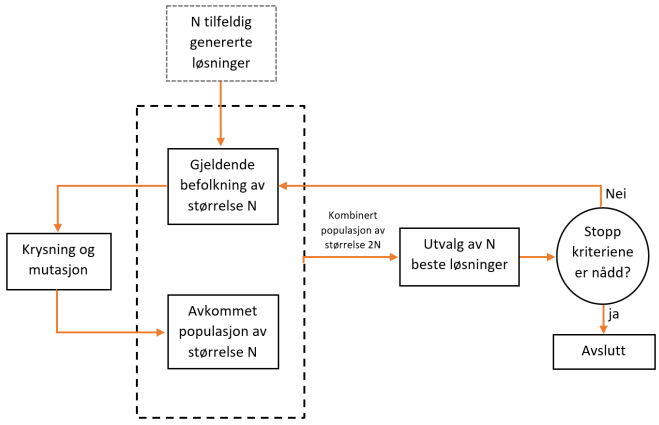


Figur 2.8: Generell arbeidsprosess av en genetisk optimaliseringsalgoritme

NSGA-II

Nondominate Sorting Genetic Algorithm II (NSGA-II) er en ikke-dominert sorteringsbasert elitisme og flerobjektiv evolusjonær algoritme (MOEA). Metoden er en av de mest populære MOO algoritmer, og leverer en rask ikke-dominert sorteringstilnærming, rask distansestimeringsprosedyre og en enkel sammenligningsoperatør av utvalgte løsninger (Yusoff et al., 2011). NSGA-II har forbedret kritiseringer som rammet de ikke-dominerte MOEA-metodene med høy beregningskompleksitet, mangel på elitisme og behov for å spesifisere delingsparameteren (Deb, 2002). Ved bruk av elitisme tillater ikke NSGA-II at en Pareto optimal løsning slettes, samtidig som en maksimal kompleksitet av O(MN2) opprettholdes, hvor M = antall objektive funksjoner, N = befolkningsstørrelse og O = øvre bundet (Eiben et al., 1994).

Generelt kan NSGA-II deles opp i seks steg, som er illustrert i Figur 2.9. Innledende populasjon er basert på problemområdet og begrensninger. Sorteringsprosessen baseres på Pareto-fronten med ikke-dominerende sortering av befolkningen. Når sorteringen er fullført, beregnes avstanden mellom de utvalgte i befolkninger. Utvelgelsen av foreldrene blir basert på rangen og avstanden mellom utvalgte. Etter at foreldrene er valgt, blir det gjennomført en genetisk operasjon. Foreldrene og avkommet blir kombinert sammen, slik at foreldrene til neste generasjon kan velges. Hver generasjon fyller opp eldre foreldrepopulasjonen, til befolkningsstørrelsen overstiger gjeldende størrelse (Yusoff et al., 2011).



Figur 2.9: Generell arbeidsprosess av NSGA-II

NSGA-II kan utrykkes som en genetisk evolusjonær algoritmisk form, som vist i Tabell 2.2. Fordelene ved NSGA-II-algoritmen er eksplisitt løsningsbevarelse, begrensede kompleksiteten og elitismen. Ulempene er muligheten for en overfylt sammenligning, som begrenser konvergensen og en ikke dominert sortering med størrelsen på 2N (Curry and Dagli, 2014).

$t \leftarrow 0$	Generasjon teller
C_0	Initial populasjon
while stopp kriterien(s) ikke sann **do**	Lag nytt generasjon
Evaluer fitness $f(x)$, av hvert individ $x \in C_t$	
Utfør reproduksjon for å skape avkom	
Velg den nye befolkningen	
$t \leftarrow t + 1$	Frem til ny generasjon

Tabell 2.2: Genetisk evolusjoner algoritme

2.2.3 Brute Force Search

"Brute force search", eller "uttømmende søk", er en enkel og generelt problemløsende algoritmeteknikk. Metoden består av systematisk beregning av alle mulige løsningsmuligheter, og kontrollering av kandidatene har tilfredsstiller problemets krav.

Brute force er enkelt å gjennomføre, og vil alltid kunne finne en løsning hvis den eksisterer. Bruken av metoden må veies for hvert enkelt prosjekt, der beregningskostnaden er proporsjonal med antall mulige tilfeller i problemet. Brute force brukes vanligvis der problemstørrelsen er begrenset, eller når en enkel implementering av problemet er viktigere enn hastighet (Trakhtenbrot, 1984).

2.3 Design prosess

Denne delen beskriver oppbygning og hensynet for tre foreslåtte designprosessmetoder benyttet til å utvikle denne oppgaven. Målet er en halvautomatisert optimaliseringsprosess i en tidlig prosjekteringsfase. Metodene skal kunne hjelpe brukeren til å utvikle et mer intelligent design ved hjelp av VP og konstruksjonsanalyse som supplerer designprosessen.

Metodene benytter tre forskjellige optimaliseringsteknikker, der grafene fungerer som en foreløpig analyse ved hjelp av et FEM-verktøy for beregning, halvautomatisk evaluering og rangering av de foreslåtte konstruksjonsdesignene. Den første metoden kalles Design utforskning", og fungerer som en manuell optimalisering. Arbeidsflyten som er opprettet med denne metoden kan brukes som en objektiv funksjon i Dynamo for de to andre metodene, henholdsvis Brute force search"og GA-optimalisering".

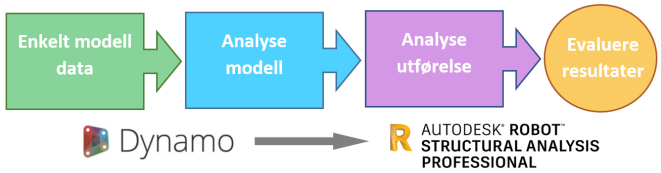
Gjennom oppgaven er de ulike trinne i hver metode forklart og illustrert, gjennom utvikling av parametriske modeller. Hensikten ved generering av Dynamo arbeidsflyt er å gjør

det enklere for brukeren å kunne utføre ulike typer optimalisering, basert på prosjektets behov og størrelse. De utviklede grafene er beregnet til bruk i en tidlig prosjekteringsfase, og benytter dermed Dynamo til å lage den parametriske geometrien. Metodene kan både benyttes til optimalisering av enkelte konstruksjonsdeler, som et dekke i en vilkårlig etasje, eller optimalisering av hele konstruksjonen.

Fokuset i oppgaven har vært å utvikle en arbeidsflyt i Dynamo, som benytter RSA som FEM-verktøy til optimalisering av konstruksjoner. Samspillet med RSA gjennom Dynamo er ennå ikke optimal som en følge av et begrenset antall tilgjengelige noder. I tilfeller med begrensninger for samspillet mellom programmene er tilkobling til RSA API blitt benyttet via Python programmering. Til evaluering av resultatene er det både benyttet Excel og tilleggspakken Mandrill i Dynamo.

2.3.1 Metode 1 - Design utforskning

Denne metoden er en manuell optimaliseringsprosess, der parameterne blir justert av bruker til optimaliseringen. En manuell optimalisering bør alltid bli gjennomført først, for å sikre at den parametriske modellen er suksessfull, før utvikling av metode 2 og metode 3. Arbeidsflyten av design utforskning er illustrert i Figur 2.10.



Figur 2.10: Arbeidsflyt av metode 1 - Design utforskning

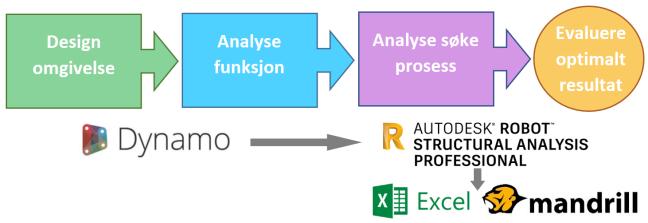
Enkeltmodell-data er oppbygning av den geometriske modellen i Dynamo. Dette kan som nevnt i Kapitel 2.1.2 gjennomføres på to måter, der denne oppgaven utvikler den parametriske modellen i Dynamo. I den geometriske modellen definerer brukeren prosjektets bestemte og varierende parametere. Det er viktig at prosessen ikke begrenser arkitektoniske friheten. Videre i "Analyse modellen' blir de valgte elementene definert i RSA, med gitte forutsetninger. Kalkulasjonen på analyse modellen gjennomføres og sjekkes, før nødvendige resultatet blir hentet tilbake til Dynamo. Innhentede resultater blir behandlet slikt som ønsket av brukeren. Prosessen i metode 1 - Design utforskning er vist i Tabell 2.3.

Enkeltmodell-data	
Step 01	Inngangsparametere • Bestemte parametere • Varierende parametere
Step 02	Oppbygning av konstruksjonens skjelett i Dynamo
Step 03	Visualisering av konstruksjonen
Analyse modell	
Step 01	Definere analytiske elementer og opplager betingelser • Bjelker, søyler og paneler • Opplager og forbindelse betingelser
Step 02	Bestemme material og tverrsnittet til elementene
Step 03	Anvende laster og lasttilfeller
Analyse utførelse	
Step 01	Gjennomføring av analyse i RSA
Step 02	Verifisere analysen
Step 03	Innhenting av resultater
Evaluere resultater	
	Resultatene blir behandlet slikt som ønsket av brukeren

Tabell 2.3: Oppbygning av metode 1 - Design utforskning

2.3.2 Metode 2 - Brute force search

Denne metoden baserer seg på den enkle algoritme metoden Brute force search. Metoden benytter modellen og analysen utført i metode 1 til å utføre og evaluere ulike verdier av inngangsparametere. Hver av de ulike kombinasjonene av varierende parametere skaper én nytt designalternativ. Brute force search kan enten utføres for alle designalternativene, eller kun for ett bestemt antall tilfeldig genererte alternativer. Metoden gir et oversiktlig bilde av problemet, og fungerer optimalt i SOO-problemer der parameterne er hele tall, som for eksempel valg av tverrsnittstype, størrelse og materialvalg for elementer i en konstruksjon. I større prosjekter med flere parametere, vil det i tillegg til en større kombinasjonsrom være utfordrende å evaluere MOO-problemer. Arbeidsflyten av Brute force search er illustrert i Figur 2.11.



Figur 2.11: Arbeidsflyt av metode 2 - Brute force search

Designomgivelsen definerer variablene og deres applikasjonsdomene i Dynamo. Det blir generert et bestemt antall designalternativer, basert på brukerens innledende parametere. Variablene i denne optimaliseringsprosessen er enten konstant eller drevet av brute force search. Antall designalternativer bestemmer hvor mange iterasjoner som blir gjennomført. For å kunne kjøre analysen for flere designalternativer, må det lages en analysefunksjon. Denne funksjonen lages ved å konvertere grafene utviklet i metode 1 til en custom node, slik at grafene kan brukes som en funksjon til de kombinerte designalternativene. I søkeprosessen blir analysen kjørt for det definerte omgivelsesdomenet. Når skriptet kjøres, vil hver iterasjon endre den geometriske modellen i Dynamo, og en ny RSA-fil opprettes i henhold til det nye designalternativet. Alle resultatene samles i utgangen av analysefunksjonen for videre evaluering. Etter at søkeprosessen er over og alle iterasjonene er gjennomført, kan resultatene eksporteres til Excel eller visualiseres med tilleggspakken Mandrill til evaluering av optimale løsninger. Prosessen i metode 2 - Brute force search er vist i Tabell 2.4.

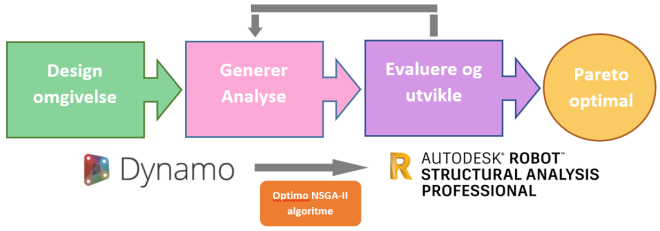
Designomgivelse	
Step 01	Inngangsparametere • Definere bestemte parametere • Definere varierende parametere
Step 02	Definere kriterier og betingelser
Analysefunksjon	
Step 01	Lage analysefunksjon basert på metode 1
Step 02	Bestemme inngang og utgangsdata
Analyse søkeprosess	
	Kjøre analysen for det definerte omgivelsesdomenet
Evaluere optimale løsninger	
	Resultater blir behandlet slikt som ønsket av brukeren • Eksport til Excel • Visualisere via Mandrill

Tabell 2.4: Oppbygning av enkelt modell data

2.3.3 Metode 3 Genetisk algoritmisk optimalisering

Genetisk algoritmisk optimalisering introduserer oss til GA-analyseprosess, og baserer seg på Darwins evolusjonsteori. GA-optimalisering i denne oppgaven benytter NSGA-II metoden, som gjør et naturlig utvalg av den opprinnelige løsningslisten, ved bruk av krysninger og mutasjonsalgoritmer for å finne den optimale løsningen. GA-optimaliseringsmetoder er nærmere beskrevet i Kapittel 2.2. Denne metoden bruker på samme måte som Brute force search, modellen og analysen utført i metode 1 til å lage fitnessfunksjoner. Disse fitnessfunksjonene legges til spesifikke noder fra Optimo, til å gjennomføre GA-optimalisering. Tilleggspakken Optimo er nærmere beskrevet i Kapittel 2.1.3. I motsetning til Brute force search metoden som vurderer alle tilfellene, er GA-optimalisering en raskere prosess. Dette ved å eliminere dominerte løsninger i hver generasjon. GA-optimalisering kan be-

nyttes i større MOO-problemer som for eksempel *form finding* problemer. Arbeidsflyten av GA-optimalisering er illustrert i Figur 2.12.



Figur 2.12: Arbeidsflyt av metode 3 - Genetisk algoritmisk optimalisering

Lik som Brute force metoden, blir bestemte og varierende parametere definert i designomgivelsen. Designvariablene skaper et variabelt domene, der brukeren definerer øvre og nedre grenser som søkes en optimal løsning for. For å kunne generere analysen må objektive fitnessfunksjoner lages til å kunne lese av populasjonens variabellister. Analysefunksjonen, som beskrevet i metode 2, fungerer som beregningsfunksjonen i objektive fitnessfunksjoner. Antallet fitnessfunksjoner representerer antallet objektive mål som skal maksimeres eller minimeres. Videre blir en liste over objektive fitnessfunksjoner koblet til Optimo-nodene for å kunne evaluere og utvikle generasjoner. Før grafene kan kjøres, bestemmer brukeren antall iterasjoner, ved å bestemme populasjonsstørrelse og antall generasjoner. Resultatene av GA-optimalisering er Pareto optimale løsninger, og kan bli behandlet ved å visualisere optimale løsningene i en parallellkoordinat-graf. Prosessen i metode 3 - Genetisk algoritmisk optimalisering er vist i Tabell 2.5.

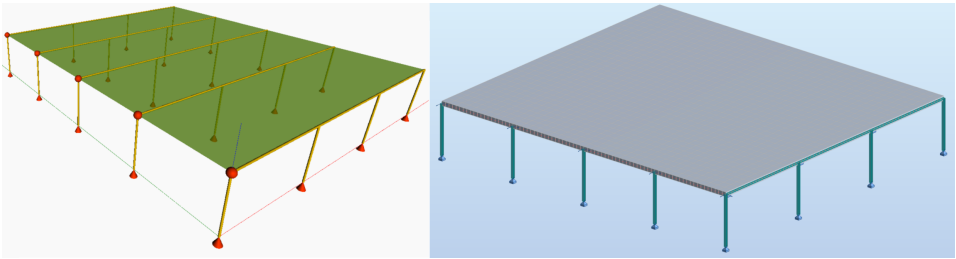
Designomgivelse	
Step 01	Inngangsparametere • Definere bestemte parametere • Definere varierende parametere
Step 02	Definere kriterier og betingelser
Generer analyse	
Step 01	Lage objektive fitnessfunksjoner basert på metode 1 • Bestemme antall objektive mål • Bestemme inngang og utgangsdata
Step 02	Bestemme øvre og nedre grenser til initial populasjonen
Evaluere og utvikle	
	Tilkobling til Optimo-noder • Bestem populasjonsstørrelse og antall generasjoner
Pareto optimal	
	Visualisering av resultatene i en parallellkoordinat-graf • Visualisere i Mandrill

Tabell 2.5: Oppbygning av genetisk algoritmisk optimalisering

Kapittel 3

Optimalisering av dekkesystemer

Dette kapitlet handler om utvikling av en parametrisk modell, som benyttes til optimalisering av et dekkesystem for et gitt rektangulært areal. En vilkårlig løsning er illustrert i Figur 3.1, med Dynamo-modell til venstre og RSA-analysemodell til høyre. Bakgrunnen for utvikling av parametriske modellene er å kunne automatisere en prosess, som kan identifisere ett eller flere forslag til optimal spennvidde for et dekke i et tidlig forprosjektstadie. Bakgrunnen for å lage en slik automatiseringsprosess er utarbeidet i sammarbeid med Sweco, med hensikt om å kunne utforske flere optimale dekkesystemalternativer, med begrenset ressursbruk. Videre gjennom dette kapitlet vil oppbygning av en parametrisk problemstilling bli forklart ved å gjennomføre tre forskjellige optimaliseringsmetoder som er forklart i Kapittel 2.3.



Figur 3.1: En virkårlig løsning av dekkesystemet med geometrisk modell i Dynamo til venstre og analysemodell i RSA til høyre

Optimalisering av dekkesystemet arbeider mot en løsning som har den laveste konstruksjonsvekten, med minimal bruk av lokale armeringsforsterkninger. Dynamo-grafene gir brukeren mulighet til å definere et rektangulært areal for dekket, der forskjellige bæresystemer skal undersøkes. Optimaliseringsprosessen undersøker plasseringer og antall bærende elementer for å opprettholde dekket, samt valg av tverrsnittstype, størrelse og material

for disse elementene fra en gitt liste.

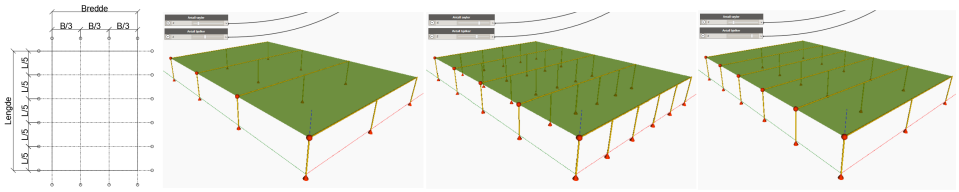
I dette kapitlet er det valgt å undersøke et toveisplate-betongdekke med fasthetsklasse B35, og tversnittstykkelse på 230 mm. Det valgte dekket skal optimaliseres for et grunnareal på 450 m2, med bredde "X-retning"og lengde "Y-retning", henholdsvis 18 x 25 m. Valget av toveisplater er et mer praktfullt eksempel, der enveisplater er som oftest prefabrikkerte elementer med faste mål fra produsenten. Den parametriske modellen er samtidig bygget opp slikt at det gir brukeren mulighet til å benytte enveisplater i optimaliseringsprosessen.

Lastforutsetningen i modellen er tiltenkt for et kontorbygg med kun egenlast, nyttelast og permanentlast. Det er imidlertid sett bort i fra sideveis avstivning, og dermed er vindlast ikke definert. Modellen lar brukeren enkelt endre eller legge til flere lasttilfeller. Følgende karakteristiske lastverdier etter NS-EN 1991-1-1:2002/NA:2008 er lagt til grunn, tabell 3.1.

