Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

MASTEROPPGAVE

Studieprogram/spesialisering: Femårig Master i Teknologi (siv.ing.), Spesialisering Byutvikling og urban design	Vårsemesteret, 2018 Åpen
Forfatter: Martin Eriksen Dilkestad	Martin Eriksen Dilkestad (signatur forfatter)
Fagansvarlig: Rolv Arnstein Øvrelid Ekstern Veileder: Gunnar Skeie, Kruse Smith	
Tittel på masteroppgaven: BIM - Parametrisk Design som beslutningsstøtte (Dagslysberegninger) Engelsk tittel: BIM - Parametric Design as a Decision-making Tool (Daylight Calculations)	
Studiepoeng: 30	
Emneord: VDC BIM Parametrisk Design Computational Design Dagslysberegninger	Sidetall: 70 +vedlegg/annet: 75 Stavanger, 14.06.2018 dato/år

Forord

Denne oppgaven, skrevet våren 2018, er den avsluttende oppgaven for min femårige mastergrad i teknologi, med spesialisering innen byutvikling og urban design, ved Universitetet i Stavanger. Den ble skrevet i samarbeid med Universitetet i Stavanger og Kruse Smith, avdeling Stavanger.

Jeg ble tidlig i studiet mitt introdusert for diverse modellerings- og BIM verktøy, hvor jeg og fikk innsikt i potensialene disse holdt for fremtiden. Hvert år blir bygge-bransjen mer digitalisert ved hjelp av nye verktøy og teknikker, hvor mange av disse har blitt implementert, nærmest som standarder for å heve prosjektenes standard. Med disse nye kommer også nye muligheter, hvor disse verktøyene og teknikkene kan videreutvikles for å effektivisere prosjekts og produksjonsfasene ytterligere. Dette er noen av grunnene til at jeg har valgt dette temaet for masteroppgaven min, hvor jeg selv ønsker å kunne være med å utforske disse mulighetene.

Denne oppgaven ga meg mulighet til å både videreutvikle og bygge ny kompetanse innenfor diverse digitale verktøy, og dette anser jeg som veldig viktig for både nyutdannede og eksisterende arbeidstakere innen bygge-bransjen. I tillegg til denne kompetansen ble jeg også introdusert for nye teorier, samt anvendelser av de diverse verktøyene. Gjennom dette fikk jeg et godt innblikk for hvordan disse kan brukes i fremtiden, og effektiviseringspotensialet de inneholder.

Jeg vil gjerne takke Rolv Arnstein Øvrelid og Knut Erik Bang fra Universitetet i Stavanger, for veiledning de har gitt meg under denne oppgaven. Jeg vil også takke Gunnar Skeie og Ragnar Øksendal, samt resten av Kruse Smith, for deres deltakelse og innspill for oppgaven. Hjelpen jeg har mottatt denne våren har hjulpet meg til å se nye muligheter innenfor VDC, BIM og deres digitale verktøy, i tillegg til hvordan disse kan bli brukt i fremtiden. Jeg er veldig takknemlig for dette.

Sammendrag

Med digitale verktøy som blir mer fremtredende for hvert passerende år, sammen med de teknologiske fremskrittene som leverer mer høyt-ytende maskiner, blir disse verktøyenes rolle viktigere innenfor byggesektoren. Virtual Design Construction (VDC) og Building Information Modeling (BIM) er i en stadig evolusjonerende tilstand, med nye bruksområder som brukes til å effektivisere prosjekterings- og produksjonsfaser. Parametrisk design har eksistert i en del år, men har fortsatt ikke blitt et husholdningsnavn innenfor bransjens normer. Likevel er mulighetene der. Sammen med Parametrisk Design, har en annen applikasjon av BIM dukket opp; Computational Design. Disse to kombinert utgjør en potensiell banebrytende måte å designe prosjekter på; hvor man vil ha muligheten til å simulere alle design på forhånd, identifisere de beste, i tillegg til å få en presentasjon av disse (som modeller). På denne måten kan entreprenørene forsikre seg om at det endelige designet er det beste av sine evner, samtidig som tids- og ressursbruk reduseres for å utvikle disse designene.

Målet med denne oppgaven er å finne en slik metode. Hvor det parametriske designet av en bygning kan brukes til å finne de mest optimale designløsninger, og dermed bidra til beslutningstaking av hvilket design som skal brukes. Parametrisk design vil trenge mål å strekke seg etter når slike simuleringer skal utføres, og i denne oppgaven vil dette målet omhandle *dagslysberegninger*. Med en god metode for å anvende parametrisk design som beslutningsstøtte skal flere slike mål kunne brukes samtidig, hvor hele designet vil ta for seg alle forskrifter når den simulerer seg gjennom designene. For å finne denne metoden blir Autodesk Revit sammen med Dynamo brukt.

De valgte verktøyene (Revit og Dynamo) gir delvis positive resultater, men lever de fremdeles ikke opp til forventningene. Bruken av NSGA II-funksjonen er antatt å levere gode resultater i henhold til problemstillingen, men verktøyene klarer ikke å gjengi denne til sitt fulle potensiale. Hovedproblemet med verktøyene er tid og minneforbruk (RAM), noe som gjør dem ugunstige for større prosjekter.

Abstract

With digital tools becoming more prominent for each passing year, along with the technological advancements providing more capable hardware, are the role of these tools becoming more essential in the construction sector. Virtual Design Construction (VDC) and Building Information Modelling (BIM) are in an ever-evolving state, with new uses providing more efficient projecting and construction phases. Parametric Design has been around for a couple of years, but still to become a household name within the norms of the sector. Still, the possibilities are there. Along with Parametric Design, another application of BIM has emerged; Computational Design. These two combined poses a potential groundbreaking way of designing projects; where the possibility to simulate *all* designs beforehand, identifying the best ones, along with a presentation of these (as models). This way the entrepreneurs can be assured that the design is to the best of its abilities, as well as reducing the time and resources needed to develop these.