Nyttelast Tab NA 6.4/6.10	Last q_K $[kN/m^2]$
Kategori C1, Undervisningsrom/kontor	3,0
Permanente laster	Last q_K $[kN/m^2]$
Etasjeskille eksl.bærende bygningsdel	1,2

Tabell 3.1: Lasttilfeller av etasjeskille i et kontorbygg

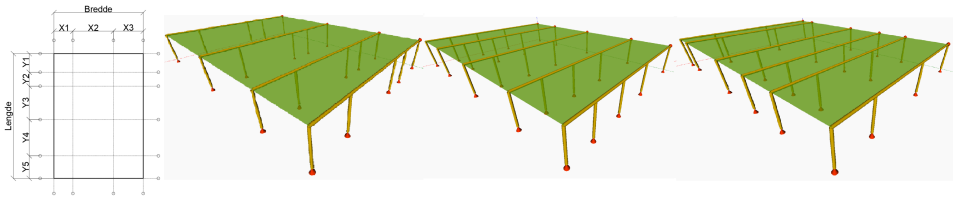
De varierende parameterne av geometrien er antall bjelker og søyler i underkant av dekket, der plassering av disse elementene vil utvikles i to forskjellige utforminger videre i oppgaven. Den første utformingen baserer seg på en praktisk løsning, ved at konstruksjonen skal for alle tilfeller ha en symmetrisk utforming. Avstanden mellom alle bjelkene og søylene er like. Figur 3.2 illustrer den symmetriske dekkegeometrien. Denne metoden å definere geometrien på vil redusere antall mulige utfall av alternativene. Et eksempel kan refereres til når bjelkene i underkant av en toveisplate blir plassert slikt at feltmomentet i alle dekkene blir like. Den symmetriske utformingen av modellen er benyttet i metodene Design utforskning og Brute force search.



Figur 3.2: Illustrasjon av det symmetriske dekkesystemet med plan-aksekrysning og tre tilfeldige genererte løsninger

Den andre utformingen av konstruksjonen er en mer kompleks løsning, ved at hvert element kan bevege seg fritt i elementets frie avstand. Figur 3.3 illustrer den usymmetriske dekkegeometrien. Denne parametriske modellen vil i motsetning til den symmetriske utformingen kunne skape uendelig alternativer. Dette inviterer oss til å bruke en mer sofisti-

kert metode som GA til utforskning av løsningene. Den usymmetriske geometridefineringen benyttes i metode 3, GA-optimalisering med Optimo.



Figur 3.3: Illustrasjon av usymmetrisk dekkesystem med plan-aksekrysning og tre tilfeldig genererte løsninger

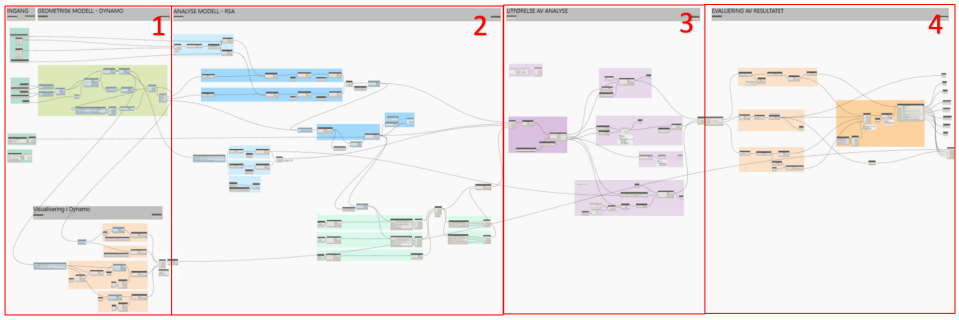
Rangering av alternativene foregår gjennom et Python-kode, og tar hensyn til både spenningstilstand og nedbøyning i konstruksjonen. Alle de utviklede Python-kodene som beskrives gjennom dette kapitlet er vedlagt sammen med de utviklede Dynamo-modellene i et elektronisk vedlegg. Oversikt over det elektroniske vedlegget er vist i en tabell i Vedlegg A.

3.1 Metode 1 - Design utforskning

Dette delkapittelet handler om oppbygning av den parametriske modellen av det usymmetriske dekkesystemet. Hensikten er å fremheve mulighetene for å automatisere en prosess ved å vise fremgangsmåten og utfordringene av dekket under utvikling av denne parametriske modellen.

Design utforskningsmetoden er en manuell optimaliseringsprosess, ved at brukeren selv bestemmer parameterne av alternativene. Denne metoden danner grunnlaget for gjennomføring av en algoritmisk optimaliseringsprosess som Brute force search eller GA-optimalisering. Modellen som er utviklet gjennom Design utforskning, fungerer som en fitnessfunksjon i de algoritmiske metodene. Design utforskning bør alltid gjennomføres først. Da modellen fungerer i en manuell optimaliseringsprosess, vil den også fungere for algoritmiske optimaliseringsprosesser.

I denne oppgaven er hele den parametriske modellen utviklet i Dynamo, der utvalgte deler av konstruksjonen blir definert i RSA, via Structural Analysis pakken og RSA API. Videre utføres kalkulering av analysen i RSA for det definerte alternativet, for deretter å tilbakehente ønskede resultater til Dynamo for videre evaluering og rangering. Figur 3.4 illustrerer en oversikt over hele grafen i Dynamo, utviklet gjennom metode 1 – Design utforskning, sammen med en punktvis forklaring i Tabell 3.2.



Figur 3.4: Oversikt over grafen for parametriske modellen utviklet gjennom metode 1 - Design utforskning

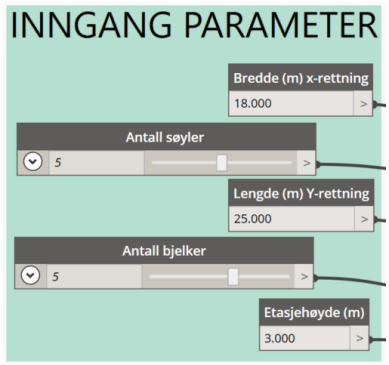
1 - Geometrisk modell
- Inngangsparametere
- Geometri skjelettet
- Visualisering av geometrien
2 - Analyse modell
- Definere analytiske elementer og opplagerbetingelser
- Definere materialet og tverrsnittet til elementene
- Anvende laster og lastkombinasjoner
3 - Utføre analyse
- Gjennomføre analyse i RSA
- Verifisere analysen
- Innhente utvalgte resultater fra RSA til Dynamo
4 - Evaluering av analysen
- Rangere resultatene via Python kode
- Visualisere resultatene i Dynamo

Tabell 3.2: Stykkevis forklaring av parametriske modellen utviklet i metode 1 - Design utforskning

3.1.1 Geometrisk modell

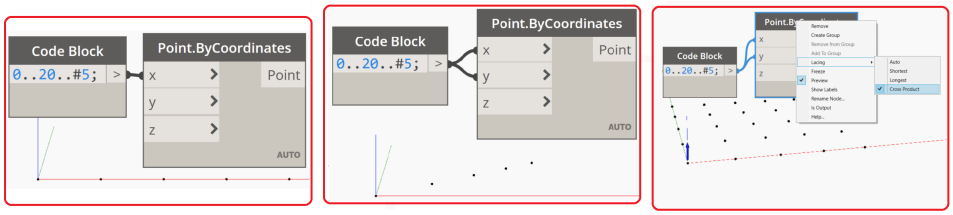
Den geometriske modellen er skjelettet til konstruksjonen, og definerer strukturen til den parametriske modellen. Oppbygning av en parametriskmodell bør alltid vurdere konstruksjonsmessige effekten av en parameter som benyttes i en automatiseringsprosess.

Faste parametere i denne oppgaven definerer forutsetninger av en analyse som optimaliseringen søkes for. De geometriske fasteparameterne er dimensjonen av flaten og etasjehøyde. Etasjehøyden definerer søylenes lengde. De varierende parameterne definerer endringer som skaper forskjellige alternativer av dekkesystemet. De geometriske varierende parametere er antall bjelker og antall underliggende søyler. Antall bjelker sammen med underliggende søyler definerer dekkets spenvidde. Figuren 3.5 illustrerer geometriske parametere som er benyttet i denne oppgaven.



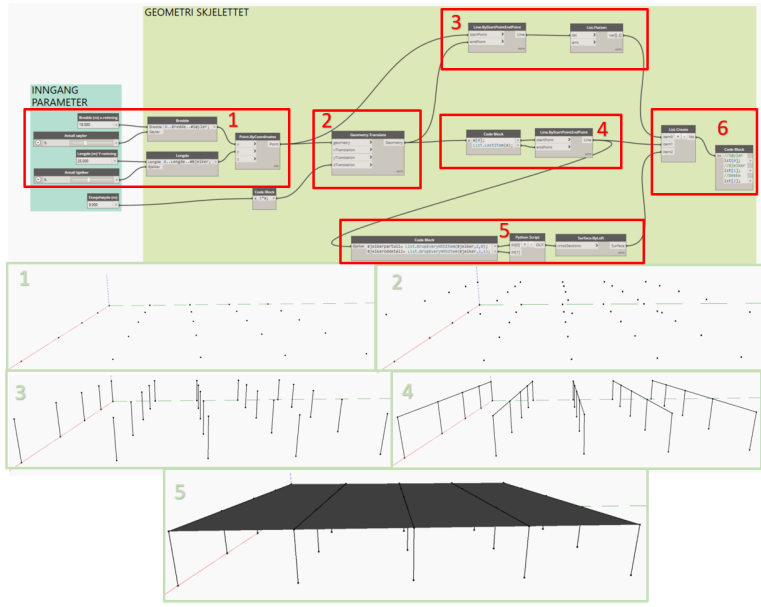
Figur 3.5: Geometriske faste og varierendeparametere for metode 1 - Design utforskning

Det geometriske skjelettet i denne metoden er definert slik at planets aksekrysningspunkter får en symmetrisk plassering av bjelker og søyler. En god metode for å starte oppbygningen av en geometrisk modell er å definere punkter som kan representere fotavtrykket til et av alternativene. Denne geometriske modellen startet med å finne søylenes plassering ved å definere en viss mengde punkter i X- og Y-retning. Disse punktene representerer henholdsvis antall søyler og bjelker. Listene med X- og Y-verdier kobles til noden *Point.ByCoordinates*, med kryssproduktkombinasjon av listene. Figur 3.6 illustrerer oppbygningsprinsipp av fotavtrykket, og det parametriske systemet.



Figur 3.6: Oppbyggingsprinsippet av fotavtrykket som fremkommer i metode 1 - Design utforskning

Etter å ha skapt fotavtrykket til et av alternativene, er samme prinsipp benyttet til å bygge opp et parametrisk fotavtrykk av søylene. Denne gang er antall punkter i X- og Y-retning valgt å være variabler. Videre er listen med fotavtrykkspunkter kopiert til en høyde med Z-koordinat lik etasjehøyden. Listen av punktene i bunnen og punktene i etasjehøyde benyttes til å definere linjer som representerer søylene. Bjelkene representeres på samme måte, ved å definere linjer med to referansepunkter fra listen av punkter i overkant. Bjelkelinjene er benyttet til å definere flater som representerer dekket. Denne listen er sortert via en Python-kode, slik at det blir laget et flateobjekt mellom alle bjelkelinjer. Dette gir brukeren mulighet til å kunne definere momentfrie kantlinjer av dekket, og dermed kunne optimalisere dekker som ikke er kontinuerlig over bjelkene. Til slutt samles og sorteres alle elementene i lister. En oversikt av det geometriske skjelettet og visualisering av prosessene i grafen er illustrert i Figur 3.7, sammen med en stykkevis forklaring i Tabell 3.3.



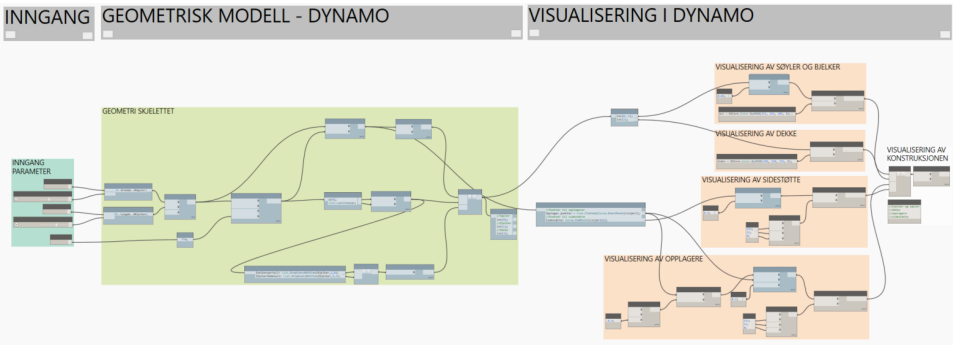
Figur 3.7: Oversikt av det geometriske skjelettet og visualisering av prosessene i metode 1 - Design utforskning

1 - Definering av søylenes fotavtrykkpunkter
2 - Kopiering av fotavtrykket i høyde med etasjehøyden
3 - Definering av søylelinjer
4 - Definering av bjelkelinjer
5 - Sortering av bjelkelisten i Python og definering av flater
6 - Samling og sortering av alle objektene

Tabell 3.3: Stykkevis forklaring av prosessene i det geometriske skjelettet utviklet i metode 1 - Design utforskning

Visualisering av konstruksjonen er benyttet til å organisere og fargelegge forskjellige konstruksjonsdeler. Som forklart i Kapittel 2.1.5, er det betydelig raskere å skape kontra å visualisere et objekt. Dynamo gir muligheten til å skjule "*Hide*slikt at det ikke trenger å visualiseres. Denne funksjonen brukes til å skjule objektene som bearbeides under utvikling av det geometriske skjelettet, og kun visualisere de valgte objektene helt til slutt.

Når en omformer en graf til en "*custom node*", er det kun objektene i utgangen av den egendefinerte noden som vil visualiseres. Dette gjør at visualiseringen blir nyttig i videre-kommende metoder, da grafen utviklet i metode 1 benyttes som en fitnessfunksjon ved at den omformes til en custom node. Dette er grunnen til at visualiserte elementer samles til en liste, slik at det senere kan sendes til utgangen av fitnessfunksjonen. En oversikt over grafen av den geometriske modellen er illustrert i Figur 3.8. Nodene som har gråere toner indikerer at objektene i noden er skjult.



Figur 3.8: Oversikt over oppbygning av geometrisk modell i metode 1 - Design utforskning

3.1.2 Analysemodell

Analysemodell-prosessen definerer den fysiske geometrien og nødvendige forutsetninger som legger grunnlaget for gjennomføring av konstruksjonsanalysen i RSA. Oppbygningen av analysemodellen benytter geometriskjelettet til å opprette konstruksjonensutforming i RSA, forså å definere elementenes fysiske form og samvirke. Etter påføring av lastene til de utvalgte elementene, samles alle objektene i en liste for kalkulering av analysen. Gjennom dette delkapittelet er valgte forbindelser, opplagerbetingelser og lastkombinasjoner for dekkesystemet vist, slikt at brukeren er kjent med kvaliteten og presisjonen til automatiseringsprosessen. Hensikten for denne delen er å fremheve mulighetene og utfordringene en kan møte under utvikling av en automatiseringsprosess for dimensjonering av konstruksjoner.

Autodesk har utgitt Structural Analysis-pakken, som inneholder en rekke med noder for å skape en kobling mellom Dynamo og RSA. Disse nodene kan håndtere de fleste oppgavene og objektdannelser i RSA, og er godt egnet for konstruksjoner uten plate-elementer. Hvis mer avanserte oppgaver er påkrevet, oppstår behovet for å skrive direkte til RSA API.

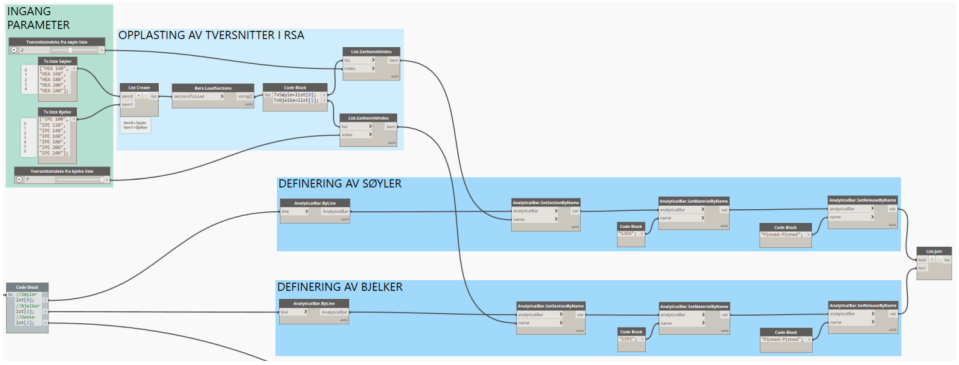
Prosjektets omfang i denne oppgaven er rettet mot en tidlig prosjekteringsfase, og dermed inngår de fleste nodene for oppretting av analysemodellen. RSA API er hovedsakelig benyttet for definering av prosesser for plate-elementer, som definering av dekketverrsnitt, endefrigjøring av dekke, flate last påføring og tilbakehenting av analyseresultater for panelene. Bruk av RSA API med Python i Dynamo er vist i Kapittel 3.1.5.

Ved optimalisering av dekkesystemet for denne casen er det valgt å undersøke optimal spennvidde for et gitt dekke i en vilkårlig etasje i et kontorbygg. De faste parameterne i analysemodellen er dekketykkelsen, samt konstruksjonens forutsetninger som last størrelse og elementforbindelse. Modellen er bygget opp slikt at brukeren har mulighet til å definere dekketykkelsen som en variabel optimaliseringsparameter i undersøkelsen. De varierende parametere i dette tilfellet er søylene og bjelkenes tverrsnitt. Tverrsnittet til disse elementene undersøkes for valgte standardprofiler fra NS 3472 av stålkvalitet S355. Den manuelle optimaliseringsprosessen benyttes til å indikere øvre og nedre størrelse av tverrsnittsprofiler som skal benyttes i de algoritmiske metodene.

Definering av søyler og bjelker

Opprinnelig vil definering av søyler og bjelker i RSA skje på sammen måte, ved at disse elementene defineres som stavelementer med to noder. Geometriinformasjonen fra bjelke- og søylelinjer i Dynamo, som også er definert med to punkter, benyttes til å bygge alle stavelementene i RSA. Deretter defineres elementenes tverrsnitt, materialkvalitet og endeforbindelser. De valgte definisjonene som benyttes må eksistere i det åpne RSA-dokumentet, eller defineres før grafen kjøres. Ettersom Dynamo-grafen er ment for automatiseringsprosessen, er ikke dette et alternativ. Derfor er listene med tverrsnittsprofiler lastet opp i det åpne RSA-dokumentet via Dynamo.

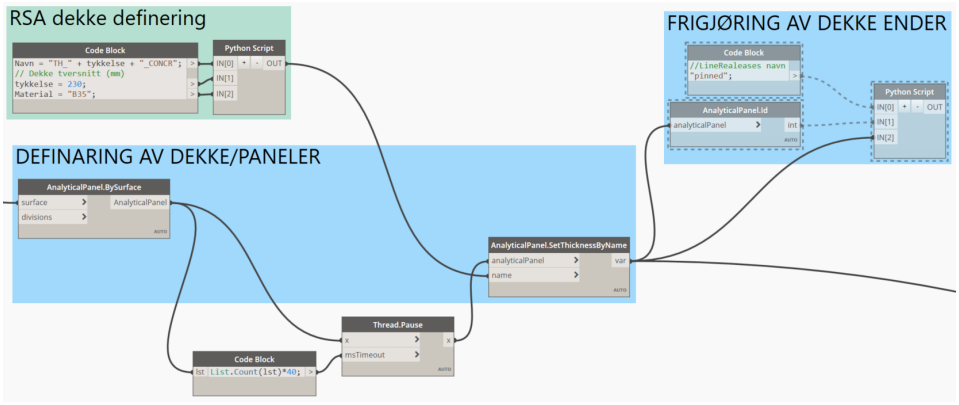
For at stavelementene ikke skal motvirke nedbøyningen av dekket, er det valgt å benytte leddlagrede søyleforbindelser. Bjelkene er definert kontinuerlig i overkant av søylene. Definering av søyler og bjelker er illustrert i Figur 3.9.



Figur 3.9: Oversikt over definering av bjelker og søyler i RSA via Dynamo

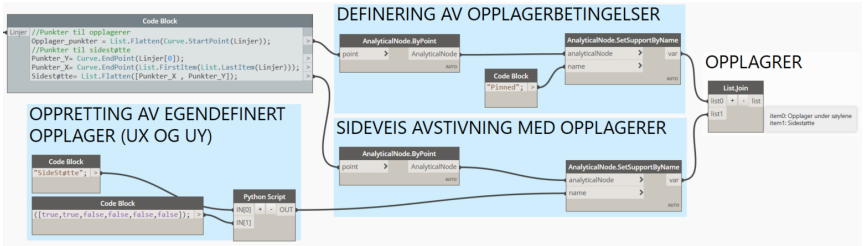
Definering av dekke
Definering av dekke i RSA som analytiske paneler, utvikles ved bruk av Structural Analysis pakken, ved å benytte koordinatene til flatene opprettet gjennom den geometriske modellen. Frem til disse panelene blir tildelt en definisjon på tykkelsen, fungerer de som "«Caddings»"i RSA. Etter at panelenestverrsnittet er definert, beregnes disse objektene som FE-Elementer. Kalkulasjonsmetoden og FE-typen blir satt til "*Defaulte*"i det åpne RSA-dokumentet. Kalkulasjonsmetodene utdypes i kapittel 3.1.3 Utføre analyse.

På samme måte som definering av stavelementene, er det kun de definerte dekketykkelsene i det åpne-RSA-dokumentet som kan benyttes. Dynamo-nodene har i dag ingen funksjon for å opprette nye dekketykkelse i RSA. Derfor er RSA API benyttet til å utvikle en Python-kode som definerer ønskede dekketykkelser. På denne måte har brukeren mulighet til å benytte dekketykkelsen som en variabel i optimaliseringsprosessen. RSA API er i tillegg benyttet til å utvikle Python-koden, som gir brukeren mulighet til å momentfrigjøre paneletes kanter. Denne funksjonen er fryst og ikke benyttet i denne oppgaven, da analysen er tiltenkt for en toveisplate. Definering av panelene og dekketykkelsen er illustrert i Figur 3.10.



Figur 3.10: Oversikt over definering av dekke i RSA via Dynamo

Definering av opplagerbetingelser
Definering av opplagerbetingelser benytter bunnpunktene fra søylene, og definerer disse som analytiske noder i RSA. Deretter er fasteopplagere definert i disse nodene. Valget mellom innspente og faste opplagerbetingelser vil for denne tilfellen ikke ha noen påvirkning på systemet, ved at søylenes endeforbindelser er valgt til å være leddet. For å stabilisere bæresystemet, er en egendefinert opplager som kun er fastholdt i det horisontale planet, benyttet i to sider av dekkekanten. Denne løsningen er valgt fremfor avstivningssystemer, for å ikke forsterke søylene og motvirkning av nedbøyningen i dekket. Den egendefinerte opplagerbetingelsen er opprettet med RSA API via en Python-kode, også kalt for "*Side-Støtte*". Definering av opplagerbetingelsene er illustrert i Figur 3.11.



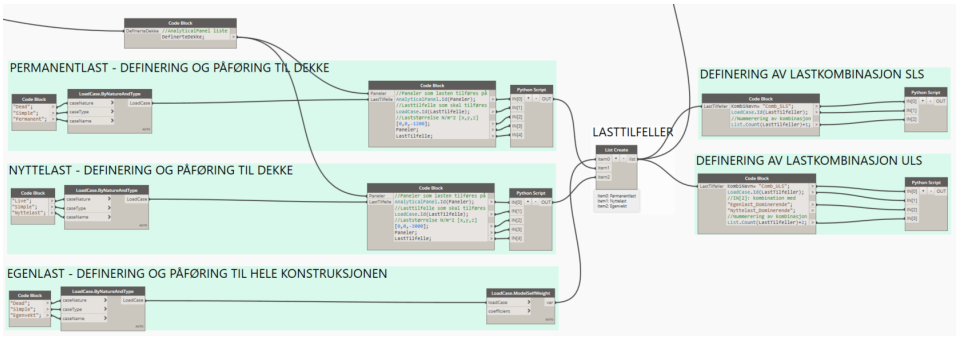
Figur 3.11: Oversikt over definering av opplagerbetingelser i RSA via Dynamo

Definering av lasttilfeller og lastkombinasjoner

Lasttilfellene egenlast, nyttelast og permanent last defineres ved bruk av Dynamo-nodene i det åpne RSA-dokumentet. Da Structural Analysis pakken kun gir muligheten til å påføre node- og linelaster, er det utarbeidet en Python-kode som gir brukeren muligheten til å definere flatelaster. De karakteristiske lastene påføres til de analytiske panelene i RSA, ved bruk av RSA API. Lasttilfellene samles fra utgangen av Python-kodene til en liste, for så å definere lastkombinasjoner.

For å kunne definere lastkombinasjonene, er RSA API benyttet til å utvikle en Python-kode som gir brukeren muligheten til å definere enkle lastkombinasjoner. Det er ikke mulig å definere lastkombinasjoner via Dynamo-nodene.

Lastkombinasjonene er begrenset til to tilfeller, og forenklet slik at nyttelasten fungerer over hele dekket. Det er sett bort ifra feltvis påvirkning av nyttelasten. Lastkombinasjonen i bruddgrensetilstand (ULS) benyttes videre for å kontrollere spenningstilstanden til elementene, og bruksgrensetilstand (SLS) til nedbøyningskontroll av dekkene. Definering og påføring av lasttilfeller og lastkombinasjoner er illustrert i Figur 3.12.



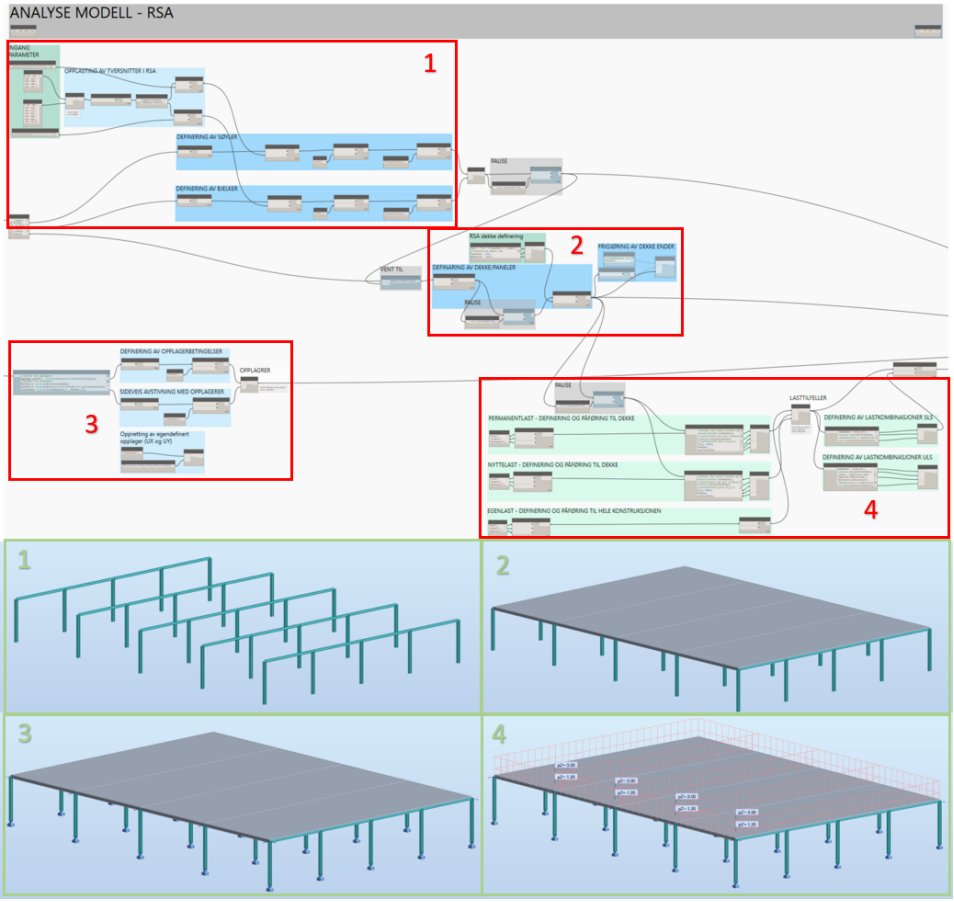
Figur 3.12: Oversikt over definering av lasttilfeller og lastkombinasjoner i RSA via Dynamo

Det er viktig å være oppmerksom på at Dynamo og RSA utfører disse prosessene med forskjellige hastigheter. Det anbefales derfor å benytte noden "*Thread.Pause*for å bremse Dynamo-prosessene, slikt at RSA får tid til å gjennomføre prosessene før det belastes med nye. Dette er spesielt viktig i skaping av større modeller, og når en benytter både Structural Analysis-pakken sammen med RSA API i samme graf. Bremsingsmengden av

prosessene er avhengig av ytelsene til brukernes verktøy, der det i denne oppgaven ble brukt 20 millisekunder per stavelement, og 40 millisekunder per panel. RSA inkluderer PARDISO-løsningen fra Intel R Math Kernel Library (Intel R MKL) som den mest fordelaktige løsningen for beregning av store strukturmodeller, ved å benytte flere kjerner av prosessoren (AUTODESK, 2017a).

For at alle prosessene i grafen skal kjøres i riktig rekkefølge er det benyttet en *Passer"* og *"VentTil"* -funksjon. Denne funksjonen benyttes for å sikre at RSA ikke blir overbelastet ved at analytiske prosesser gjennomføres steg for steg. Pause- og ventefunksjonen er fargelagt med grått i grafen.

En oversikt over oppbygningen av den analytiske modellen og visualisering av prosessene i RSA er illustrert i Figur 3.13, sammen med et stykkevis forklaring i tabell 3.4.



Figur 3.13: Oversikt over analysemodellen og visualisering av prosessene i metode 1 - Design utforskning

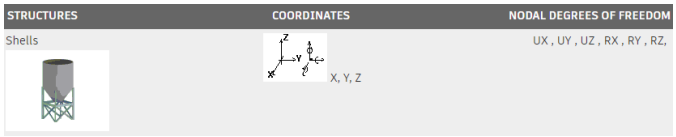
1 - Definering av søyler, bjelker og samvirke mellom disse
2 - Kopiering av dekke og dekketykkelse
3 - Definering av opplagerbetingelser
4 - Definering av lasttilfelle og lastkombinasjoner

Tabell 3.4: Stykkevis forklaring av prosessene i analysemodellen utviklet i metode 1 - Design utforskning

3.1.3 Utføre analyse

Gjennom utførelse av analysen er den definerte modellen i RSA kalkulert, og nødvendige resultater hentet tilbake til Dynamo. Før resultatene bekreftes for videre evalueringsprosess, er til slutt modellen og beregningene inspisert og validert i RSA. Dette delkapitlet viser analysemetoder som er benyttet for beregninger av denne oppgaven, og mulighetene for eventuelle endringer.

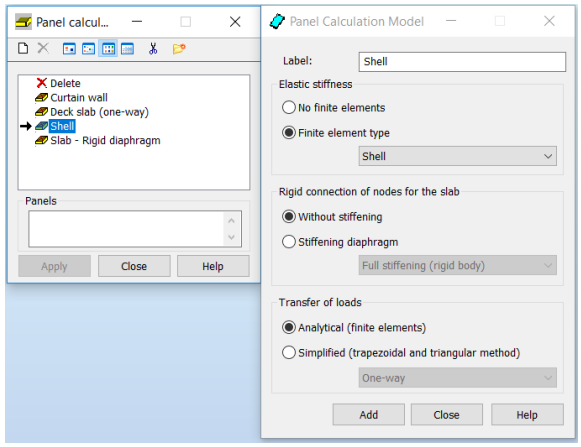
Analyse og beregnings type
Da denne automatiseringsprosessen er ment til forprosjektstadige av et prosjekt, er det kun gjennomført en lineær statisk analyse av denne modellen. Beregningsmodellen som benyttes ved å starte kalkulasjonen via Dynamo-nodene, settes til egendefinert eller *Default*-innstillinger av det åpne RSA-dokumentet. Denne oppgaven benytter opprinnelige innstillinger av konstruksjonstype "*Shell Design*", med seks frihetsgrader i hver node, Se figur 3.14.



Figur 3.14: Konstruksjonstypen *Shell Design* og beregningsforutsetninger

Den opprinnelige innstillingen benytter en kalkuleringsmodell for FE, kalt "*Shell*". Parameterne av Shell-metoden benytter skall-FE, uten "*Rigid*tilkobling av noder i panelene. Dette sørger for å gi et realistisk bilde på panelenes nedbøyningsform ved at koblingene i nodene ikke oppfører seg som et stivt legeme. Parameterne av kalkuleringsmodellen *Shell* er vist i figur 3.15.

Overføring av belastningene er avhengig er FE-typen og størrelsen, der RSA benytter de såkalte Discrete Kirchoff-Mindlin Triangle (DKMT) og Discrete Kirchoff-Quadrilateral (DKMQ), henholdsvis for tre og fire noders elementer i bøyningstilstand av paneler (AUTODESK, 2017b). Denne oppgaven benytter elementtypen av DKMQ med fire noder og 12 frihetsgrader. Dette elementet er validert for både bruk av tykke og tynne plater, ved at elementet er fri for "*shear locking*"og passerer "*the patch test*for en vilkårlig mesh-geometri (Katili, 1993).



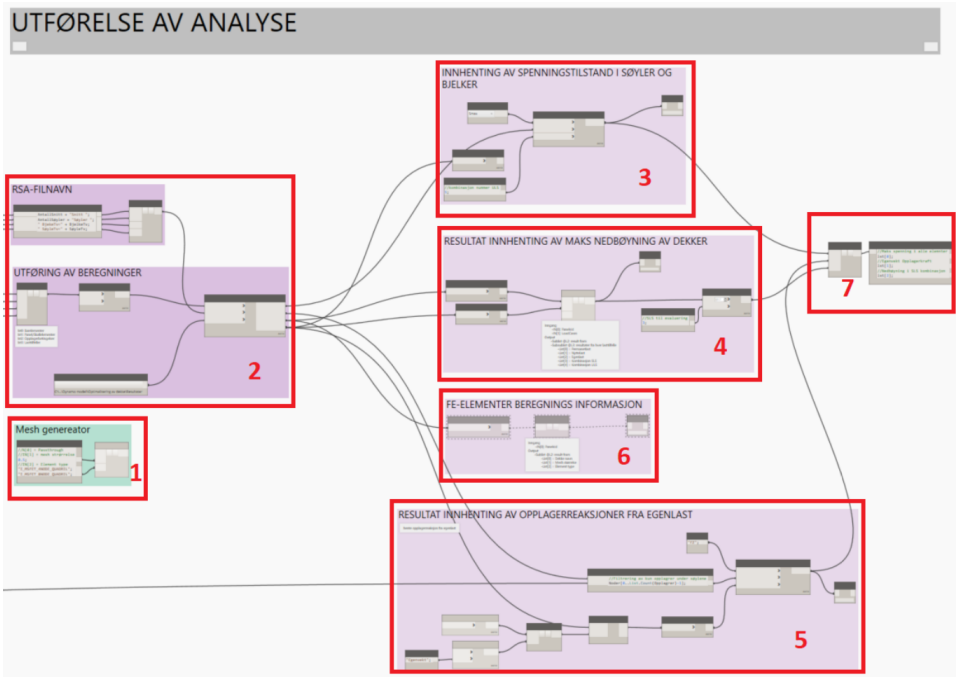
Figur 3.15: Opprinnelig innstilling av plateberegning i RSA

Alle stavelementene benytter en standard Matrix Structural Analysis (MSA), med to-noders elementer (Przemieniecki, 1985).

Oppbygging
Kalkulasjonen av modellen gjennomføres ved å starte kalkulasjonsmotoren i RSA. Utføring av beregningen benytter Dynamo-noden "*Analysis.CalculateWithSave*", som gjennomfører en linjer analyse i RSA. Denne noden lagrer alle gjennomførte analysene i en spesifisert mappe. Dette gir brukeren mulighet til å kunne gjennomføre en mer detaljert analyse for ønskede alternativer.

For at kalkulasjonen skal gjennomføres for hele analysemodellen, må følgende informasjon sendes inn i kalkuleringsnoden: analytiske stavelementer (og skallelementer), opplagerbetingelser, noder, lasttilfelle(r) og påførte laster. Denne funksjonen sikrer at alle prosessene er definert i det åpne RSA-dokumentet, før kalkulasjonen startes. Da en benytter RSA API til å gjennomføre en prosess, må brukeren selv sørge for at prosessene er utført, ved å eventuelt bremse Dynamo. Dette vil utdypes nærmere sammen med Python-kodene i Kapittel 3.1.5.

Tilbakehenting av resultatene fra stavelementer og opplagere gjennomføres ved bruk av Dynamo-noder. For FE-panelene benyttes RSA API, da dette ikke er mulig med Dynamo-nodene. For evaluering av stavelementene, er den maksimale normalspenningen "*S-max*fra kombinasjonen ULS tilbakehentet. For Evaluering av konstruksjonsvekten er alle reaksjonskreftene "*FZ*", fra lasttilfellet egenlast hentet tilbake. Til slutt er den maksimale nedbøyningen fra kombinasjonen SLS i dekkene hentet til Dynamo. En oversikt over oppbygningen av utførelse av analysen er illustrert i Figur 3.16, sammen med en stykkevis forklaring i Tabell 3.5



Figur 3.16: Oversikt over utførelse av analyseprosessene i metode 1 - Design utforskning

1 - Tilpassing av global mesh i RSA
2 - Utføring av beregningene
3 - Innhenting av spenningstilstand i stavelementene
4 - Innhenting av maks nedbøyning i panelene
5 - Innhenting av opplagerreaksjonskrefter
6 - Kontroll om panelenes mesh-innstillinger er valgt som definert
7 - Samling av resultatene til videre evaluering

Tabell 3.5: Stykkevis forklaring av prosessene i utføring av analysen utviklet i metode 1 - Design utforskning

Verifisering og validering

Verifisering av modellen er gjennomført etter generering av et tilfeldig alternativ, med fem bjelker og fire underliggende søyler. Tverrsnittet for bjelkene og søylene er henholdsvis IPE 200 og HEA 240. Før gjennomgang av beregningsresultater bør det alltid gjennomføres en kontroll av objektene og forutsetninger definert tidligere gjennom analysemodellen. Elementene i RSA for dette tilfellet kontrolleres for antall, plassering, forbindelser, materialkvalitet og tverrsnitt, samt riktig definering av last og lastkombinasjoner. Elementtabellene fra RSA kan ses i Figur 3.17, og bekrefter at utregningene er gjennomført for en riktig definert konstruksjon. Som nevnt tidligere vil kalkulasjonsmodell av panelene settes til opprinnelig innstilling *Shell*, selv om dette ikke presiseres av paneltabellen i RSA.