The goal of this thesis will be to create such a method. Where the parametric design of a building can be used to find the most optimal design solutions, thereby helping with the decision-making of which design to use. Parametric design needs objectives for these simulations, and in this thesis the objective will be about *daylight calculations*. With a good method of applying parametric design as a decision-making tool, multiple such objectives will be used at the same time, where the entire design will be able to follow regulations as it simulates through the designs. For these simulations, Autodesk Revit along with Dynamo was used.

Although the chosen tools (Revit and Dynamo) delivers the results to some degree, they still do not live up to expectations. The use of the NSGA II function is excepted to deliver preferable results, but the tools fails to make use of this to its full potential. The main issue with the tools are the time- and memory (RAM) consumption, which makes them unfavorable with larger projects in mind.

Innholdsfortegnelse

Nøkkelbegrep

Algoritme	Kode som brukes til å utføre oppgaver (Kapittel 3.1.1)
Arbeidsflyt	System av hjelpemidlene (noder) i visual programming
BIM	Building Information Modelling (Kapittel 2.1.1)
C#	Programmeringsspråk (Kapittel 3.2.4)
Computational Design	Metode for å undersøke resultater utfra forskjellige variabelverdier (Kapittel 3.1.3)
Grensesnitt	Uteseende til programvarer, som brukes til å utføre oppgaver i programmene.
IFC	Industry Foundation Classes – filformat for BIM-modeller
Node	Hjelpemiddel innenfor visual programming
Parameter	Egenskaper knyttet til modellelementer
Parametrisk design	Bruk av parameter til å utføre endringer til elementer i en modell (Kapittel 3.1.2)
Pareto Front	Diagrampresentasjon av resultater (Kapittel 3.2.2.2.1)
Python	Programmeringsspråk (Kapittel 3.2.4)
Skript	Et system med koder/algoritmer
Variabel	En verdi som kan endres
VDC	Visual Design Construction (Kapittel 2.1)
Verktøy	Programmer som brukes til å løse oppgaven.

1. Introduksjon

Hva skal jeg gjøre?

I denne oppgaven ønsker jeg å finne en metode for å optimalisere daglys (i form av dagslysfaktor) for en boligenhet i.h.t. satte variabler. Dette vil bli utført ved hjelp av parametrisk design (regelsett for variabler) som beslutningsstøtte. Dette betyr at ulike kombinasjoner av variabler vil bli simulert og testet mot hverandre for å finne de mest optimale alternative løsningene med tanke på dagslys.

Dette er en metode som ikke *kun* omhandler dagslysberegninger, men også kan brukes til å optimalisere flere aspekter ved designet av en bygning. Jeg ser derfor på denne oppgaven som et pionerprosjekt for bruk av parametrisk design som beslutningstøtte, for å vise frem hvilke muligheter denne metoden vil kunne bringe i fremtiden. Personlig ser jeg for meg at dette kan bli en stor del av designfasen i bygge bransjen i fremtiden.

Hvorfor gjør jeg dette?

Bygge-bransjen prøver å holde følge med den teknologiske utviklingen, og vi har allerede sett mange måter hvor de teknologiske fremskrittene har effektivisert de forskjellige prosessene. Nye metoder som Integrated Concurrent Engineering (ICE) og Building Information Modelling (BIM) har hjulpet til å spare tid og ressurser gjennom hele prosjekt- og byggefasen. I tillegg kan disse metodene levere mer nøyaktige og kalkulerte løsninger enn tidligere.

I dagens planlegging går mesteparten av designbeslutningene ut på manuelle beregninger for hver enkelt kriteria. I denne oppgaven blir dagslysberegninger som kriteriet fokusert på, og det er lagt krav for minimumsmengder med dagslys for enkelte rom i et prosjekt, slik at dette må dokumenteres. I dag utføres dette som oftest manuelt ved hjelp av matematiske formler, og simuleringsverktøy. Andre kriterier bruker også lignende metoder.

Designprosessen for prosjekter er en tradisjonell metode på den måten at den ikke har endret seg mye over årene. Arkitekten bruker ofte intuisjon og erfaring for å nå beslutninger om hvilke løsninger som vil være best for prosjektet. Det vil kun være et lite antall alternative løsninger som blir undersøkt på denne måten, grunnet tid og ressurser som trengs til å gå gjennom hver enkelt.

Når det gjelder bruken av parametrisk design i dagens prosjekter blir det ikke brukt til sin fulle styrke, siden mulighetene bak det ikke har blitt visst frem i til sitt fulleste. Spesielt når det gjelder parametrisk design innen dagslysberegninger har det ikke blitt visst frem noen gode eksempler som gjør det verdt å se bort fra de tradisjonelle metodene med matematiske formler og simuleringsverktøy. Om denne oppgaven får gjennomslag vil den kunne erstatte disse metodene, og effektivisere prosjekteringsfasen.

I dag er har ikke parametrisk design blitt utforsket til sitt fulle potensial. Enkelte bruker det til å modellering, gjennom manipulering av prosjektets parametere knyttet opp mot modell-elementer. Denne manipuleringen kan bruke regler og logikk knyttet opp mot parameterne til å lage abstrakte designløsninger. Andre ser også på metoder som Generative Design (Kapittel 3.1.3) og Computational Design (Kapittel 3.1.3) hvor programmene kan simulere gjennom forskjellige designløsninger, i et forsøk for å kartlegge de beste designløsningene. Disse metodene knyttes opp mot regler og logikk for å kunne regne seg frem til gode løsninger. Det er få som utforsker disse metodene i dag, selv om dette konseptet har et veldig stort potensial. Med en god metode for bruk av parametrisk design som beslutningsstøtte kan bli essensielt i prosjekteringsfasen for entreprenører.

Tilfellene jeg har sett innebærer ikke kun dagslysberegninger, men noen ganger er dette del av et større system. Tanken er at hele designforslag skal kunne ta i betraktning flere forskjellige kriterier, som dagslys, tomteutnyttelse, BRA%, energibesparende tiltak, osv. På denne måten vil man kunne bruke lignende kriterier og forskriftspåleggelser til å produsere bedre designløsninger, samtidig som man sparer tid og ressurser. Disse metodene er som sagt kun delvis utforsket, hvor forskjellige programmer og metoder blir brukt. Enkelte ganger er dette egenutviklede programmer som kun er tilgjengelig for produsenten selv.

Med denne oppgaven ønsker jeg å finne en måte for å kunne kartlegge de beste designløsningene. Denne oppgaven tar for seg dagslysberegninger som kriteria, men kun som eksempel for å finne en metode som også kan anvendes andre kriterier. Dette betyr at om denne oppgaven får gjennomslag vil det kunne legges til andre kriterier enn kun dagslys, og forhåpentligvis kunne ta for seg alle viktige kriterier som må planlegges for når man skal produsere et design. I denne oppgaven vil populære programmer brukes, som de fleste entreprenører og arkitekter allerede har tilgang eller kjennskap til. På denne måten kan denne metoden også ha potensial til å kunne være lettere å implementere til gjeldene prosesser, samt lettere å bruke siden programmene allerede er kjent for de fleste.

Med tanke på dette seg jeg for meg at parametrisk design er fremtiden, men først må en god metode for å anvende dette bli funnet. Hvis denne oppgaven får gjennomslag vil prosjekteringsprosessen kunne automatiseres til en viss grad, hvor parametrisk design har potensialet til å heve kvaliteten til designløsningene for et prosjekt; Parametrisk design kan brukes til å utforske ***alle*** designløsninger, med alle kriterier (ikke kun dagslys). På denne måten også kunne finne ***nye*** og ***bedre*** løsninger som arkitekten ikke nødvendigvis hadde funnet. En god metode kan også utføre presise kalkulasjoner ved hjelp av simuleringsverktøy, hvor disse kan implementeres i selve designfasen, i motsetning til ***etter***. Hvert prosjekt har forskjellige forhold som må tas rette for grunnet ***lokasjon*** og ***klima***, men denne metoden har mulighet til å ta i bruk data og tilrettelegge designet utfra disse forholdene. I dag vil arkitekter og ingeniører utføre disse beregningene for enkelte designløsninger de selv velger ut, men om vi hadde brukt parametrisk design ville hele denne prosessen blitt ***mye*** mer effektiv.

I tillegg til metoder, teknikker og programmer trengs det også kompetanse innenfor de digitale verktøyene, og dette er og en viktig faktor for hvorfor jeg skriver denne oppgaven, hvor jeg ønsker å utvide min kompetanse innen BIM og parametrisk design, og alt det innebærer.

For å oppsummere: Dette er hvorfor jeg vil se etter en metode for å anvende parametrisk design, i tillegg til computational design; Effektivisering av hele prosjekteringsfasen. Hvor *flere* alternativer kan bli undersøkt og simulert, slik at bedre løsninger kan bli funnet. Kalkulasjonene kan bli *presist* utført med gode simuleringsverktøy *for hvert* alternativ, slik at ingen blir oversett. Alt i alt har dette potensialet til å delvis automatisere, samt heve kvaliteten for hele prosjekteringsfasen, hvor designet er skreddersydd for prosjektets lokasjon og forhold. På denne måten vil i tillegg *tid og ressurser* bli spart under denne fasen i prosjektet, *i tillegg* til å heve kvalitet. Dette er hvorfor jeg ser for meg at parametrisk design er fremtiden, og hvorfor jeg utfører denne oppgaven. Hvor jeg kan hjelpe med å bringe potensialet til parametrisk design inn i lyset, i tillegg til å styrke kompetansen min innenfor temaet.

Hva trenger jeg for å gjøre dette?

Program som brukes til å modellere, hvor geometrien til bolig vil bli brukt som referanse for analysene.

Programmer som kan lage skripter til å utføre oppgavene, samtidig som det kan kommunisere med modelleringsprogrammet for å oppdage eller utføre endringer i sanntid. Oppgavene som må utføres gjennom skriptene er dagslysberegning og optimalisering av dagslysberegningen. Resultatene fra analysen må også kunne presenteres på forståelig vis, så samme program som brukes til å lage skriptene bør ha muligheten til dette.

Som tidligere nevnt trengs også god kompetanse innenfor bruken av disse programmene, og dette punket bærer mest vekt siden det kan ta lang tid å bygge opp en god kompetanse innenfor de forskjellige programmene.

2. Bakgrunn

Dette kapittelet vil legge et grunnlag for hvorfor denne oppgaven blir utført.

2.1 Virtual Design Construction (VDC)

Mange vet om begrepene Building Information Modelling og Integrated Concurrent Engineering, eller forkortelsene deres BIM og ICE. Disse verktøyene og teknikkene er deler av VDC. VDC er ikke én teknikk eller ett verktøy, men setter disse elementene i et system (Linge, s.a.). VDC sikter mot et helhetlig bilde over prøvde tekniker og verktøy for å effektivisere alle fasene i et prosjekt. Det handler om samhandlingen mellom mennesker og disse verktøy og tekniker.

BIM, ICE, osv. er kun biter av VDC, men VDC sikter altså til å sette disse i et system hvor de kan bygge på hverandre. På denne måten kan det hjelpe forståelsen for hvordan disse verktøyene og teknikkene kan brukes om hverandre i et mer helhetligbilde. VDC blir brukt til å forbedre kommunikasjon mellom alle aktører og deltakere i et prosjekt, for å redusere misforståelser. På denne kan prosjektets prosesser effektiviseres fra start til slutt, og føre til tid-, kost- og ressursbesparelser gjennom prosjektet.