Bar	Node 1	Node 2	Section	Material	Structure object	Releases
1	2	1	IPE 200	S355	Bar	Pinned-Pinned
2	3	27	IPE 200	S355	Bar	Pinned-Pinned
3	4	28	IPE 200	S355	Bar	Pinned-Pinned
4	5	29	IPE 200	S355	Bar	Pinned-Pinned
5	6	30	IPE 200	S355	Bar	Pinned-Pinned
6	7	2	HEA 240	S355	Bar	Pinned-Pinned
7	8	3	HEA 240	S355	Bar	Pinned-Pinned
8	9	4	HEA 240	S355	Bar	Pinned-Pinned
9	10	5	HEA 240	S355	Bar	Pinned-Pinned
10	11	6	HEA 240	S355	Bar	Pinned-Pinned
11	12	31	HEA 240	S355	Bar	Pinned-Pinned
12	13	32	HEA 240	S355	Bar	Pinned-Pinned
13	14	33	HEA 240	S355	Bar	Pinned-Pinned
14	15	34	HEA 240	S355	Bar	Pinned-Pinned
15	16	35	HEA 240	S355	Bar	Pinned-Pinned
16	17	36	HEA 240	S355	Bar	Pinned-Pinned
17	18	37	HEA 240	S355	Bar	Pinned-Pinned
18	19	38	HEA 240	S355	Bar	Pinned-Pinned
19	20	39	HEA 240	S355	Bar	Pinned-Pinned
20	21	40	HEA 240	S355	Bar	Pinned-Pinned
21	22	1	HEA 240	S355	Bar	Pinned-Pinned
22	23	27	HEA 240	S355	Bar	Pinned-Pinned
23	24	28	HEA 240	S355	Bar	Pinned-Pinned
24	25	29	HEA 240	S355	Bar	Pinned-Pinned
25	26	30	HEA 240	S355	Bar	Pinned-Pinned

Node	X (m)	Y (m)	Z (m)	Support code	Support
1	18,00	0,0	3,00	xxfff	SideStøtte
2	0,0	0,0	3,00	xxfff	SideStøtte
3	0,0	6,25	3,00	xxfff	SideStøtte
4	0,0	12,50	3,00	xxfff	SideStøtte
5	0,0	18,75	3,00	xxfff	SideStøtte
6	0,0	25,00	3,00	xxfff	SideStøtte
7	0,0	0,0	0,0	xxxfff	Pinned
8	0,0	6,25	0,0	xxxfff	Pinned
9	0,0	12,50	0,0	xxxfff	Pinned
10	0,0	18,75	0,0	xxxfff	Pinned
11	0,0	25,00	0,0	xxxfff	Pinned
12	6,00	0,0	0,0	xxxfff	Pinned
13	6,00	6,25	0,0	xxxfff	Pinned
14	6,00	12,50	0,0	xxxfff	Pinned
15	6,00	18,75	0,0	xxxfff	Pinned
16	6,00	25,00	0,0	xxxfff	Pinned
17	12,00	0,0	0,0	xxxfff	Pinned
18	12,00	6,25	0,0	xxxfff	Pinned
19	12,00	12,50	0,0	xxxfff	Pinned
20	12,00	18,75	0,0	xxxfff	Pinned
21	12,00	25,00	0,0	xxxfff	Pinned
22	18,00	0,0	0,0	xxxfff	Pinned
23	18,00	6,25	0,0	xxxfff	Pinned
24	18,00	12,50	0,0	xxxfff	Pinned
25	18,00	18,75	0,0	xxxfff	Pinned
26	18,00	25,00	0,0	xxxfff	Pinned

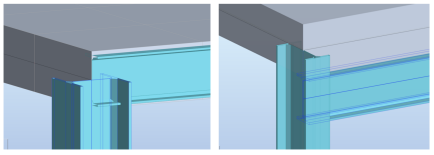
Panel	Thickness	Material	Meshing type	Structure object	Calculation model	FE type
26	TH_230_CONCR	B35	Coons	Panel	-	-
27	TH_230_CONCR	B35	Coons	Panel	-	-
28	TH_230_CONCR	B35	Coons	Panel	-	-
29	TH_230_CONCR	B35	Coons	Panel	-	-

Case	Load type	List			
3 Egenvekt	self-weight	1to29	Whole structur	-Z	Factor=1,00
1 Permanent	(FE) uniform	26to29	PX=0,0	PY=0,0	PZ=-1,20
2 Nyttelast	(FE) uniform	26to29	PX=0,0	PY=0,0	PZ=-3,00

Combinations	Name	Analysis type	Combination	Case nature	Definition
4 (C)	Comb_SLS	Linear Combinati	SLS	live	(1+2+3)*1.00
5 (C)	Comb_ULS	Linear Combinati	ULS	live	(1+3)*1.35+2*1.05

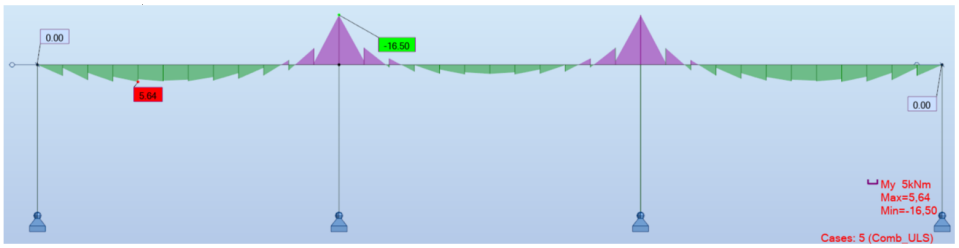
Figur 3.17: Elementtabeller fra RSA som validerer definert analysemodell i metode 1 - design utforskning

Det må spesifiseres at bjelkenes fulle effekt av annet arealmoment blir ikke tatt med i beregningene, da bjelkene defineres i samme høyde som panelene. Dette kan for en mer detaljert analyse justeres, ved bruk av funksjonen "*Offsets*"i RSA. Ved å definere bjelkene i underkant av dekket, som illustrert i Figur 3.18, vil maksimal nedbøyning i bjelkene avta med 7% for lastkombinasjonen ULS.



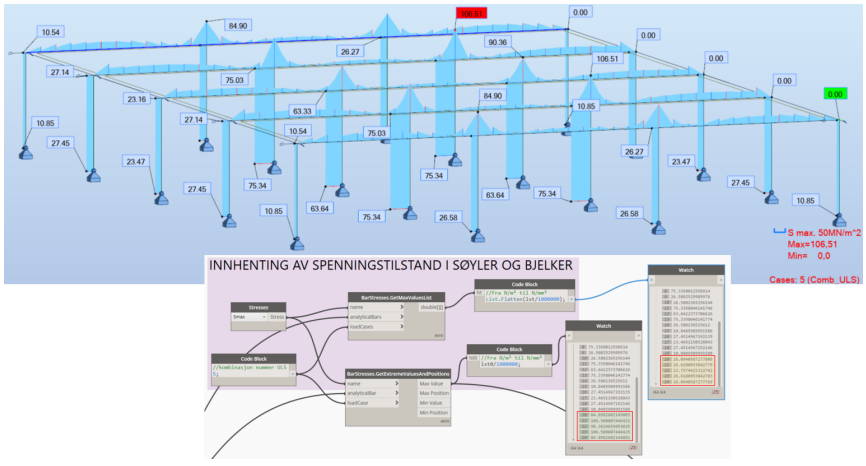
Figur 3.18: Bidrag fra annet arealmoment ved bruk av funksjonen *Offsets* til høyre, og definert modell i denne oppgaven til venstre

Etter at analyse modellen er validert, undersøkes konstruksjonssystemets lastoverføring fra panelene ned til opplagerene. Momentdiagrammet i stavelementene, se Figur 3.19, bekrefter leddet forbindelsen i stavelementene med kontinuerlige bjelker i overkant. Den hakkete utformingen av momentdiagrammet i bjelkene, er på grunn av lastoverføring fra FE-nodene i panelene til bjelkene.



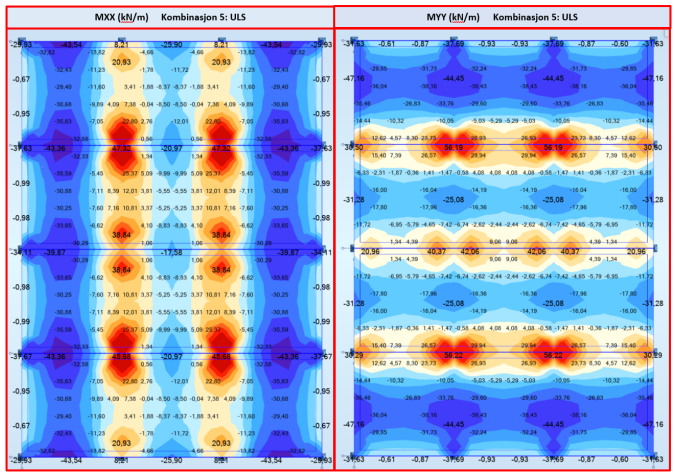
Figur 3.19: Momentdiagram (My) for ytterste bjelker og underliggende søyler

Det er observert ulike verdier for spenningstilstand av bjelkene i Dynamo og RSA. Dette ved bruk av Dynamo-noden "*BarStresses.GetMaxValuesList*". Forskjellen skyldes at denne noden søker største spenning i stavelementenes halve lengde. Dette vil samsvare kun for maksimal spenningstilstand i søylene, da disse er utsatt for en konstant trykkspenning. Ved bruk av Dynamo-noden "*BarStresses.GetExtremeValuesAndPositions*"isteden, søkes største spenningstilstand i hele elementets lengde. Figur 3.20 illustrer spenningstilstand for stavelementene i RSA, og den opprettede Dynamo-noden som benyttes videre i modellen.



Figur 3.20: Spenningstilstand for stavelementene i RSA, og Dynamo nodene som søker etter største normalpenning i elementene

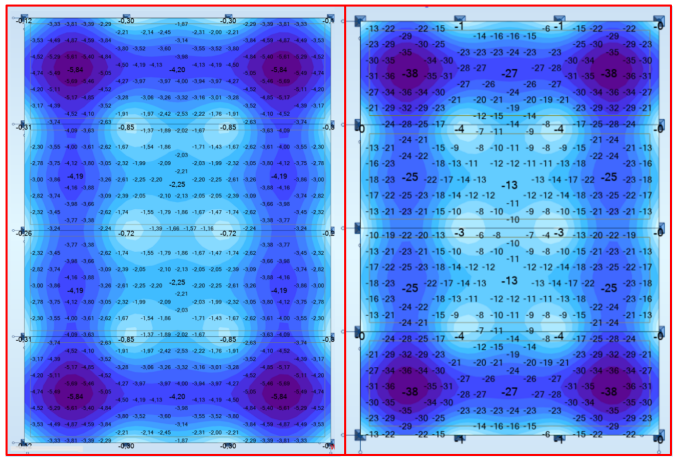
Til slutt undersøkes dekkets nedbøyning ved å først undersøke platenes samvirke og kraftoverføring for valgt kalkulasjonsmetode og mesh. Kalkuleringsmodellen kan godkjennes dersom det ikke er observert ujevn kraftoverføring mellom nodene, og samvirket mellom panelene er kontinuerlig. Dette kan ses av Figur 3.21, som illustrer momentkart MXX og MYY for platene, med reduksjon av krefter nær søyler. De analytiske resultatene for valgt mesh er godkjent, ved å sammenligne resultatene for mesh-størrelse 0,1 meter, for både fire- og åtte-noders elementer.



Figur 3.21: Momentkart MXX og MYY for platene i RSA med reduksjon av krefter nær søyler

Da denne optimaliseringsprosessen er ment for en tidlig prosjekteringsfase, er dimensjonering av armering i dekkene ikke tatt hensyn til. For videre evaluering av alternativene, er det samtidig et behov for å kunne indikere dekkenes nedbøyning. For indikasjon av nedbøyningen, er det valgt å ta utgangspunkt i et dekke med hovedarmering Ø12 og CC250 i begge retninger i topp og bunn av dekket. Dette armeringsforholdet er som regel ønsket av entreprenører for praktiske utførelsesårsaker, og er over minimumskravene til hovedarmering etter (EC2) NS-EN 1992-1-1:2004.

Etter at kalkulasjonsmetodene for panelene er godkjent, verifiseres nedbøyning av panelene. Nedbøyningsresultater i Dynamo benytter RSA API for tilbakeføring av de største analytiske nedbøyningsverdiene fra FEM-kalkulasjonen. Disse verdiene kan bekreftes å samsvare med RSA. Nedbøyningsverdiene er sammenlignet med dekket, etter at armeringen er kalkulert og dimensjonert i RSA i henhold til NS-EN 1992-1-1:2004. Sammenligning av resultatene er illustrert i Figur 3.22, som viser til store forskjeller mellom de analytiske resultatene til venstre, og resultater etter påført armering til høyre.



Figur 3.22: Sammenligning av elastisk nedbøyning etter gjennomført FEM-analyse og nedbøyning etter kalkulering av nødvendig armering i henhold til NS-EN 1992-1-1:2004

Denne forskjellen reflekteres fra kalkuleringsalgoritmen i RSA, for beregning av nedbøyning i skall- og plateelementer. Algoritmen benytter FEM for beregning av den analytiske nedbøyningen *elastisk nedbøyning"*, basert på en isotropisk elastisk materiale. Deretter benyttes den elastiske nedbøyningen og FEM-analysen for beregning av den "*ekte* nedbøyningen i panelene, ved endring av stivheten basert på armeringsforhold og riss i betongen. Beregning av den ekte nedbøyningen er lik produktet av den elastiske nedbøyningen og stivhetskoeffisient D/B, se Formell 3.1.

$$U_R^i = U^i * \frac{D}{B} \tag{3.1}$$

Der:

- U_R^i = ekte forskyvning av beregningspunktet i av en plate som tar hensyn til riss og beregnet armering
- U^i = Elastisk forskyvning av beregningspunktet i
- D = dekkens stivhet forutsatt materialets elastisitet (som i FEM-beregninger)
- B = Ekvivalentstivhet av platen, beregnet i forhold til element riss, reologiske effekter, beregnede armering og etc.

En slik tilnærming benyttes ned til lineær skalering av individuelle elastiske forskyvninger. Beregningsmetoden av stivhetskoeffisienten sammen med den ekvivalente stivheten, kan ses i Formel 3.2. Beregningsalgoritmen for ekvivalent stivhet utføres for to retninger, og benytter en veid gjennomsnittfaktor c_f av komponent stivhetsverdier (AUTODESK, 2019). Den veide gjennomsnittsfaktoren og den elastiske stivheten kan henholdsvis ses i Formell 3.3 og Formell 3.4.

$$(\frac{D}{B})^i_{xy} = (\frac{D^i}{B^i})_{xy} = \frac{D}{c_f * B_x + (1 - c_f) * B_y} \tag{3.2}$$

Den veide gjennomsnittsfaktoren c_f:

(3.3)

- $\frac{|M_{xx}|}{|M_{yy}|} > 4, \rightarrow C_f = 1$
- $0.25 \leq \frac{|M_{xx}|}{|M_{yy}|} \leq 4, \rightarrow C_f = 0.5 + \frac{2}{3} * \frac{|M_{xx}| - |M_{yy}|}{max(|M_{xx}|;|M_{yy}|)}$
- $\frac{|M_{xx}|}{|M_{yy}|} < 0.25, \rightarrow C_f = 0$

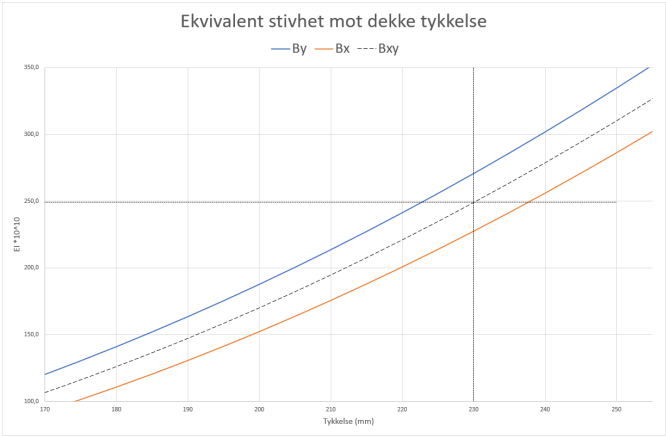
Den elastiske stivheten:

$$D = E * \frac{1[m] * h^3}{12} \tag{3.4}$$

Denne oppgaven benytter en forenklet beregningsprosess for å indikere den ekte nedbøyningen i panelene, basert på kalkulasjonsprinsippet RSA. Den ekte nedbøyningen beregnes ved å multiplisere en stivhetskoeffisient basert på et armert dekke, til den elastiske nedbøyningen som er hentet via RSA API. Denne prosessen gir brukeren mulighet for å undersøke nedbøyningseffekter ved endring av dekkenstverrsnitt og hovedarmeringen som en varierende parameter.

Den realistiske nedbøyningen er avhengig av hvilken tilstand tverrsnittet er i, slikt at effekten av reologiske og opprising kan tas med i beregningene. En realistisk antagelse for dekket vil være at tverrsnittet er i Stadium II (opprisset) midt i feltene og over støttene, og i Stadium I (uopprisset) over et område mellom felt og støttene (Stemland and Johansen.H, 2016).

Beregning av stivhetskoeffisienten benytter den elastiske stivheten D" vist i Formel 3.4, og en forenklet versjon av den ekvivalente stivheten B" vist i Formel 3.2. Den forenklede ekvivalentstivheten tar utgangspunkt i et opprisset betongtverrsnitt i Stadium II, dette for beregning av ekvivalente stivheter B_x og B_y. Den veide koeffisienten i denne oppgaven vil ha lite påvirkning på den ekvivalente stivheten B_{xy}, da det ikke er lokale armeringsforsterkninger i dekken. I denne oppgaven er den ekvivalente stivheten B_{xy} valgt som et gjennomsnitt av stivheten i x og y-retning, $C_f = 0.5$. Ekvivalente stivheten B_x og B_y, sammen med den vektede ekvivalent stivheten B_{xy} for ulike dekketykkelser er illustrert i Figur 3.23. B_x og B_y er beregnet med hovedarmering i y-rettning.



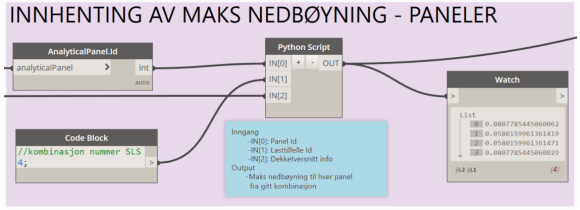
Figur 3.23: Sammenligning av ekvivalent stivhet B_x og B_y for en dekke med hoved armering i y-retning

Beregning av stivhetskoeffisienten, gjennomføres i Python-koden sammen med innhenting av nedbøyningene. De ekvivalente stivhetene B_x og B_y beregnes for dekketversnitt i Stadium II etter Formel 3.5 og 3.6, (Cantero, 2016).

$$I_{c2} = \frac{1}{2}\alpha^2(1 - \frac{\alpha}{3})bd^3 \tag{3.5}$$

$$\alpha = \sqrt{(\eta\rho) + 2\eta\rho} - \eta\rho \quad ; \eta = \frac{E_s}{E_{cm}} \quad ; \rho = \frac{A_s}{bd} \tag{3.6}$$

Resultatene etter denne justeringen er illustrert i Figur 3.24. Disse er høyere enn resultatene for den ekte nedbøyningen i RSA, vist i Figur 3.22. Dette er på bakgrunn av antagelsen av hele dekketverrsnittet i Stadium II, og lokale forsterkninger etter dimensjonering av armeringen i dekket. Den beregnede nedbøyningen bør benyttes forsiktig, da dette er gjennomført kun for indikering av mer reelle nedbøyningsverdier.



Figur 3.24: Nedbøyningsresultatr etter justering av nedbøyningen med stivhetskoeffisienten

3.1.4 Evaluering av analyse resultatet

Gjennom evalueringsprosessen er alternativene av den parametriske modellen evaluert og rangert, slikt at en kan lettere indikere de optimale løsningene. Rangering av alternativene gjennomføres gjennom et Python-kode, og benytter en samling av informasjon til å poengtere hvert tilfelle basert på kriterier og betingelse gitt av brukeren. Gjennom dette delkapitlet blir brukeren kjent med rangeringssystemets oppbygning, samt bakgrunn for valg og endring av kriterier og betingelse.

Optimaliseringsprosessen i denne oppgaven har som mål å identifisere optimale dekkesystemer, med et objektivt mål om lavest mulig material vekt. Med dette objektive målet alene må det gjennomføres en konstruksjonssikkerhetskontroll for hver av tilfellene. I en algoritmisk optimaliseringsprosess vil resultatet gå mot et tilfelle med lengste dekkespenn og færrest stavelementer, uten å ta hensyn til konstruksjonens sikkerhet. I denne oppgaven er det laget et poengsystem, som vil bevare konstruksjonssikkerheten for gitte kriterier og betingelser.

Rangering og poengtering av alternativene kan gjennomføres på utallige måter, med bakgrunn i ønsket presisjon av optimale alternativer. Kriteriene i denne oppgaven avdekker alternativene som er underdimensjonert, der disse alternativene ikke kan tolereres. Betingelsene har som mål å skape en avhengighet mellom elementenes utnyttelse og konstruksjonens egenvekt. Rangering av alternativene er basert på tre objektive mål: konstruksjonsvekt, utnyttelse av stavelementer og utnyttelse av FE-paneler.