Ved å anvende disse elementene kan informasjon samles fra hele prosjektfasen, som videre kan brukes til å forbedre prosessen i fremtiden.

2.1.1 Building Information Modelling (BIM)

Building Information Modelling, eller Byggingsinformasjonsmodell på norsk, har blitt et sentralt begrep innenfor VDC og bygg-bransjen de siste årene. Det er et viktig verktøy som hjelper til å effektivisere prosjekter, fra prosjektering til produksjon. Det omhandler en geometrisk 3D modell, hvor informasjon er implementert i alle modell-elementer. Denne informasjonen i modellen er selve essensen i BIM, og åpner for mulighetene som BIM kan by på. Tidligere når byggene ble prosjektert på papir var det vanskelig å få med informasjon til denne graden, men nå kan denne informasjonen anvendes fra start til slutt i prosjektet. Ved å bruke BIM kan kommunikasjonen og samhandling mellom aktører og deltakere i prosjektet forbedres, hvor alle jobber rundt samme modell med god oversikt over hvordan deres arbeid er i sammenheng på andre fag. Når flere fag kan jobbe med samme modell, vil også deres samhandling forbedres. Dette kan føre til diverse tid-, kost- og ressursbesparelser hvor feil blir oppdaget tidlig, i motsetning til på byggeplassen.

Det er ikke kun informasjonskapasiteten som er forbedret fra tradisjonelle metoder, men også hvor effektivt endringer kan utføres i et prosjekt. Modelleringsprogrammene tillater for både store og små endringer ved hjelp av noen tastetrykk, hvor man i tillegg kan kvalitetssikre modellen gjennom endringer ytterligere ved hjelp parametrisk design (kapittel 3.1.2).

BIM brukes ikke bare i prosjekteringsfasen, men også i produksjonsfasen. BIM-kiosker har blitt mer og mer populært på anleggsplasser, og gir i gjendeld detaljerte og oversiktlige tegninger/modeller av prosjektet.

BIM eier potensialet til å effektivisere og automatisere diverse prosjektfaser, hvor kalkulasjoner utføres av datamaskiner og presisjonsarbeid av roboter kan lede vei for hvordan hele prosjekter gjennomføres i fremtiden.

Modeller og byggeelementer kan moduliseres i modelleringsprogrammene, slik at de kan deles med andre og lett implementeres i forskjellige prosjekter. På denne måten vil sikre, prøvde modeller kunne brukes om igjen uten å måtte prosjekteres om igjen.

Alt i alt hjelper BIM til å levere kvalitetssikre modeller, med muligheten til å simulere gjennom prosjektets prosesser for å oppdage forbedringspotensialer og feil. I tillegg gir det innsikt i flere dimensjoner av et prosjekt; 2D for planer, 3D for presentasjon og visualisering, 4D for planlegging og fremdrift, i tillegg til andre dimensjoner knyttet til bl.a. tid og kostnader (Linge, s.a.). Med slik detaljert planlegging åpner muligheten for å se utførelsen av prosjektet før produksjonen er i gang.

2.2 Parametrisk design i bygg-bransjen

Bruk av parametrisk design verktøy gir brukeren muligheten til å legge inn betingelser, grenser og verdier som brukes til å lage et design. Disse verktøyene bruker algoritmer (kapittel 3.1.1) til å generere geometri og design. Brukeren legger inn instrukser til programmet som brukes til å styre designet. Disse instruksjonene kan endres underveis, hvor designet vil endres i sanntid. Dette åpner muligheten for å implementere logikk inn i algoritmene, som underbygger regler og begrensinger, satt av brukeren, for et sterkere design. Parametrisk design gir også muligheten for å skape relasjon mellom elementene i et design, slik at de kan reagere til endringer hos hverandre. I tillegg kan analyser implementeres i disse verktøyene, som videre sikrer designet ytterligere.

Med disse verktøyene har vi også muligheten til å automatisere og optimalisere disse designprosessene. Programmene jobber allerede med algoritmer og algoritmisk logikk, slik at det finnes nesten ikke grenser for hva som kan bli mulig i fremtiden. Ved å legge til algoritmer som kan regne seg frem til hva som gjør et design er bedre enn andre, kan vi også videreutvikle dette slik at programmet kan finne et design som er bedre igjen. På denne måten kan vi tenke oss at parametrisk design har muligheten til å bli en vital del av design prosessen, hvor den kan kalkulere seg frem til en god løsning raskere enn et menneske, med større sikkerhet for at den er bedre enn samtlige andre løsninger. Mennesket har begrenset kapasitet (og tid) når det gjelder å beregne disse designene, mens disse verktøyene i tillegg kan finne utvikle nye design, som et menneske ikke nødvendigvis hadde kommet over.

Ved å kombinere parametrisk design med BIM kan prosjekteringsfasen bli ytterligere effektivisert. Med sikrere designløsninger kan tid og kostnader reduseres for prosjekteringen, i tillegg til at materialbruken for bygningen kan reduseres. Men alt dette kommer an på algoritmene som skal utføre disse oppgavene. Om man har utviklet gode algoritmer, som har bevist seg i prosjekter kan de lagres, deles og brukes i andre sammenhenger. På denne måten trenger man ikke lage dem på nytt.

Parametrisk design har blitt mer brukervennlig med årene, og er den dag i dag implementert i en rekke modelleringsprogramvarer; Ett av disse er Autodesk Revit (som blir forklart videre i kapittel 3.2.1). I tillegg brukes parametrisk design i diverse visual programming verktøy.

2.3 Dagslysberegninger

Dagslysberegninger brukes til å dokumentere dagslysforhold for bygninger. Dagslys varierer med årtid og tid på døgnet, i tillegg til bygningsorientering, høyde over havet, og skyggeskapende elementer i omgivelsene. Når bygget prosjekteres må også vindusdimensjoner og refleksjonsfaktoren for materialer også tas i betraktning, ettersom de også påvirker mengden dagslys som kommer inn i bygningen. Dokumentering av dagslysberegninger for prosjekter er pålagt utfra forskrifter, hvor det er et minimumskrav til dagslysfaktor (se kapittel 3.1.5) for oppholdsrom.

Mangel på dagslys kan føre til trøtthet, i tillegg til "mørketidsdepresjon" i enkelte byer med lite sollys i vinterhalvåret (SINTEF, 2016, s.4). Dette påvirker menneskers evne til å utføre produkt arbeid, og er dermed viktig å dokumentere beregningene. Definisjonen av oppholdsrom er rom hvor personer vil oppholde seg over lengre tid, og gjelder da både stue/soverom i boliger, samt kontorlokaler og lignende.

Ved å følge disse forskriftene vil minimumskrav for dagslys være fulgt, uansett geografiske-, klimatiske forhold eller bygningsorientering (Lyskultur, 2014). Daglysfaktor kan kalkuleres med matematiske formler eller ved hjelp av simuleringsverktøy. Dagslysberegninger vil bli videre diskutert i kapittel 3.1.5.

2.4 Kruse Smith

Kruse Smith er en av Norge største entreprenører, og har avdelinger innenfor bygg, byggfornyelse, anlegg og bolig- og eiendomsutvikling. Etablert i 1933 med hovedkontor i Kristiansand (Wikipedia, 2018), strekker virksomheten seg langs kysten fra Bergen til Oslo.

Kruse Smith jobber med fokus på innovative løsninger, samt solide og bærekraftige resultater. Dette innebærer at de aktivt jobber mot implementering av moderne verktøy og tekniker, som kan effektivisere og kvalitetssikre oppdrag. Blant disse er har mulighetene ved parametrisk design blitt utforsket i senere tid.

Denne oppgaven ble utlyst av Kruse Smith før årsskiftet (2017/2018), som jeg søkte på og ble tildelt. Kruse Smith verdsetter muligheten til å jobbe med slike oppgaver, både bachelor- og masternivå, og har et bra system for oppfølging i løpet av oppgavesemesteret.

2.4.1 Konseptbolig

Konseptboligene er en annen av de nye metodene Kruse Smith utforsker for å styrke prosjektene sine, både gjennom prosjekt- og produksjonsfasen. Dette konseptet innebærer bruk av moduler som kan komplekser settes sammen av utallige forskjellige kombinasjoner. Ved bruke disse modulene med satte dimensjoner kan det være mulig å bruke verktøy/programmer til å simulere og kalkulere gode sammensetninger ved hjelp av parametrisk design.

Disse modulene blir i dag brukt til å prosjektere Havneparken i Sandnes, og dette har blitt brukt som referanseprosjekt for oppgaven. Oppgavens hovedfokus gjelder hvordan parametrisk design kan bli brukt som beslutningsstøtte, slik at dette referanseprosjektet ikke nødvendigvis vil bli utforsket til full grad.

3. Teori & Programmer

Dette kapittelet vil gjennomgå teorier og programmer som blir brukt for å løse oppgaven, og er ment å gi leser en grunnleggende forståelse for hvordan teoriene er anvend programvarene, både generelt og for å løse denne spesifikke oppgaven.

3.1 Teori

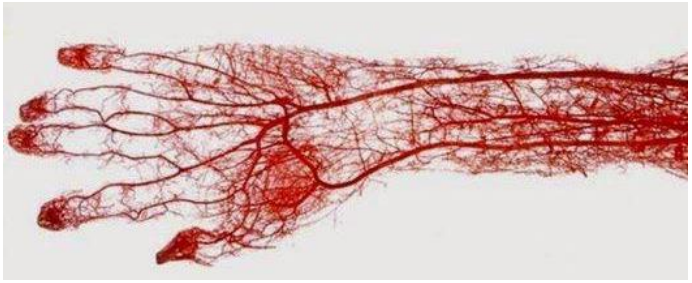
Utførelsen av simuleringene er bygget opp av en rekke teorier. Mange av disse teoriene bygger også på hverandre, og er tett knyttet til hverandre. Mye av denne teorien er implementert i programvarene, men dette kapitlet vil i første omgang være ment bygge en forståelse for disse teoriene, og hvordan de anvendes. Riktig anvending av disse teoriene kan hjelpe til å effektivisere prosjekteringsfasen.

3.1.1 Algoritmisk Teori

Hva er en algoritme? En algoritme er et sett med steg eller instrukser over hva som skal utføres i en oppgave. En kode (innen programmeringsverden) er en algoritme. Alt fra input, videre til prosessen for hva som skal bli gjort med disse verdiene, og så returnere resultatet fra prosessen for videre bruk. Disse prosessene kan variere i stor grad. Vi kan ta for oss et eksempel, hvor vi skal legge sammen to verdier; Dette vil skje steg-for-steg i den algoritmiske verden. Først vil oppgi verdiene *x* og *y*. Neste steg blir å legge disse to verdiene sammen. Denne prosessen vil bli utført ved å anvende matematisk logikk, og resultatet vil bli tatt videre til neste steg; Resultat (eller output). Resultatet kan så bli brukt videre i andre algoritmer eller bli hentet vist til som endelig resultat. Dette er et veldig simpelt eksempel, men mer avanserte oppgaver er bygget opp på samme måte (steg-for-steg). En algoritme er altså et sett med steg som, som kombinert danner en prosess. Samtidig kan en algoritme ses på som biter med koder, eller motsatt kan koder ses på som algoritmer.