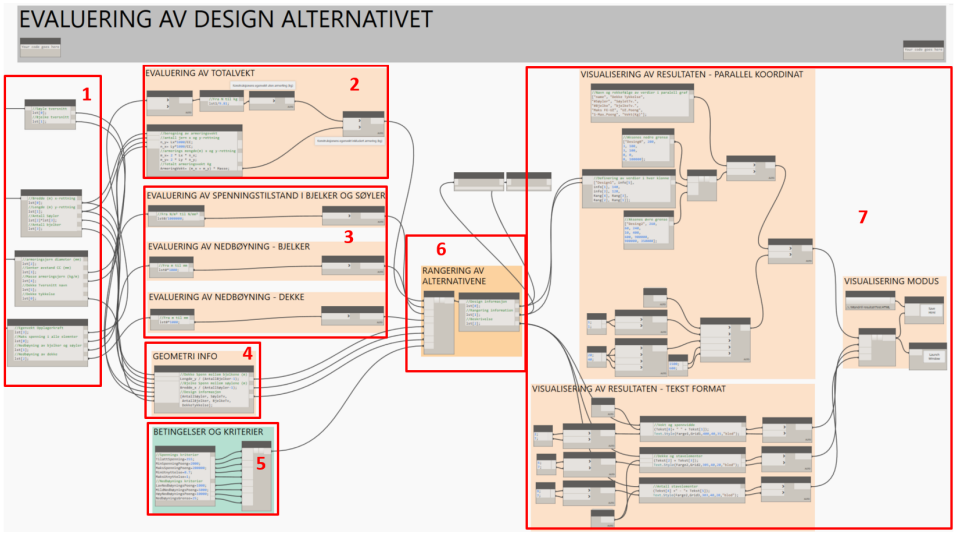
Generelt kan rangering av alternativene utvikles enten som et SOO- eller MOO-problem. Fordelene og ulempene for SOO- og MOO-problemer er utdypet i Kapitel 2.2 Optimaliseringstyper og metoder. For denne oppgaven kan rangering av alternativene basert på et SOO-problem utvikles enkelt, ved å konvertere alle objektive mål til et felles mål kalt for straffet vekt "*Penalty Wigh*". Oppbygningsprinsippet for rangering av alternativene som et SOO-problem er illustrert i Figur 3.25.

Figur 3.25: Oppbygningsprinsippet for rangering av alternativene som et SOO-problem

Da SOO-optimaliseringstyper gir lite grunnlag for å undersøke påvirkningseffekten av hvert objektivt mål, er rangeringsprosessen videre i denne oppgaven utviklet som et MOO-problem. På denne måten kan flere optimale løsninger sammenlignes og vurderes. Generelt er det enklere å lage en slik rangeringsprosess, men derimot vil beslutningsprosessen av metoden være vanskeligere ved et manuelt optimalisering. Oppbygningsprinsippet av rangering som et MOO-problem er illustrert i Figur 3.26.

Figur 3.26: Oppbygningsprinsippet for rangering av alternativene som et MOO-problem

Før evaluering og rangering av alternativene, er nødvendige geometriske resultater og analyseresultater samlet og organisert i starten av denne prosessen. Det er viktig å være oppmerksom på at Dynamo og RSA kan benytte forskjellige enheter for innhentede resultater. Den totale konstruksjonsvekten regnes ved å summere resultatene fra opplagerreaksjoner fra lasttilfellen egenlast, og legge til beregnet armeringsmengde i dekket. Armeringsmengde beregnes basert på informasjonen gitt ved oppretning av dekket. Videre er alle spennings- og nedbøyningsverdiene omformet til henholdsvis Mpa og mm. Disse verdiene sammen med en liste av betingelser og kriterier blir sendt til en Python-kode, som sørger for rangering av alternativene. Resultatet fra alle de objektive målene er visualisert i en parallellkoordinat-graf, ved bruk av Dynamo-pakken Mandrill. Oppbygning av prosessen evaluering av resultatene er illustrert i Figur 3.27, sammen med et punkt forklaring i Tabell 3.6.



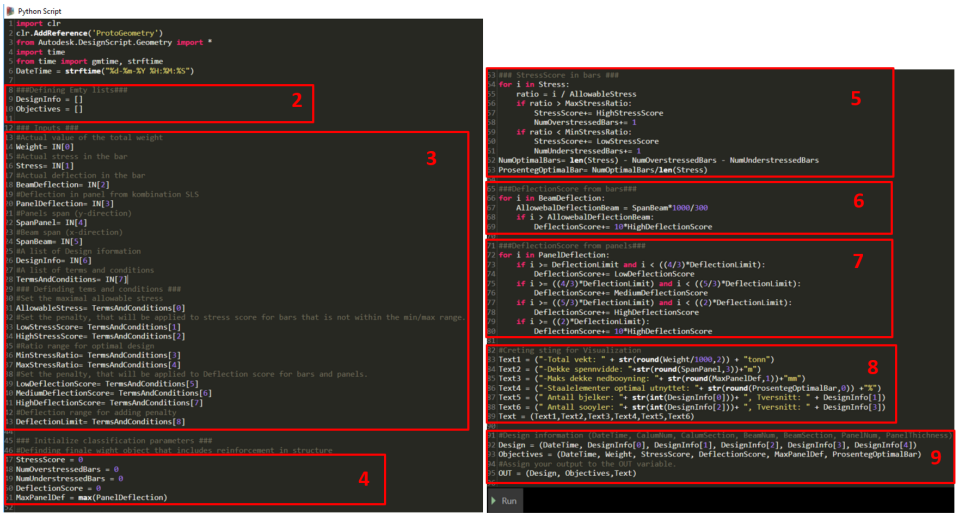
Figur 3.27: Oppbygning av prosessen evaluering og rangering av resultatene, med prosessen for visualisering av resultater i en parallell koordinat-graf

1 - Samling og sortering av nødvendig informasjon fra hele modellen
Tverrsnitt av stavelementene
Dekke tverrsnitt og armeringsinformasjon
Geometrisk informasjon og antall elementer
Analyseresultater
2 - Beregning av egenvekt inkludert armering
Beregning av konstruksjonsvekt fra lasttilfellet egenvekt til Kg
Beregning av armeringsvekt
3 - Konvertering av spenninger fra N/m2 til Mpa og nedbøyninger fra m til mm
4 - Beregning og samling av geometrisk informasjon
Beregning av spennvidde for dekker og bjelker
Samling av elementenesinformasjon i en liste
5 - Definering av kriterier og betingelser
Utnyttelseskriterier for stavelementene - spenningskriterier
Utnyttelseskriterier for FE - nedbøyningskriterier
6 - Python kode til evaluering og poengtering av alternativer
7 - Visualisering av alternativene i en parallellkoordinat-graf

Tabell 3.6: Stykkevis forklaring av prosessene i evaluering av resultatene

Den automatiske evalueringsprosessen av konstruksjonen utføres ved bruk av toleranser angitt av brukeren. Disse toleranseverdiene kan endres ut ifra konstruksjonstype, og benyttes til å redegjøre sikkerhetskoeffisienter. Evaluering og rangering av alternativene utføres ved å pålegge et straffepoeng til objektive målene, dersom stressforholdet eller nedbøyningen er enten over eller under et gitt brukerdefinert utnyttelsesforhold.

Utnyttelse av stavelementene er definert gjennom et objektivt mål kalt for spenningspoeng. Poengteringsystemet beregner utnyttelsesgraden i hvert element, der overbelastede elementene får et høyt straffepoeng, da de ikke kan tolereres. De underbelastede elementene får en mindre straff. Dekkets utnyttelsesforhold er definert gjennom et objektivt mål, kalt nedbøyningspoeng. Dette poengteringssystemet baseres på at dekken skal kunne opprettholdes kun med den definerte hovedarmeringen, uten behov for lokale forsterkninger. Brukeren angir en nedbøyningsgrense for dekket, der nedbøyningene over denne grensen får et straffepoeng. Denne grensen baseres på det tidspunktet det er behov for lokale armeringsforsterkninger i dekket. Da store dekkespenninger blir urealistiske å dimensjonere, er størrelsen på straffepoenget basert på overskredet størrelse fra den angitte nedbøyningsgrensen. Python-koden som sørger for poengfordeling og rangering av alternativene er illustrert i Figur 3.28, sammen med stykkevis forklaring i Tabell 3.7.



Figur 3.28: Oppbygning av den automatiske evalueringskoden i Python, som sørger for poengfordeling av elementene basert på utnyttelsesgraden

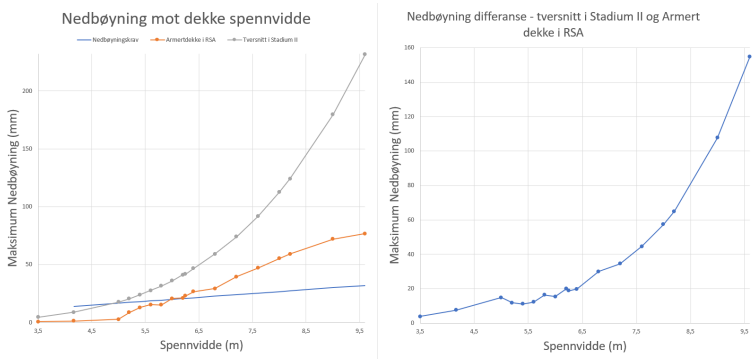
1 - Importering av pakker
2 - Oppretting av utgangslister
3 - Spesifisering av inngangsverdier
4 - Definering av objektive mål
5 - Spenningspoeng fra utnyttelse i stavelementer
6 - Nedbøyningspoeng fra bjelke elementer
7 - Nedbøyningspoeng fra FE-paneler
8 - Sammensetting av alternativens informasjon, tekstformat
9 - Samling og klargjøring av utgangsverdier

Tabell 3.7: Stykkevis forklaring av den automatiske evalueringskoden i Python

Nedbøyningsgrensen er basert på sammenligning av den beregnede nedbøyningen i Dynamo med hele tverrsnittet i Stadium II, og den virkelige nedbøyningen i RSA etter kalkulering for nødvendig armering. Sammenligningen er gjennomført for varierende dekkespennvidde, med hensikt om å kunne indikere grensen for behov av ekstraarmering, ettersom dekkets spennvidde øker. Resultatene fra sammenligningen er illustrert i Figur 3.29, med nedbøyningsoppførselen av dekkene til venstre og differansen av resultatene til høyre.

Den grå kurven representerer de beregnede nedbøyningsverdier i denne oppgaven, for forskjellige dekkespennvidder. Den jevne eksponentielle veksten av kurven forårsakes av antagelsen med fullopprisset betongtverrsnitt fra start, og konstant armeringsforhold. Den virkelige nedbøyningsoppførselen kan ses fra den beregnede nedbøyningen i RSA, ettersom beregningene tar hensyn til lokal opprissing i dekket. En kan fra grafene anslå at betongens opprissing startes fra spennvidde fem meter. Fra det tidspunktet reduseres stivheten, som en effekt av de lokale opprissingsområder. Videre oppstår det svingninger i

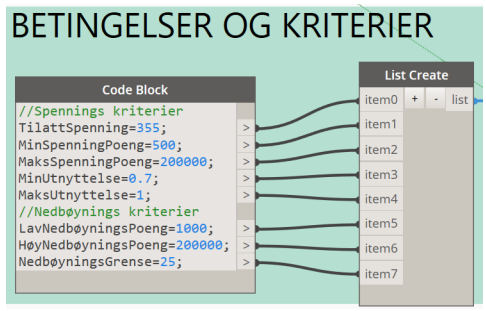
nedbøyningen, basert på spredning av riss i betongen og lokale armeringsforsterkninger i disse områdene. Nedbøyningsgrensen er valgt for en spennvidde som forårsaker lokale armeringsforsterkninger i RSA, for deretter å hente verdien for samme spennvidde fra grafen med tverrsnitt i Stadium II. Nedbøyningsgrensen er her valgt til 25 mm.



Figur 3.29: Sammenligning av nedbøyningsverdier av dekke med ekvivalent tversnitt i Stadium II og verdier etter kalkulering av nødvendig armering i RSA

De øvrige kriterier og betingelser er illustrert i Figur 3.30. Spenningskriterier definerer det optimale utnyttelsesområdet for stavelementene. I denne oppgaven vil elementene som er under dette området få lavspenningspoeng 500, og overskridende elementer får storspenningspoeng 200000. Nedbøyningskriteriene straffer nedbøyninger som har overskredet nedbøyningsgrensen basert på den eksponentielle differansen er illustrert i Figur 3.29. All nedbøyning over 125 mm får et høy nedbøyningspoeng. Nedbøyningene poengteres som følge:

$$Hvis: \quad Grense < U_z < 5Grense \quad \rightarrow LavedboyningsPoeng \times e^{0,031U_z}$$
$$Hvis: \quad 5Grense \leq U_z \quad \rightarrow HoyedboyningsPoeng$$



Figur 3.30: Oversikt over evalueringskriterier og betingelser som er benyttet i denne oppgaven

3.1.5 Tilkobling til RSA API via Dynamo

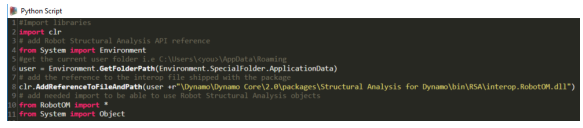
Den parametriske modellen vist gjennom dette kapittelet har hovedsakelig benyttet Dynamo Structural Analysis-pakken for gjennomføring av arbeidsprosesser i RSA. Da nodene i pakken er best tilegnet for oppretting av modeller med kun stavelementer, er pakkens begrensninger overkommet ved bruk av RSA API gjennom Python-koder. Hittil i oppgaven er RSA API benyttet til å definere følgende oppgaver:

- Oppretting av nye opplagerbetingelser
- Oppretting av nytt dekketverrsnitt
- Frigjøring av dekkekanter
- Påføring av flatelaster til paneler
- Definering av lastkombinasjoner
- Definering av global mesh for FE
- Tilbakehenting av resultater fra paneler

Da Structural Analysis-pakken ikke er *open source*, kan kombinasjonen av pakken og bruk av RSA API være tidskrevende og utfordrende. Objektene som er definert gjennom Analysis pakken er *wrapped*, slikt at disse objektene ikke kan bli benyttet direkte via RSA API. For å gjennomføre endringer til objekter definert gjennom Analysis-pakken, er ID-nummeret benyttet til å kommunisere med disse objektene via RSA API. På samme måte blir ikke definerte objekter gjennom RSA API gjenkjent av Analysis-pakken. Dette skaper utfordringer, da kalkuleringsnoden i Analysis-pakken trenger å ha med alle definerte objekter.

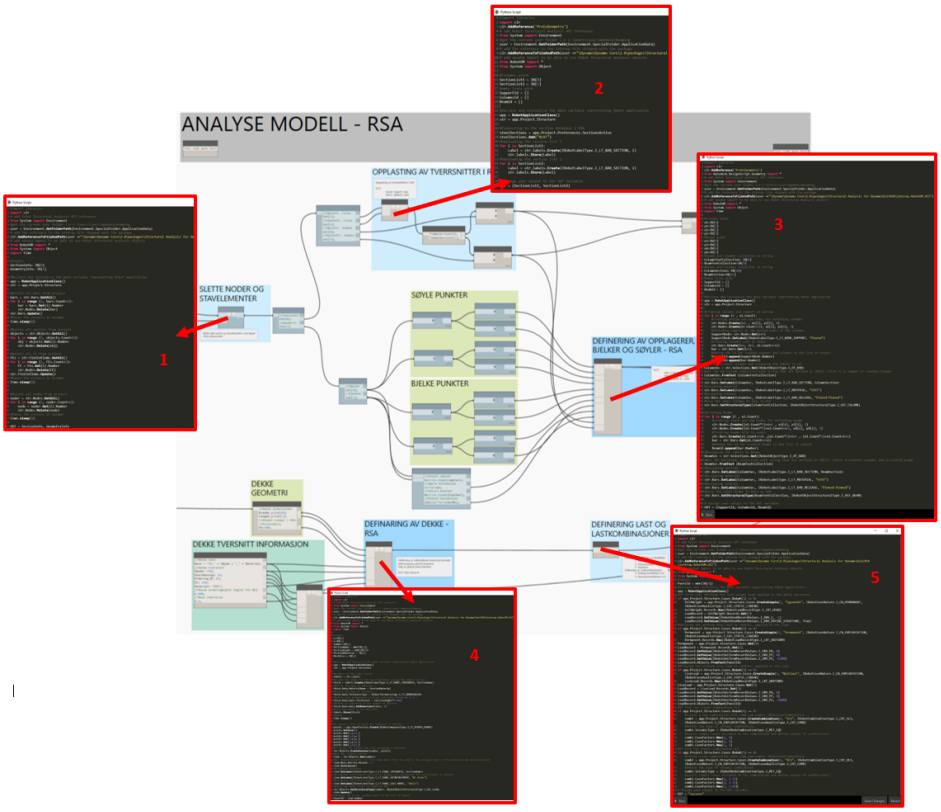
Uten å gå dypere inn i dette, er konsekvensen av å benytte RSA API gjenspeilet i denne oppgaven med en rekke feilmeldinger, som følge av prosessene for påføring av laster og definering av lastkombinasjoner. Disse prosessene fungerer henholdsvis for den manuelle optimaliseringsmetoden, der feil i modellen enkelt kan oppdages ved å kjøre grafen. Da den parametriske modellen skal benyttes i en algoritmisk optimaliseringsprosess, er det viktig at det ikke oppstår noen feilmeldinger gjennom utførelsen. Videre i denne oppgaven er modellen redigert slik at all kommunikasjon med RSA gjennomføres via RSA API, uten bruk av Structural Analysis-pakken.

Tilkobling til det åpnede RSA-dokumentet og API skjer gjennom .dll-filen interop.RobotOM.dll. denne filen ligger i OS-systemet Appdata-mappen i Dynamo-pakken Structural Analysis,og også inn i installasjonens fil av RSA. Figur 3.31 illustrerer tilkoblingen til RSA API via Python i Dynamo.



Figur 3.31: Tilkoblingen til RSA API via Python i Dynamo

Definering av analyse modellen gjennom RSA API er illustrert i figur 3.32, med stykkevis forklaring i Tabell 3.4. Analysemodellen starter med å slette alle elementene i det åpne RSA-dokumentet, slikt at definerte objekter fra forrige gjennomkjøring blir slettet. Utgangen av denne noden er en liste med det geometriske skjelettet og en liste med tverrsnitt. Tverrsnittene lastes opp fra databasen i RSA i en egen kode, før valgt tverrsnitt sendes videre til koden som definerer alle stavelementene og opplagerbetingelser. Bjelkene og søylene defineres ved å benytte start- og endepunktkoordinater fra det geometriske skjelettet. Dekket defineres fra en liste med tverrsnittsinformasjonen gitt av brukeren, for så å sette nødvendige beregningsinformasjon for disse FEne. Etter at dekket er definert, sendes panel-ID videre til Python-koden som definerer alle laster og lastkombinasjoner, samt påføring av lastene på dekket. Modellens forutsetninger er lik som modellen vist i Kapittel 3.1.2.