Når vi snakker om algoritmisk kode kan vi trekke paralleller til naturen, hvor naturen pakker informasjonen som kode inn i celler. Disse cellene utfører diverse funksjoner utfra deres DNA og tilhørende celler. På denne måten kan komplekse systemer bli bygget til å utføre alle mulige oppgaver, som vi f.eks. oppbyggingen og formene til organisk liv hvor DNAet (koden) kan bli satt sammen til å danne forskjellige skapninger. Vi ser mange gjentakelser på tvers av organisk liv hvor samme «kode» har blitt brukt i forskjellige skapninger. Selv om dette høres usaklig ut nå vil det bli vist til likheter innen programmering for hvordan dette kan bli anvendt (Kapittel 3.1.7).

Figur 1: Årer i et blad (Leaf-Veins) *(fwallpapers.com, 2018)*



Figur 2: Blodårer i en arm (Blood Vessels) *(tellmewhyquestions.blogspot.com, 2016)*

Mer faglig sett er *algoritmisk kode* grunnleggende innenfor programmering, og sikter til kodene inne i "cellene". På samme måte som naturen bruker celler til å utføre oppgaver, brukes algoritmisk kode innen programmering for å utføre oppgaver. Hvor forskjellige organiseringer av celler (med kode) kan bli strukturert til å utføre forskjellige typer oppgaver.

Algoritmisk design sikter til måten cellene (kodene) er organisert i et system til å samhandle med hverandre og utføre oppgaver, hvor sammensetningen av kodene dikterer deres funksjon. På denne måten kan mange forskjellige systemer bli bygget kun ved å endre på organiseringen av kodene, og dette er bakgrunnen for hvordan programmeringsspråk er bygget opp.

Algoritmisk engineering legger grunnlag for implementering av algoritmisk teori i programvarer, slik at den kan settes ut i praksis. På denne måten kan vi bruke den algoritmiske logikken i programmene.

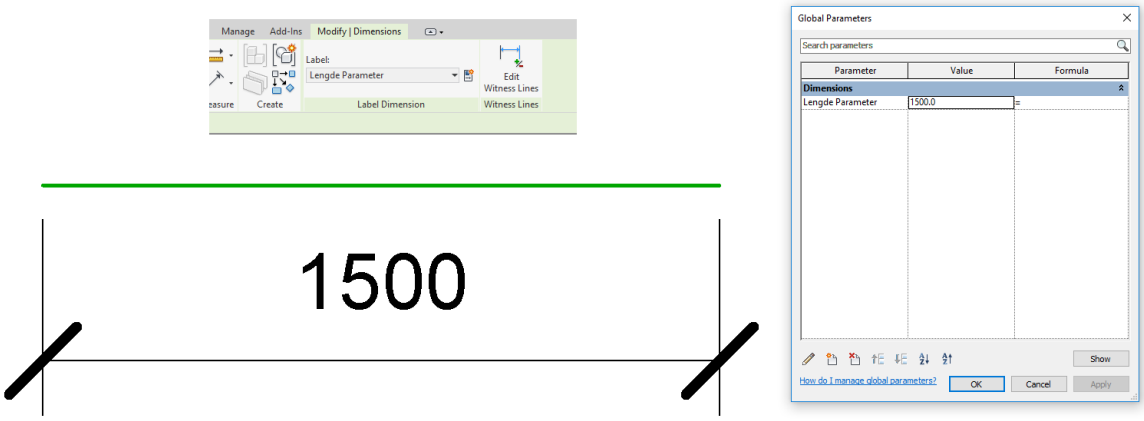
De neste teoriene har alle algoritmisk teori som grunnlag, hvor de alle er bygget opp til å følge denne logikken.

3.1.2 Parametrisk design

Først, en parameter er en variabel i en funksjon som kan endres, som videre kan brukes til å endre resultatet av funksjonen. Disse parameterne kan endres ved hjelp av input (i form av brukersatte verdier eller resultater fra andre funksjoner). Ved å koble disse parameterne opp mot elementer i et prosjekt kan vi styre og manipulere elementene direkte gjennom parameterne, samtidig som man er sikret at parameterne opererer med verdier innenfor satte grenser. Dette er en videreutvikling av algoritmisk teori hvor parameterne kan styres av begrensinger, regler og/eller andre parametere, anvendt ved hjelp av kode (Wikipedia, 2018). Ved å koble parameterne opp mot hverandre vil de også kunne reagere til endringer i andre parametere. Dette gir en dynamisk arbeidsflyt og en robust modell, hvor man kan styre et helt prosjekt gjennom parametere.

Når vi kobler en parameter til et element i prosjektet, legger vi også til informasjon i elementet. Dette er dette som er tanken bak BIM. Denne informasjonen kan bli brukt både under og etter prosjekteringsfasen, og gir et detaljert innblikk i de forskjellige elementene og deres funksjon.

Parametrisk design er blant annet integrert i enkelte modelleringsprogramvarer, som Revit, og om du noen gang har jobbet i et slikt program har du jobbet med parametrisk design.



Figur 3: Eksempel av en lengde-parameter i Revit.

Ved å bruke å parametrisk design til å videreføre en idé til geometri åpnes mulighetene til videre eksperimentering med designet gjennom parameterne. Ved å endre noen av disse parameterne kan et resultat bli endret fullstendig. I tillegg blir det mulig å utforske og videreutvikle idéer som tidligere ikke var tatt i betraktning, men er resultat fra eksperimentering med parameterne.

Parametrisk design gir muligheten til å utføre store endringer til et prosjekt kun ved å endre noen parametere, i motsetning til de tradisjonelle metodene hvor en arkitekt må gjøre dette for hånd.

3.1.3 Computational design

Computational design endrer perspektiv fra parametrisk design, hvor man fokuserer potensielle resultater istedenfor inputene. Resultatet trenger ikke være kjent på forhånd, men er ment å utforske alle mulige løsninger slik at man kan finne den aller beste.

Det ønskede resultatet trenger fortsatt inputs (i form av parametere), men hvilke verdier disse trenger for å oppnå resultatet er uvisst. Dette er hvor parametrisk design kommer inn.