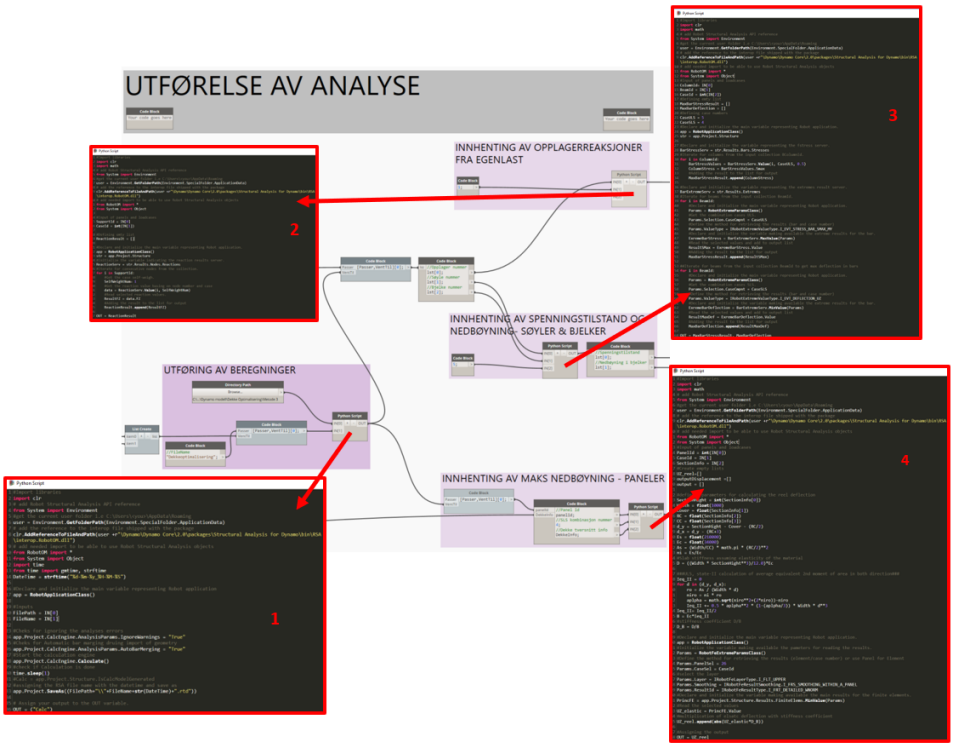


Figur 3.32: Oppbyggning av analysemodell gjennom RSA API i Dynamo

1 - Slette noder og stavelementer
Slette alle stavelementer
Slette alle FE noder
Slette alle noder
2 - Opplasting av tverrsnittslister
Tilkoblingen til database
Definering av tverrsnitt fra en gitt liste
3 - Definering av opplagrer, søyler og bjelker
Definering av søylenes start- og endenoder
Definering av opplagerbetingelser i nederste noden
Definering av søyler med start- og endenoder
Tildeling av tversnitt, materialer og endeforbindelser
Definering av bjelker start- og endenoder
Definering av bjelker med start- og endenoden
Tildeling av tversnitt, materialer og endeforbindelser
4 - Definering av dekke
Definering av dekketverrsnitt
Definering av dekkegeometri
Tildeling av tverrsnitt og kalkulering metode
Parametervalg av global mesh-størrelse
5 - Definering av laster og lastkombinasjoner
Definering av egenlast for hele konstruksjonen
Definering av permanentlast og påføring til dekke
Definering av nyttelast og påføring til dekke
Definering av lastkombinasjon SLS
Definering av lastkombinasjon ULS

Tabell 3.8: Stykkevis forklaring av Python-kodene som er benyttet til definering av analysemodellen gjennom RSA API i Dynamo

Kalkulering og utføring av analysene gjennomføres etter at analysemodellen er definert i RSA. Utføring av analysen gjennom RSA API er illustrert i figur 3.33, med et punkt forklaring i Tabell 3.9. Etter gjennomføring av kalkuleringen og lagring av analysemodellen, tilbakehentes alle nødvendige resultater forklart i Kapittel 3.1.3.



Figur 3.33: Oppbyggning av analysemodell gjennom RSA API

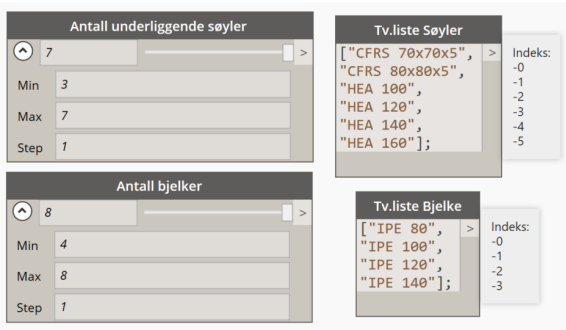
1 - Utføring av beregninger
Valg av analyseparametere
Starte kalkuleringmotor
Lagring av prosjektet
2 - Innhenting av opplagerreaksjoner
Innhenting av resultater fra egenlast
3 - Innhenting av spenningstilstand og nedbøyning i stavelementer
Innhenting av maks spenning i søyler fra lastkombinasjon ULS
Innhenting av maks spenning i bjelker fra lastkombinasjon ULS
Innhenting av maks nedbøyning i bjelker fra lastkombinasjon SLS
4 - Innhenting av nedbøyning fra dekke
Beregning av stivhetskoefisienten
Innhenting av maks elastisk nedbøyning fra lastkombinasjon SLS
Beregning av ektenedbøyning i dekke

Tabell 3.9: Stykkevis forklaring av Python-kodene som er benyttet til kalkulering og utførelse av analysen gjennom RSA API i Dynamo

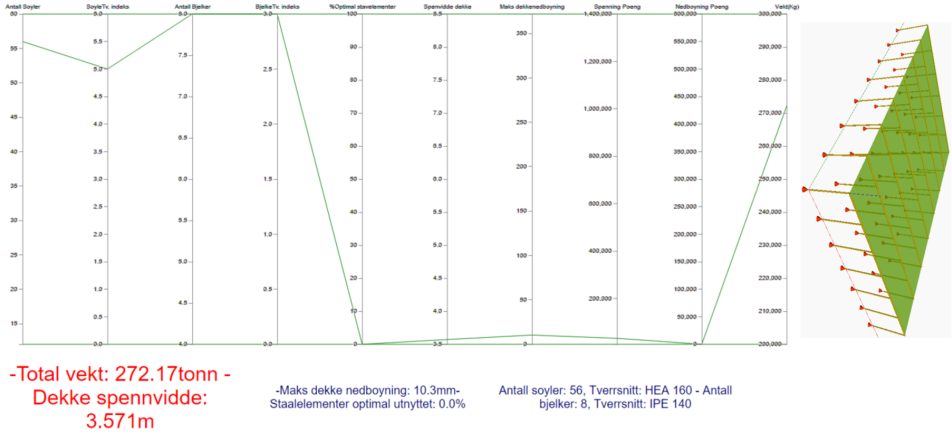
3.1.6 Oppsummering og resultater

Gjennom dette kapittelet er det vist til oppbygning av et parametrisk dekkesystem, der denne modellen benyttes videre i en algoritmisk optimaliseringsprosess. Dekkesystemet som har blitt illustrert gjennom dette kapittelet har tatt utgangspunkt i en rektangulær toveisplate, tiltenkt for et kontorbygg. Den geometriske utformingen av modellen ble først definert i Dynamo, for deretter å definere geometrien og analysenes forutsetninger i RSA. Definering og utførelse av analysemodellen benyttet i første omgang er Structural Analysis pakken, der begrensningene til pakken er overkommet ved bruk av RSA API. Da Analysis pakken hovedsakelig er beregnet til modeller med kun stavelementer, har bruk av RSA API sammen med pakken ført til noen feilmeldinger. Dette gjaldt hovedsakelig for påføring og definering av laster og lastkombinasjoner. Videre ble modellen redigert, slik at all kommunikasjon med RSA gjennomgår ved bruk av RSA API, via Python-koder i Dynamo. Det er observert en kraftig reduksjon av total gjennomføringstid, fra tre minutter til underkant av 30 sekunder.

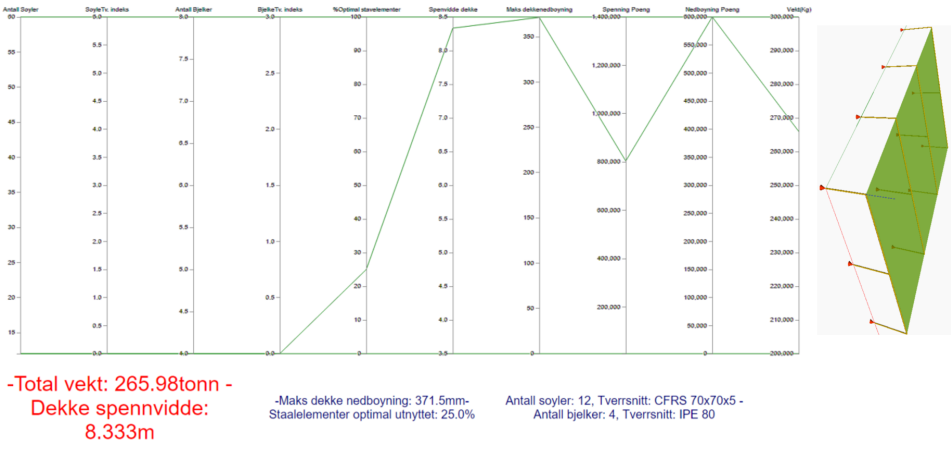
Da hovedformålet med dette kapittelet var å bygge en parametrisk modell som kan benyttes i en algoritmisk optimaliseringsprosess, er denne modellen benyttet til å finne øvre og nedre grense av variablene. Variablenes øvre og nedre grense er illustrert i figur 3.34, med visualisering av alternativene henholdsvis i Figur 3.35 og Figur 3.36.



Figur 3.34: Illustrasjon av variablenes øvre og nedre grenser



Figur 3.35: Visualisering av resultater av alternativ øvre grense



Figur 3.36: Visualisering av resultater av alternativ nedre grense

3.2 Metode2 Brute force søk

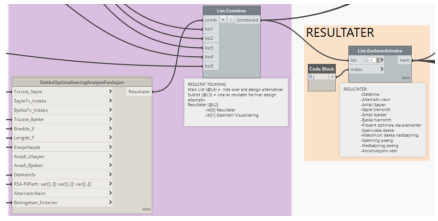
Dette kapitlet handler om utvikling og gjennomføring av en algoritmisk optimaliseringsprosess, for det parametriske dekkesystemet utviklet gjennom kapittel 3.1. Den algoritmiske optimaliseringen baseres på metoden Brute force search, og genererer et gitt antall tilfeldige alternativer for evaluering av en optimal løsning. Denne optimaliseringsprosessen er beregnet for en tidlig prosjekteringsfase, med formål om å kunne identifisere optimal oppbygging for et gitt dekke.

Optimaliseringsprosessen i denne oppgaven benytter den utviklede parametriske modellen som en analysefunksjon, og gjennomfører en komplett analyse for 600 tilfeldige genererte alternativer. Variablenes domene som benyttes for generering av alternativene er presentert i resultatene fra Metode 1 - Design utforsking, Figur 3.34. Kombinering av disse variablene skaper 600 mulige utfall, basert på domene for antall vanlige og underliggende søyler, og tverrsnittsvalg for søyler og bjelker.

Resultatene av alternativene evalueres og visualiseres som både SOO- og MOO-problem, med tre objektive mål: materialvekt, utnyttelse av dekket og utnyttelse av stavelementer. Resultatene av de objektive målene og rangering av analysen gjennomføres via Python-koden, som er nærmere beskrevet i Kapittel 3.1.4. Resultatene fra alternativene presenteres ved bruk av Dynamo-pakken Mandrill, utviklet av Archi+lab for visualisering av data i Dynamo. Alternativenes informasjon og resultater fra alle alternativene er eksportert til Excel og vedlagt sammen med resultater fra modellen.

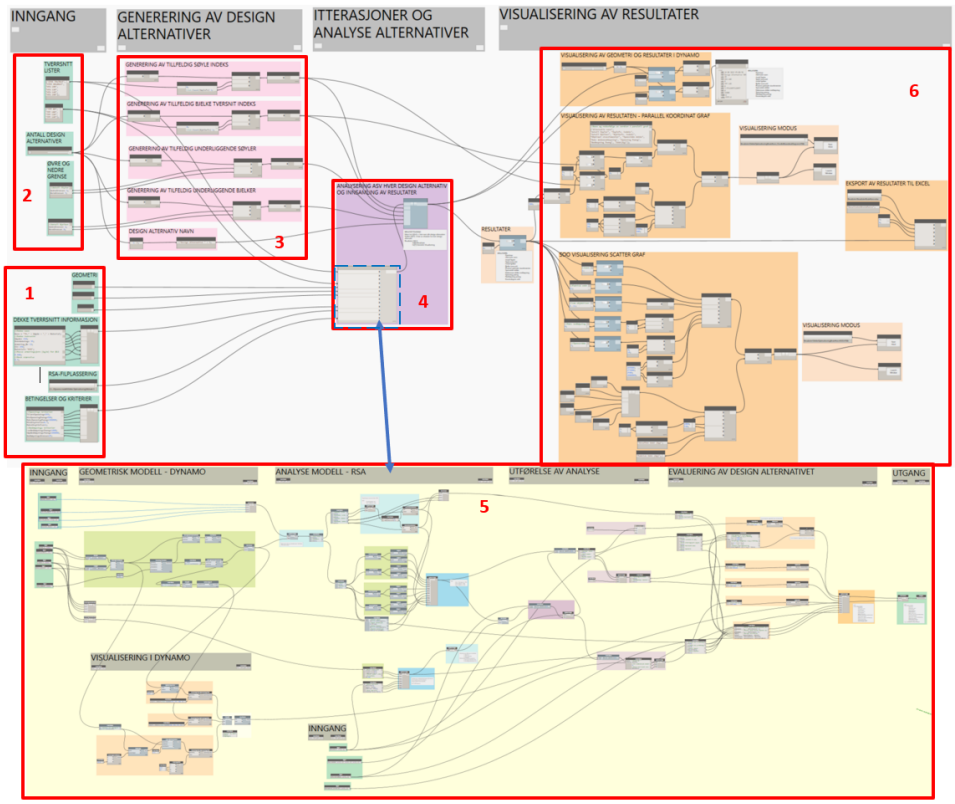
3.2.1 Oppbygging av Brute force search

Oppbygningen av denne optimaliseringsmetoden, som nevnt i Kapittel 2.2.3, benytter den parametriske modellen til å utvikle en analysefunksjon. Oppretting av analysefunksjonen gjennomføres ved å konvertere den parametriske modellen til en *custom node*, kalt *Dekke-OptimaliseringAnalyseFunksjon"*. Ved oppretting av den egendefinerte noden, er faste og varierende parametere trukket ut, slikt at endringer kan gjøres på en enkel måte. Outputen av denne noden er to lister. Én med resultater og én med visualisering av geometrien av alternativet i Dynamo. Analysefunksjonen og Dynamo-noden "*List.Combine*som benyttes for å gjennomføre iterasjoner fra gitte lister av varierende parametere, er illustrert i Figur 3.37.



Figur 3.37: Dekkeoptimaliseringsanalyse-funksjonen og Dynamo-noden List.Combine som benyttes for å gjennomføre iterasjoner

Når analyse funksjonen er opprettet, er alle faste parametere koblet til denne noden. For hver varierende parameter lages det en liste med tilfeldig genererte alternativer, basert på variablenes domene. Disse listene kobles videre til *List.Combine*-noden, som gjennomfører iterasjoner for alle variable verdier i disse listene. Etter at iterasjoner er ferdige, samles resultater fra alle tilfellene i utgangen av denne noden. Til slutt er resultatene sortert og videresendt til visualiseringsmetodene av resultatene. Oversikt over oppbygning av optimaliseringsmetoden Brute force search er illustrert i Figur 3.38, sammen med et punkt forklaring i Tabell 3.10.



Figur 3.38: Oversikt over oppbygning av optimaliseringsmetoden Brute force search

1 - Faste parametere
2 - Varierende parametere
- Tverrsnittindeks søyler
- Tverrsnittindeks bjelker
- Antall underliggende søyler
- Antall bjelker
- Antall alternativer

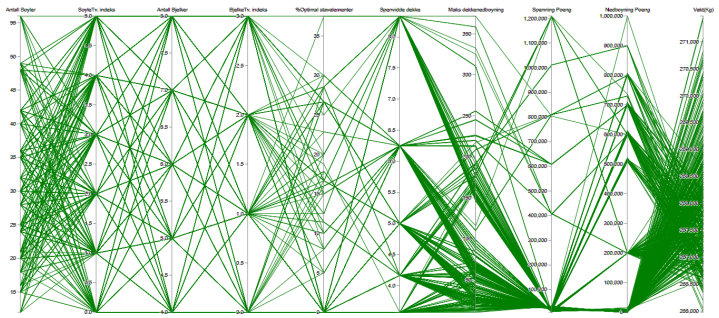
3 - Oppretting av tilfeldige genererte lister av variablene
4 - Gjennomføring av iterasjoner av analysefunksjonen fra variabellistene
5 - Analysefunksjonens innhold utviklet fra modellen i metode 1 - Design utforskning
6 - Visualisering av resultatene
- Visualisering av resultater og geometrien av ønsket tilfelle i Dynamo
- Eksport av alle resultatene til Excel
- MOO - Visualisering av alle tilfellene i en paralellkoordinat-graf
- SOO - Visualisering av alle tilfellene i en punktkoordinat-graf

Tabell 3.10: Stykkevis forklaring av oppbygning til optimaliseringsmetoden Brute force search

3.2.2 Resultater

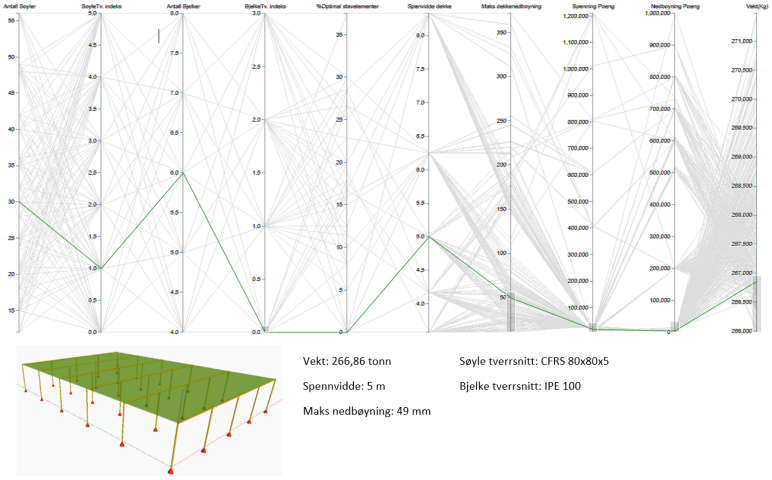
Resultatene som er vist gjennom dette kapitlet, er etter gjennomføring av 600 alternativer. Visualisering av alternativene illustreres først i en parallellkoordinat diagram, forså å vurdere to utvalgte løsninger som optimale løsninger. Videre er resultatene visualisert som en SOO-problem i en spredningsdiagram, ved å summere alle de objektive målene til et felles må kalt straffet vekt". Denne visualiseringen benyttes for å indikere den optimale løsningen basert på denne grafen. Visualiseringsmetodene vist gjennom dette kapitlet og resultatene fra alle alternativene i en Excel-fil, er vedlagt i den elektroniske innleveringsmappen.

Visualisering av alle alternativene i en parallellkoordinat-graf er illustrert i Figur 3.39. Denne visualiseringsmetoden gir muligheten til å gruppere resultatene, slikt at de ønskede resultatene lettere kan identifiseres. Gruppering av alternativene kan utføres for vedlagte HTML-filer.

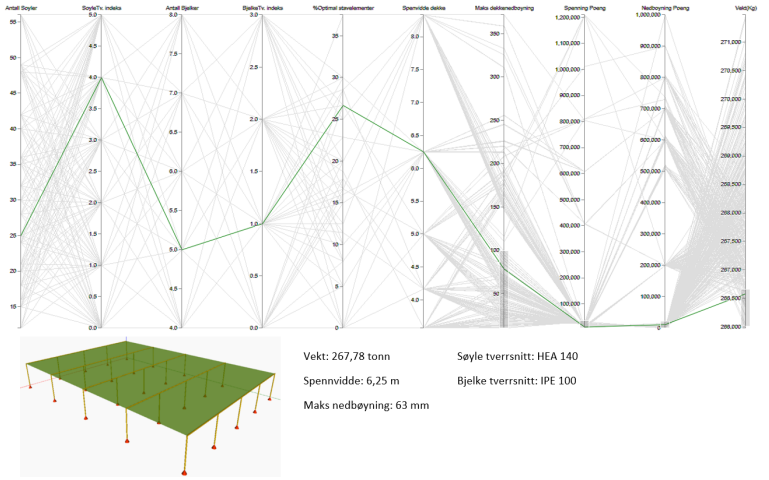


Figur 3.39: Visualisering av alle alternativene i en parallellkoordinat-graf

Figur 3.40 og Figur 3.41 illustrer to valgte alternativer ved gruppering av områder for spenningspoeng og nedbøyningspoeng, forså å finne alternativer med lavest vekt i disse områdene.