En kunstig intelligens (artificial intelligence eller A.I.) i simuleringsverktøyet bruker algoritmer for å simulere gjennom parametere innenfor de satte grensene, slik at «alle» mulige resultater har mulighet til å bli utforsket. Etter simuleringene kan brukeren gjøre endringer i mål og begrensinger for å prøve å oppnå et mer ønskelig resultat, om resultatene fra første runde ikke er gode nok. På denne måten kan simuleringsverktøy finne alle mulige løsninger for en oppgave, hvor en person tidligere måtte regne ut disse for hand. Tradisjonelt sett måtte en arkitekt tidligere regne ut 15-20 forskjellige designforslag for å se hvilke av disse som gir mest ønskelig resultat, men nå kan en datamaskin brukes til å regne gjennom tusenvis av designforslag.

Disse simuleringene kan strekke seg etter ett eller flere mål. Hvis det blir satt opp flere motstridende mål vil simuleringen prøve å finne best mulig resultat og komporimiss mellom målene. I denne oppgaven f.eks. ønsker vi å maksimere mengden dagslys inn i boligen samtidig som vi ønsker å minimere material kostnadene. Disse er to motstridene mål. Vi får mer dagslys ved å bruke større vinduer, behandle overflater til å reflektere mer lys, osv., men bruker samtidig mer materialer. Denne metoden prøver å finne de alternativene som gir best resultat for begge målene. Når alle løsningene er simulert vil programmet kunne presentere de beste løsningene i.h.t. de satte målene. På denne måten kan brukeren fortsatt få velge ut den løsningen anses som best, utfra kompromisset mellom målene. Denne metoden for å utføre simuleringene kalles «Multi-objective optimization».

Når alle løsningene er simulert kan de beste bli presentert i en såkalt *Pareto Front* (kapittel 3.2.2.2.1). I dette diagrammet vil alle de presenterte løsningene være like gode, men med forskjellige kompromiss mellom målene. Her kan brukeren velge ut hvilket resultat som vil være best utfra hvilke kompromiss som er mest verdsatt. For denne oppgaven kan dette da gjelde to resultater med forskjellig mengde dagslys og materialer, men fortsatt være like gode. Dette vil si at ett resultat kan gi mye dagslys, men vil kreve større kostnad til materialbruk (til vinduer og overflatebehandling), mens et annet alternativ som gir mindre dagslys fortsatt er et like godt alternativ siden det tar bruk av mindre materialkostnader. Her kan brukeren velge hvilket av alternativene som de vil ta videre utfra deres egen preferanse.

Ved å bruke computational design kan man spare tid og ressurser, samtidig som man kan finne løsninger som man ikke hadde funnet med tradisjonelle metoder. Dette er veldig likt parametrisk design, som computational design er en videreføring av.

Det finnes også en lignende metode som *kalles Generative design*, men denne metoden kan gi varierende resultater i forhold til computational design. Begge metodene simulerer gjennom alle mulige kombinasjoner, men hvordan man finner de beste kombinasjonene varierer mellom metodene. I generative design er det brukeren selv som må gå gjennom alle mulige - og så velge ut hvilke kombinasjoner som anses som best. Mens i computational design er det maskinen som bestemmer hvilke kombinasjoner som er best, utfra brukersatte regelsett. Ved å bruke computational design vil det resultatene være valgt på et empirisk grunnlag, og vil dermed være en sikrere måte å se til at de beste kombinasjonene blir undersøkt. På denne måten kan computational design brukes som en optimaliserings-metode, hvor programmet klarer å finne de mest optimale kombinasjonene utfra satte mål.

3.1.4 Matematisk optimalisering

Matematisk optimalisering er veldig enkelt sagt: En måte å maksimere eller minimere resultatet av en funksjon, utfra variabler. Dette er hvordan fitnessfunksjonene i optimaliseringsfunksjonen fungerer.

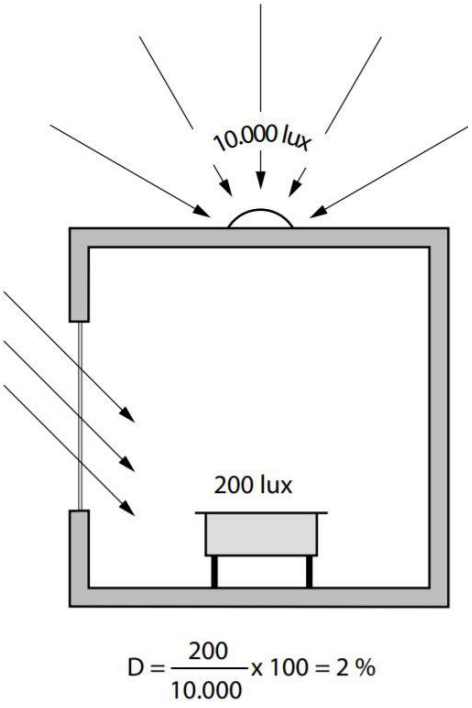
3.1.5 Dagslysberegninger

Dagslysberegningene i oppgaven baserer seg på beregning av *dagslysfaktor*, men siden fokuset er på parametrisk design som beslutningsstøtte og selve metoden for hvordan dette kan gjøres vil detaljnivået for dagslysberegningene være veldig enkelt.

Dagslysfaktor viser til forholdet mellom innendørs og utendørs belysningsstyrke under et definert himmelforhold, målt for et gitt punkt innen- og utendørs. Formelen for dette blir da:

$$Dagslysfaktor = \frac{Innendørs\ belysningsstyrke\ (lux)}{Utendørs\ belysningsstyrke\ (lux)}$$

Utfra forskrifter er det krav til minimum 2% dagslysfaktor i oppholdsrom i en boenhet. Dette kravet kan oppnås ved flere kombinasjoner av variabler som vindusdimensjoner, refleksjonsoverflater og skyggeforhold. For å beregne dagslysfaktor kan man bruke matematiske formler eller simuleringsverktøy (programvare). Ofte blir en kombinasjon av disse brukt, slik at resultatene kan bli kryssjekket. I denne oppgaven vil det kun bli brukt simuleringsverktøy til å utføre beregningene, siden oppgaven fokuserer på å automatisere denne prosessen.