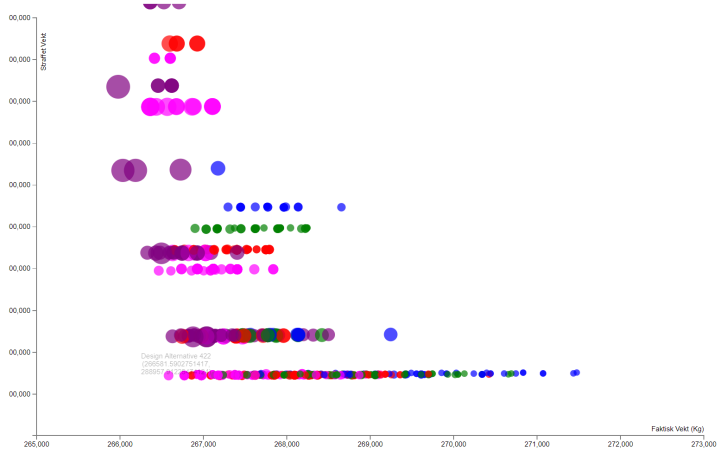


Figur 3.40: Visualisering av et utvalgt alternativ fra parallellkoordinat-graf

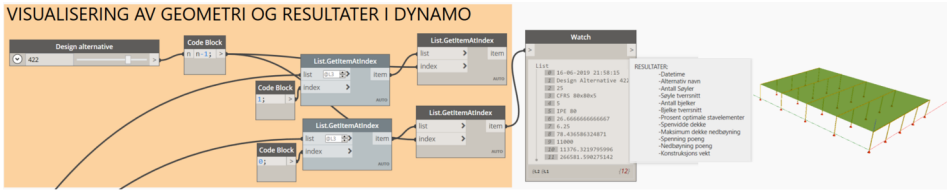


Figur 3.41: Visualisering av et utvalgt alternativ fra parallellkoordinat-graf

Resultatene fra alle objektive målene er summert og illustrert i en spredningsdiagram i Figur 3.42. Straffet vekt er i y-aksen mot den faktiske vekten i x-aksen. Størrelsen i punktene i grafen illustrerer dekkets nedbøyning, og fargene representerer forskjellige spennvidder. Den valgte optimale løsningen fra denne visualiseringsmetoden er alternativ 422, som er markert i grafen. Visualisering av resultatene og geometrien for alternativ 422 er illustrert i Figur 3.43, ved bruk av visualisering av alternativene i Dynamo.



Figur 3.42: Visualisering av resultatene som et SOO problem i en spredningsdiagram mot faktisk vekt



Figur 3.43: Visualisering av et utvalgt alternativ fra parallellkoordinat diagram

3.3 Metode3 Optimalisering med Optimo

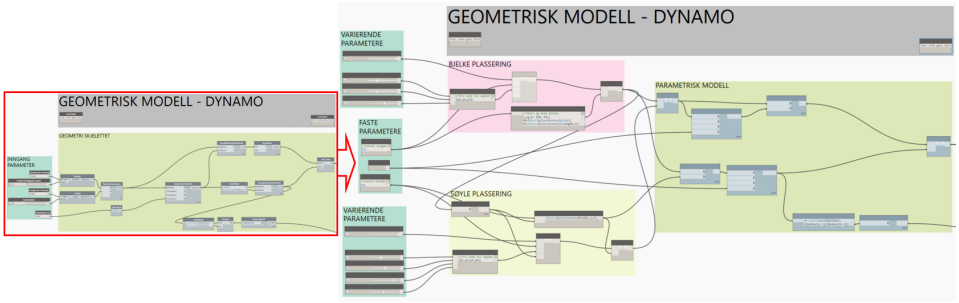
Dette kapitelet handler om oppbygning og gjennomføring av en GA-optimaliseringsprosess for et usymmetrisk parametrisk dekkesystem, illustrert i Figur 3.3. Optimaliseringsalgoritmen benytter Dynamo-pakken Optimo, som baseres på optimaliseringsteknikken NSGAII, nærmere beskrevet i Kapittel 2.2.2. Hensikten med å benytte GA-optimaliseringsprosess er å begrense antall iterasjoner for den usymmetriske modellen med uendelig mulige tilfeller, samt en automatisk beslutningsprosess av optimale løsninger.

Dette kapitlet starter med å bygge opp den usymmetriske parametriske modellen ved å endre det geometriske skjelettet fra modellen utviklet tidligere i kapittel 3.1.1. Da den parametriske modellen er utviklet, lages det en ny analysefunksjon på samme måte som vist i Kapittel 2.2.3 Brute force search. Deretter benyttes analysefunksjonen til å lage fitnessfunksjoner for tre objektive mål. Disse objektive målene kobles til Optimo-nodene, med den generelle oppsette for Optimo vist i Figur 2.6. Til slutt er domene for optimaliseringsprosessen valgt, basert på resultatene fra Brute force search. I denne oppgaven er den GA-optimaliseringsprosessen gjennomført for 15 generasjoner med en populasjon størrelse på 100 alternativer.

3.3.1 Usymmetrisk parametrisk modell

Den usymmetriske parametriske modellen er identisk som den tidligere utviklede modellen, med kun endringer av prosessen for det geometriske skjelettet. Det geometriske skjelettet starter på samme måte som tidligere med definering av søylenes fotavtrykk, før de resterende elementene defineres.

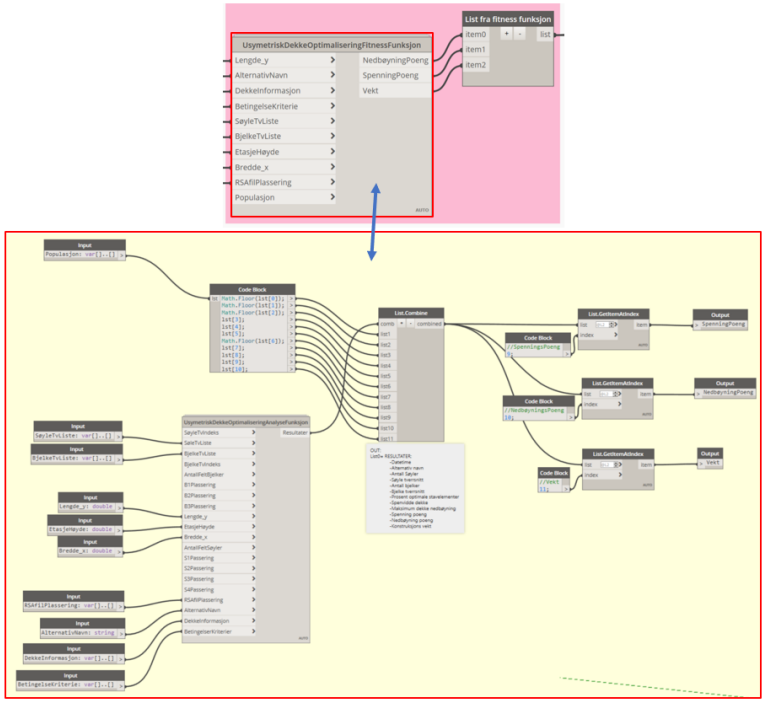
De variable parameterene i denne modellen er antall mellomliggende bjelker feltbjelker", og antall mellomliggende søyler feltsøyler"i underkant av bjelkene. Plassering av disse elementene har en friavstand på flatens lengde og bredde, henholdsvis for feltbjelker og feltsøyler. Endringer av det geometriske skjelettet er illustrert i figur 3.44.



Figur 3.44: Endring av geometrisk skjelett fra symmetrisk til usymmetrisk parametrisk modell

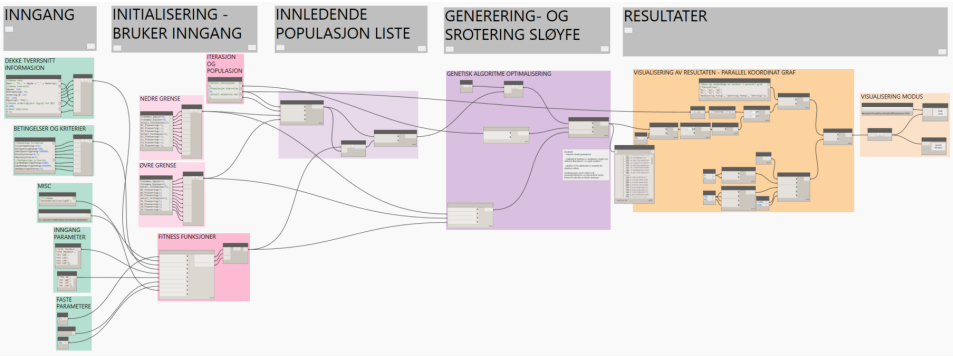
3.3.2 Oppbygning av GA-optimalisering med Optimo

Oppbygning av denne optimaliseringsmetoden benytter den usymmetriske parametriske modellen til å utvikle en analysefunksjon kalt for "*UsymmetriskDekkeOptimaliseringAnalyseFunksjon*". Da analysefunksjonen er opprettet, benyttes denne noden til å lage fitnessfunksjoner, spenningspoeng, nedbøyningspoeng og vekt. Oversikt over oppbygning av fitnessfunksjoner er illustrert i Figur 3.45.



Figur 3.45: Oversikt over oppretting av fitnessfunksjoner ved bruk av analysefunksjon

Til slutt er fitnessfunksjoner og domene for alle varierende parametere koblet til det generelle oppsettet for Optimo. I denne oppgaven er nedre og øvre grense for feltbjelkene henholdsvis 2 og 3, og for feltsøylene henholdsvis 2 og 4. Oversikt over oppsett av GA-optimalisering med Optimo er illustrert i Figur 3.46.



Figur 3.46: Oversikt over oppsett av GA-optimalisering med Optimo

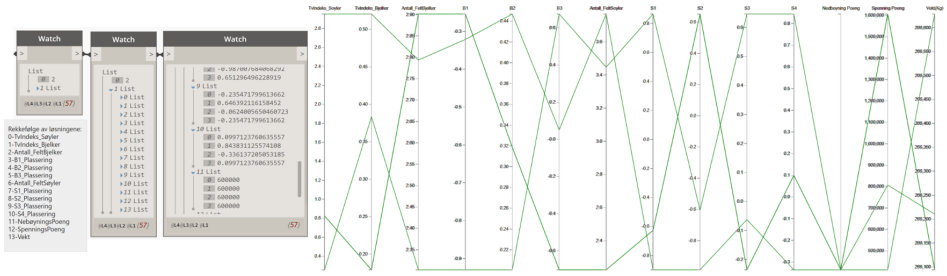
3.3.3 Resultater

Gjennom dette delkapitlet er resultatene fra GA-optimaliseringsprosessen for den usymmetriske modellen vist. Testen er gjennomført for 15 generasjoner med en populasjonsstørrelse på 100 individer. Dette utgjør totalt 1500 iterasjoner av analysefunksjonen.

Den totale gjennomføringstiden for denne optimaliseringsprosessen var 15 timer og 28 minutter. Gjennomføringstid av den første iterasjonen er 19 sekunder og for den siste iterasjonen 56 sekunder.

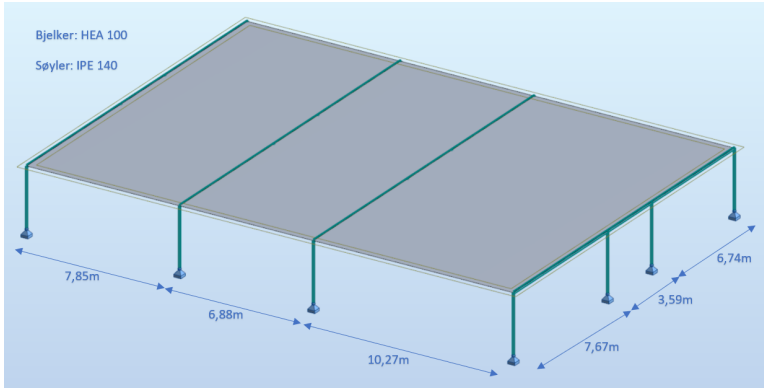
Resultatene fra GA-optimalisering med Optimo visualiseres i to lister. Den første listen representerer generasjonen av resultater som inneholdes i den andre listen. Resultatene i den andre listen representerer de Pareto-optimale løsningene for alle variablene og de objektive målene. Hver av variablene og de objektive målene inneholder en liste med ikke-dominerte resultater likt populasjonsstørrelsen. Variablenes resultater representerer DNA for hvert individ, og de objektive målene viser til hvor godt hvert av individene er representert i den gjeldende generasjonen. GA og Pareto-fronten er nærmere beskrevet i Kapittel 2.2.1.

I denne oppgaven er resultatene fra Optimo visualisert i en parallellkoordinat-graf. Grafen er utarbeidet ved å bruke Dynamo-pakken Mandrill. Figur 3.47 illustrerer resultatene fra Optimo for den usymmetriske modellen, med populasjonsstørrelse lik fire etter to generasjoner.



Figur 3.47: Resultater fra Optimo og visualisering i en parallellkoordinat-graf, for den usymmetriske modellen med populasjonsstørrelse lik fire etter to generasjoner

Resultatene fra testen med 1500 iterasjoner er ved en uforventede feil ikke visualisert fra Optimo nodene. Da alle Pareto optimale løsningene er blant de 100 siste alternativene, har RSA-filene blitt benyttet til å identifisere den optimale løsningen. Den optimale løsningen som er identifisert fra denne optimaliseringsprosessen er illustrert i Figur 3.48.



Figur 3.48: Den indikerte optimale løsningen for den usymmetriske modellen med populasjonsstørrelse lik 100 etter 15 generasjoner

Kapittel 4

Diskusjon

Arbeidsflyten ved bruk av VP i et tidlig prosjektstadium, gir arkitekter og ingeniører muligheten til å utforske flere designalternativer i et parametrisk miljø, med begrenset ressursbruk for å utarbeide optimale løsninger basert på prosjektets mål. Utvikling og bruk av VP for arkitekter og ingeniører i byggebransjen er kontinuerlig under kraftig utvikling. I dag gir arbeidsflyten mellom VP og Dynamo et større handlingsrom for oppbygning av parametriske BIM-modeller for designere og arkitekter, da Dynamo kommer som en integrert del av Revit. Oppbygning av parametriske BIM-modeller med Dynamo sammen med Revit kan i større grad gjennomføres uten behov for tradisjonell programmering, kontra kobling mellom Dynamo og RSA. Det større handlingsrommet inviterer flere brukere uten programmeringsbakgrunn til bruk av VP. Gjennomføring av konstruksjonsmessige analyser i et parametrisk miljø ved bruk av Dynamo og RSA benytter Dynamo-pakken Structural Analysis. Denne pakken gir brukere uten bakgrunn i programmering muligheten til parametrisk styring av RSA. Den enkle metoden å kunne styre RSA gjennom Dynamo inviterer byggingeniører til gjennomføring av algoritmiske optimaliseringsmetoder av konstruksjoner. Structural Analysis-pakken er hovedsakelig begrenset til modeller med kun stavelementer. Dette vil i en reell situasjon begrense byggingeniørenes hverdagslige bruk av Dynamo og RSA. Gjennom denne oppgaven er disse begrensningene overkommet ved bruk av RSA API. Dette gjenspeiler en mindre praktisk bruk av Dynamo og RSA, der det kun er begrenset antall brukere med nødvendige bakgrunn i programmering. Det må samtidig nevnes at, ved bruk og kjennskap til VP, er veien til tekstbasert programmering betraktelig kortere.

Denne oppgaven har gjennom utvikling av den parametriske modellen av dekkesystemet, lykkes med å utvikle modeller via Dynamo som inkluderer FE sammen med nødvendige FEM-analyseforutsetninger. Forutsetningene i denne oppgaven har vært påføring av flatelaster, definering av lastkombinasjoner og parametrisk definering av dekketverrsnitt. Videre har oppgaven lykkes med å bestemme kalkuleringsparametere for FEM-beregningen, og utførelse av kalkuleringen for så å hente nødvendige resultater til Dynamo.

Nøyaktigheten av resultatene for den parametriske modellen er forklart gjennom verifisering av resultater i Kapittel 3.1.3, der forenklingene som er valgt kan i en reel situasjon ha større effekter for indikering av optimale løsninger. Dette kan henvises hovedsakelig til to tilfeller i denne modellen, der den ene er definering av lastkombinasjoner med nyttelast over hele dekket. For et mer nøyaktig resultat kan en benytte automatisk generering av lastkombinasjoner via RSA API. Den andre forenklingen er knyttet til beregning av den faktiske nedbøyningen, med en antagelse om totalt dekketverrsnitt i Stadium II. For at resultatene fra nedbøyningene skal samsvare med den beregnede nedbøyningen i RSA, kan en benytte RSA API til beregning av armering i dekket, for deretter å hente nedbøyningsresultater.

Rangering og evaluering av alternativene basert på resultatene fra RSA, er gjennomført ved å poengtere utnyttelsesgraden av konstruksjonens elementer. Denne metoden har fungert godt for å indikere alternativene som ikke er ønskelige, for deretter å finne alternativer med lavest materialbruk. Ettersom det er benyttet forskjellige materialer i denne modellen, har vektlegging av poengteringen ikke fungert optimalt. En kan tenke seg at en kilo med betong kontra stål bør ha forskjellige vektlegging og poengtering. I en reel situasjon er to objektive mål for poengfordelingssystemene av alternativene optimalt. Den ene er et objektivmål som baseres på materialenes kostnad, og den andre objektive målet baseres på materialenes CO2-utslipp. Hver av disse poengfordelingssystemene kan benyttes som objektive mål sammen med den totale konstruksjonsvekten i optimaliseringsanalyser. På denne måten er det enklere å identifisere effekten av forskjellige alternativer basert på miljø og økonomi.

Resultatene fra optimalisering av den symmetriske parametriske modellen med Brute force search har lykkes med å identifisere de optimale løsningene. I denne oppgaven har hensikten med denne optimaliseringen vært å avdekke hvilke løsninger som reduserer behovet for lokale armeringsforsterkninger. Fra resultatene som er illustrert i kapittelet 3.2, er løsningen vist i Figur 3.43 valgt som den optimale løsningen. En av fordelen med å lagre alternativenes RSA-filer kommer spesielt til nytte, da en skal gjennomføre en mer detaljert analyse av de identifisere løsningene. RSA-filen for det optimale alternativet leveres i det elektroniske vedlegget. Ved å verifisere dette alternativet i RSA, er det observert et minimum lokalt forsterkningsbehov. Med dette kan det sies at den algoritmiske optimaliseringen med Brute force search for den symmetriske parametriske modellen har lykkes.

GA-optimaliseringsprosessen for den usymmetriske modellen har et betraktelig større variabelt domene, med 11 variabler. Gjennomføring av Brute force search i den tidligere optimaliseringen har vært til nytte for å kunne danne et bilde over løsningsrommet der optimaliseringen søkes. Resultatet fra denne analysen har ikke klart å konvergere mot den optimale løsningen. Resultatet av den optimale løsningen vist i Figur 3.48 er samtidig nærme til den tiltenkte optimale løsningen. Denne testen er gjennomført med kjennskap til hvordan utformingen det optimale alternativet vil være. Da dekket er definert kontinuerlig over alle bjelkene, vil den optimale løsningen befinne seg et sted slikt at nedbøyning i alle feltene er like. Den tiltenkte årsaken til at resultatene ikke er konvergert kan være et lavt populasjonstall ved gjennomføring av analysen. Ved å øke populasjonen i en GA-optimaliseringsprosess vil en raskere kunne nærme seg de optimale løsningene. I noen tilfeller med lav populasjonsstørrelse, kan den optimale løsningen ikke bli oppdaget av

analysen. En annen grunn til at resultatene ikke har konvergert, være at det finnes flere løsninger for denne testen, ettersom det finnes tre objektive mål i analysen. I slike tilfeller kan en omforme alle objektive mål til et SOO-problem. Omforming av MOO til SOO for denne oppgaven er vist i kapittelet 3.1.4.

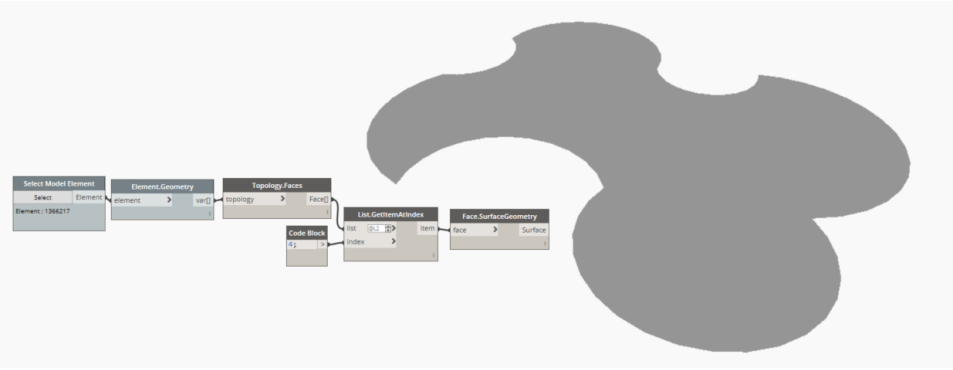
Resultatene fra analysene har i stor grad klart å oppnå alle målene med å utvikle en parametrisk modell som inkluderer FE i algoritmiske optimaliseringsprosess. Hvor godt denne modellen kan gjenbrukes for andre tilfeller, vil være avhengig av brukerens bakgrunn og kjennskap til Dynamo og programmering i Python. Da all kommunikasjon med RSA er gjennomført via RSA API ved bruk av Python koder, kan dette virke omfattende for ikke-programmerere. For å danne robuste løsninger som er mer brukervennlige, bør kodene opprettes som ZeroTouch-noder i programmeringsspråket C#. Ved å benytte C# for kommunikasjon med RSA, vil det samtidig gi en ekstra redusering av gjennomføringstiden.

Den praktiske bruken av parametrisk design for hverdagslige arbeidsoppgaver for bygningsingeniører, kan reflekteres med ressursbruken av å utvikle en parametrisk modell, kontra nyttigheten av disse modellene. Denne rapporten har i stor grad vist den praktiske nyttigheten for utvikling av parametrisk modeller. Ressursbruk for gjennomføring av en optimaliseringsprosess i et parametrisk miljø kan refereres til oppbygnings- og gjennomføringstiden av parametriske modeller.

Gjennomføringstiden av denne modellen har hatt en kraftig reduksjon fra tre minutter til underkant av 30 sekunder. Ved omforming av modellen til en egendefinerte node, er denne tiden redusert til 20 sekunder. Dette vil i teorien utgjøre ca. åtte timers gjennomføringstid for 1500 iterasjoner. Det har blitt observert en økning av minnebruk for RSA-filer med 37 Kb for hver iterasjon. Dette er grunnen til at den totale gjennomføringstiden for GA-optimaliseringsprosessen har brukt 15 timer på gjennomføring av 1500 modeller. Dette kan sannsynligvis unngås ved å benytte RSA API til å åpne nytt RSA-dokument i starten av hver iterasjon.

Oppbygningstiden av parametriske modeller er som oftest avhengig av gjenbruk av gamle modeller, og kompleksiteten til prosjektet. Når den parametriske modellen er laget vil det som regel være nyttig å bruke alle optimaliseringsmetodene vist gjennom denne oppgaven til å søke etter optimale løsninger. Dette vil som regel være en tidkrevende prosess, da det hver gang må lages nye fitnessfunksjoner, og brukes tid på visualisering av modellene. En god nyhet er Autodesk sitt nye prosjekt kalt Refinery, som benytter en parametrisk modell til å gjennomføre algoritmiske optimaliseringsmetoder. Refinery benytter de samme optimaliseringsteknikkene som er vist gjennom denne oppgaven, med GA-optimalisering basert på NSGA-II. Fordelen med Refinery er en optimal visualiseringsmetode av alternativene, og en rask bruk av en parametrisk modell til algoritmiske optimaliseringsprosesser. I dag er Refinery utgitt som Beta-versjon, der det ikke er mulig å benytte GA-optimalisering for modeller som kommuniserer med andre programmer. Det er laget et videoopptak for bruk av Refinery for optimalisering av den symmetriske modellen med Brute force search. Videoen er vedlagt i det elektroniske vedlegget.

Den potensielle bruken av denne automatiseringsprosessen kan henvises til et eksempel med uregelmessige dekkegeometrier. I slike tilfeller vil denne automatiseringsprosessen komme spesielt til syne, da en kan identifisere forskjellige optimale løsninger i en tidligprosjekteringsfase fra en skisse tegning. Dekket fra skissetegningen kan enten utvikles direkte i Dynamo, slikt som er gjort gjennom denne oppgaver, eller importeres fra et BIM modell gitt av arkitekten. illustrerer et eksempel av en uregelmessig dekkegeometri som er importert fra Revit. Da denne geometrien er hentet til Dynamo, kan det parametriske geometriske skjelettet lages, for deretter å koble denne geometrien til resterende deler av grafen. Endring av det geometriske skjelettet er vist gjennom utvikling av den usymmetriske modellen. Når den parametriske modellen er utviklet, kan brukeren undersøke dekket med algoritmiske optimaliseringsmetoder. Ved bruk av Refinery vil denne prosessen gjennomføres på en enkel og optimal måte.



Figur 4.1: Eksempel av en uregelmessig dekkegeometri som er importert fra Revit til Dynamo

Kapittel 5

Konklusjon

Denne oppgaven har siktet på å undersøke mulighetene til å automatisere konstruksjonsanalyse av dekkesystemer i et parametrisk miljø, og benytte algoritmiske optimaliseringsmetoder til å indikere flere optimale løsninger for prosjekter i en tidlig prosjekteringsfase. Gjennom oppgaven er det utviklet en beregningsmetode i det visuelle programmeringsverktøyet Dynamo, som benytter analyseverktøyet Robot Structural Analysis, gjennom en kombinasjon av visuell programmering i Dynamo og tekstbasert programmering i Python. Denne kombinasjonen muliggjør parametrisk styring av analyseverktøyer, og dermed utvikling av parametriske analysemodeller. Beregningsmetoden i Dynamo er benyttet til utvikling av to forskjellige parametriske modeller for dekkesystemer, der disse modellene har blitt testet gjennom algoritmiske optimaliseringsmetoder for identifisering av optimale løsninger. Optimalisering av modellene er testet gjennom to algoritmiske optimaliseringsmetoder: Brute force search og Genetisk algoritmisk optimalisering. Metodene som presenteres kan hjelpe bedrifter gjennom automatisk generering av flere designmuligheter, og forbi konkurransedyktige konkurrenter i en digitalisert industri. Denne ideen støtter tankegangen bak Generativ design.

Hensikten med denne oppgaven har vært å vise til praktisk bruk av parametriske modeller for byggingeniører, ved å automatisere et av hverdagslige oppgavene for en bygningsingeniør. Behovet for identifisering av optimale dekkesystemer i en tidlig prosjekteringsfase med lav ressursbruk er en av de praktiske eksemplene, utarbeidet i samarbeid med Sweco.

Gjennom oppgaven er det tatt utgangspunkt i et gitt dekkeareal for et kontorbygg, der optimalisering av to type oppbygning av dekkesystemer er undersøkt. Den ene parametriske modellen er utviklet for et dekkesystem med symmetrisk aksekrysning. Da antall mulige alternativer for denne modellen har vært begrenset, er optimaliseringen gjennomført ved bruk av Brute force search. Den andre parametriske modellen er utviklet for et dekkesystem med en usymmetrisk aksekrysning. Ettersom denne metoden har et tilnærmet uendelig utfallsrom, er optimaliseringen gjennomført ved bruk av Dynamo-pakken Optimo, som benytter den genetiske optimaliseringsteknikken NSGA-II til å finne de ikke-dominerte al-

ternativene. Resultatene fra analysene er visualisert i Parallelkoordinat-grafer, som er en nyttig metode for evaluering av de optimale løsningene. Resultatene fra analysene har i stor grad klart å indikere de optimale løsningene, med en kvalitet og nøyaktighet beregnet til et tidlig prosjekteringsstadie.

Arbeidet som er presentert gjennom denne oppgaven gir en beregningsmetode for automatisk optimalisering av dekkesystemer. Det må presiseres at metodene imidlertid ikke erstatter ingeniøren, men hjelper til i beslutningsprosessen ved å indikere de optimale løsningene.

5.1 Fremtidig arbeid

Dette kapitlet beskriver ideer for fremtidig arbeid av beregningsmodellen for dykkesystemet, utviklet gjennom denne oppgaven.

Inkludering av detaljerte forbindelser
Inkludering av et detaljerte elementforbindelser av konstruksjonen i den parametriske modellen, fører til en mer detaljert analyse og sparer oppbygnings- og kalkuleringstid ved kontroll av modellene. Parametrisk definering av forbindelser gir muligheten til å kunne sammenligne den totale effekten for valg av forskjellige typer forbindelser, med begrenset ressursbruk.

Inkludering av dimensjonering for armering
Inkludering av beregnet nødvendig armering i dekket vil ha den største effekten på detaljnivået til analysens resultater. Dette kan gjennomføres ved bruk av RSA API og beregningsalgoritmen for armering i RSA.

Vektlegging og poengtering av evalueringsprosess
Utarbeide evalueringsmetoder som kan vektlegge konstruksjoner bygget opp med forskjellige materialer, basert på miljø og økonomi. På denne måten vil det være enklere å sammenligne effekten av resultatene fra analysen.

Automatisk importering av uregelmessige dekkegeometrier
Automatisk importering av uregelmessige dekkegeometrier fra Revit til Dynamo, og automatisk definering av geometrisk oppbygning av dekkesystemet i Dynamo. Ved bruk av denne metoden kan denne beregningsmodellen benyttes i et tidlig prosjekteringsstadie til avdekking av alle typer dekkegeometrier med minimal ressursbruk.

Reprodusering av resultatene som en tilleggspakke i Dynamo
Ved å omskrive den utviklede beregningsmodellen ved bruk av C#, kan prosessene utvikles som ZeroTouch-noder, som er en mer brukervennlig løsning enn Python. Disse prosessene kan benyttes til å lage en tilleggspakke i Dynamo for optimalisering av dekkesystemer. Denne pakken i en kombinasjon med prosjektet Refinery, kan gjennomføre raske optimaliseringsanalyser ved bruk av algoritmiske optimaliseringsmetoder.

Bibliografi

Autodesk, 2018a. The dynamo primer, for dynamo v2.0. https://primer.dynamobim.org/en/index.html, accessed: 2019-04-09.

Autodesk, 2018b. A new way to get dynamo sandbox. https://dynamobim.org/a-new-way-to-get-dynamo-sandbox/, accessed: 2019-04-09.

AUTODESK, H., 2017a. Multithreaded solver - autodesk knowledge network. https://knowledge.autodesk.com/support/robot-structural-analysis-products/learn-explore/caas/CloudHelp/cloudhelp/2018/ENU/RSAPRO-UsersGuide/files/GUID-EA136410-79B7-4679-AEA9-AC949BA061AB-htm.html, accessed: 2019-05-25.

AUTODESK, h., 2019. Plate and shell deflections - calculations. https://knowledge.autodesk.com/support/robot-structural-analysis-products/troubleshooting/caas/CloudHelp/cloudhelp/2020/ENU/RSAPRO-UsersGuide/files/GUID-5825DF36-A541-4D2D-B521-A53ABB80760E-htm.html, accessed: 2019-05-25.

AUTODESK, S., 2017b. Robot - what types of finite elements are used in the program? https://knowledge.autodesk.com/support/robot-structural-analysis-products/troubleshooting/caas/sfdcarticles/sfdcarticles/ROBOT-what-types-of-finite-elements-are-used-in-the-program.html, accessed: 2019-05-25.

Biilmann, T., 2015. Det er nu vi bygger fremtiden. Libris Business.

Cantero, D., 2016. Formula list (concrete1). INSTITUTT FOR KONSTRUKSJONSTEKNIKK NTNU, 13.

Cavieres, A., Gentry, R., Al-Haddad, T., 2011. Knowledge-based parametric tools for con-

crete masonry walls: Conceptual design and preliminary structural analysis. Automation in Construction 20 (6), 716–728.

Chukwuchekwa, N., 2011. Parametric optimisation using genetic algorithm. INTERNATIONAL JOURNAL Of ACADEMIC RESEARCH 3 (2), 325–335.

Curry, D., Dagli, C., 2014. Computational complexity measures for many-objective optimization problems. Procedia Computer Science 36, 185–191.

Deb, K., 2002. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. Springer, Berlin, Heidelberg 6 (2), 182–197.

Deb, K., 2014. Multi-objective Optimization. Springer, Boston, MA.

Eiben, A., Ruttkay, Z., Raue, Z., 1994. Genetic algorithms with multi-parent recombination. Davidor Y., Schwefel HP., Männer R. (eds) Parallel Problem Solving from Nature 866, 78–87.

Fisher, A., Shorma, S., 2010. Exploiting autodesk robot structural analysis professional api for structural optimization. SMART Solutions, Buro Happold Ltd, 1–10.

Giagkiozis, I., Fleming, P., 2015. Methods for multi-objective optimization: An analysis. The University of Sheffield, 1–17.

Katili, I., 1993. A new discrete kirchhoff-mindlin element based on mindlin–reissner plate theory and assumed shear strain fields—part ii: An extended dkq element for thick-plate bending analysis. John Wiley and Sons, Ltd 36 (11), 1885–1908.

Kenbur, R., 2005. Pareto's revenge. The University of Sheffield, 1–12.

Kensek, K., Khan, W., 2013. Integration of environmental sensors with bim, 7 seven case studies. School of Architecture, University of Southern California, 1–7.

Przemieniecki, J., 1985. Theory of matrix structural analysis. INTERNATIONAL JOURNAL Of ACADEMIC RESEARCH, 468.

Rahmani, M., Bergin, M., et, a., 2014. Bim-based parametric building energy performance multi objective optimization. 32nd eCAADe Conference 224, 1–10.

Rahmani, M., Stoupine, A., Zarrinmehr, S., 2015. Optimo: A bim-based multi-objective optimization tool utilizing visual programming for high performance building design. Conference of Education and Research in Computer Aided Architectural Design in Europe 1, 673–382.

Rahmani, M.and Zarrinmehr, S., 2015. Bpopt: A framework for bim-based performance optimization. Energy and Buildings, 108, 401–412.

Ruiz, F., Miettinen, K., Wierzbicki, A., 2008. Introduction to multiobjective optimization: Interactive approaches. Springer-Verlag Berlin Heidelberg, 1–32.

Stemland, S.and Rodum, E., Johansen.H, 2016. Alkalireaksjoner – veiledning for konstruktiv analyse - etatsprogrammet varige konstruksjoner 2012-2015. Statens vegvesens rapporter Nr.601, 33.

Talerico, G., 2017. Untangling python: A crash course on dynamo's python node. Autodesk University, 1–30.

Tammik, J., 2016. All you ever wanted to ask about the revit api , the building coder , and other non-ui revit topics. Autodesk, 1–6.

Tierney, P., 2015. Efficiently working with large data sets in dynamo. https://github.com/DynamoDS/Dynamo/wiki/Efficiently-Working-With-Large-Data-Sets-In-Dynamo, accessed: 2019-04-16.

Trakhtenbrot, B., 1984. A survey of russian approaches to perebor (brute-force search) algorithms. Annals of the History of Computing 6 (4), 384–400.

Varmeulen, D., 2017. Construction dynam(o)ite—explode productivity with dynamo. Autodesk University, 1–130.

Vermeulen, D., 2018. Structural dynam(o)ite: Optimized design and fabrication workflows with dynamo. Autodesk University, 1–56.

Yusoff, Y., Ngadiman, M., Zain, A., 2011. Overview of nsga-ii for optimizing machining process parameters. INTERNATIONAL JOURNAL Of ACADEMIC RESEARCH 15 (2), 3978–3983.

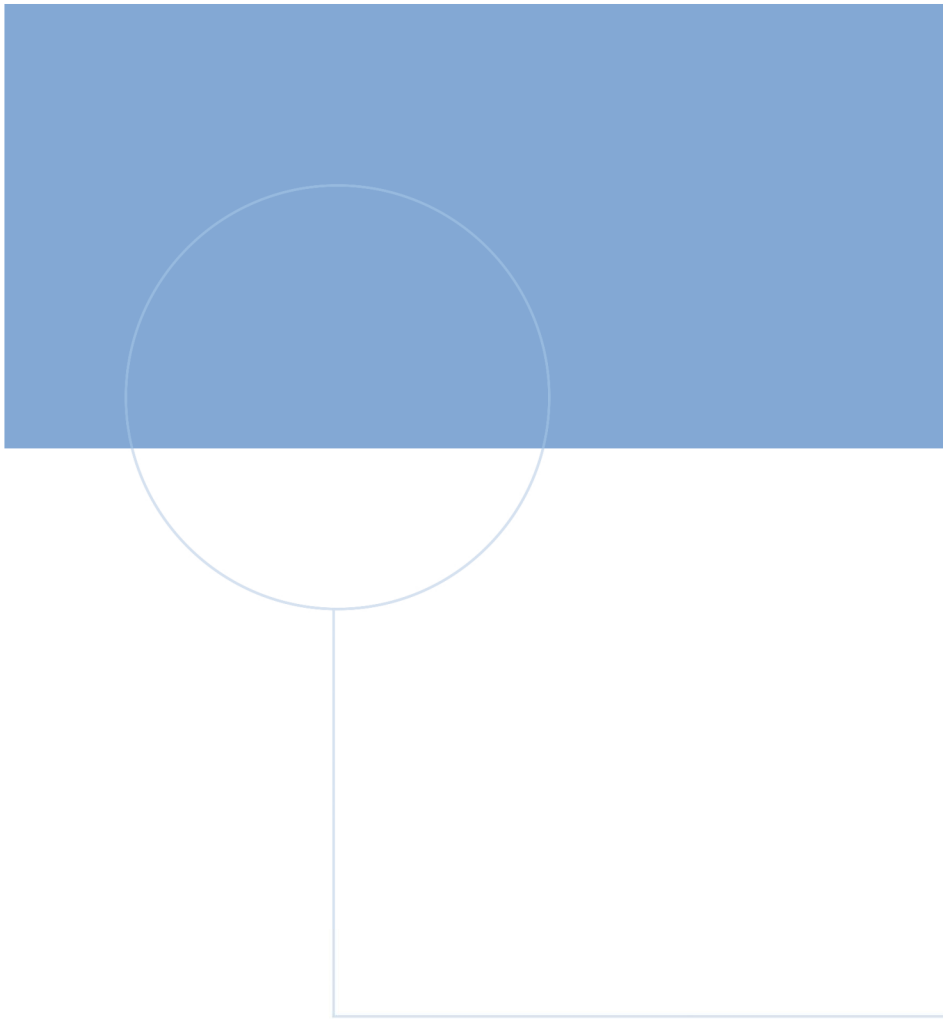
Vedlegg

Vedlegg A

Dette vedlegget viser oppbygging og innhold av det elektroneske vedlegget.

Hovedfiler	Underfiler	Beskrivelse	Filer
1. Dynamo modeller			
	Metode 1 Design utforskning	Dynamo grafer (.dyn)	2
		Visualisering (.HTML)	2
	Metode 2 Brute force search	Dynamo grafer (.dyn og .dyf)	2
		Visualisering (.HTML)	2
		Resultater (.xlsl og .rtd)	2
	Metode 3 GA-optimalisering	Dynamo grafer (.dyn og .dyf)	4
		Visualisering (.HTML)	1
		Resultater (.rtd)	1
2. Python koder			
		Python koder (.txt)	10
3. Refinery			
		Video (.mp4)	1
		Dynam graf (.dyn)	1

Tabell 5.1: Oversikt over digital vedlegg



NTNU
Kunnskap for en bedre verden

SWECO