Figur 4: Dagslysfaktor - prinsipp (Lyskultur, 2014)

Noen av faktorene (variablene) som påvirker daglys er: Lokasjon inkludert høyde (etasjer), bygningsorientering i.h.t. himmelretning, vindusdimensjoner, refleksjonsevne til materialer (vegger, gulv, osv.), og geometri som lager skygger (balkonger, osv.).

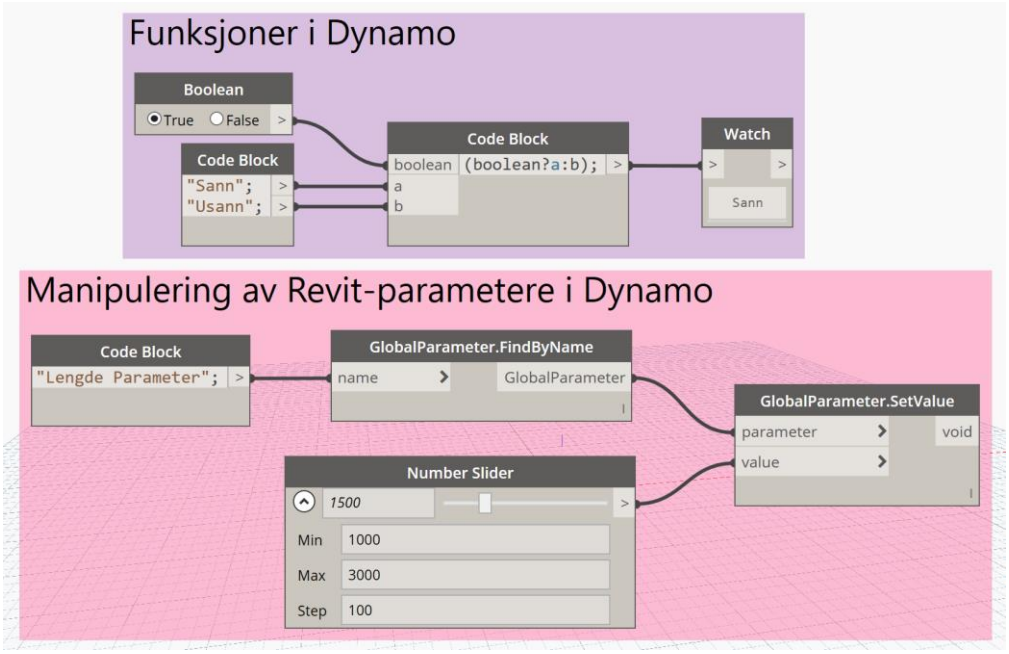
Siden det i første omgang vil fokuseres på å finne en metode for å kunne bruke parametrisk design som beslutningsstøtte vil det ikke fokuseres for mye på å presentere resultatet for dagslysfaktor, men heller hvordan programmet vil beregne seg frem til de beste resultatene for dagslys generelt. Dette betyr at presentasjon av resultatene kan variere utover i oppgaven.

3.1.7 Visual Programming

Visual programming sikter til grensesnittet som skal hjelpe brukeren til å behandle designet og tar i bruk visuelle hjelpemidler, som f.eks. nodene som blir brukt i Dynamo (se kapittel 3.2.2). Disse nodene baserer seg på algoritmisk tenkning, hvor de er bygget opp av koder til å utføre oppgavene sine. Visual programming baserer seg altså på algoritmisk programming, men er forenklet form slik at man ikke trenger sterk kompetanse innen programmering for å kunne bruke dem. På denne måten gir dette mange av de samme mulighetene som algoritmisk programming, men gjør det samtidig lettere å bruke for de som ikke eier kompetansen. I kapittel 3.1.1 ble det dratt paralleller til naturen, hvor sett med koder er lagt inn i celler. Hjelpemidlene er laget på samme vis, hvor vi kan se på dem som celler med kode. Kombinert med andre celler kan disse danne et komplekst program som utføre avanserte oppgaver.

Når vi snakker om algoritmer, betyr det at vi setter opp instruksjoner for hva vi ønsker at programmet skal gjøre. Dette utføres steg-for-steg, og kan bli satt opp på to forskjellige måter: *Tekstbasert* anvender algoritmene direkte opp mot programmet for å utføre oppgavene, mens *visual programming* bruker hjelpemidlene slik at brukeren kan organisere algoritmene til å utføre oppgavene. Et kjapt eksempel for å gi et bedre bilde over hvordan disse metodene funger er hvordan en bruksanvisning for en kommode fra IKEA er satt opp. Bruksanvisningen eller stegene (algoritmene) for hvordan denne kommoden skal settes sammen kan bli satt sammen kan bli gjort både tekstbasert, visuelt, eller en kombinasjon av begge. Ved tekstbasert vil instruksjonene er listet i rekkefølge i form av tekst, mens visuelt vil det derimot kun være bilder som viser rekkefølgen for instruksjonene. Begge metodene vil resultere i en ferdig kommode, men hvilken måte å lese instruksjonene som er best vil variere fra menneske til menneske. Dette gjelder på samme vis for visuell- og tekstbasert programmering.

Visual programming er en videreutvikling av parametrisk design (kapittel 3.1.2), hvor systemet kobles opp mot parametere i prosjektet som samhandler og manipulerer hverandre. Ved å organisere og koble disse parameterne med noder kan brukeren lage komplekse programmer for et prosjekt, hvor en fortsatt har oversikt og logikk bak systemet.



Figur 5: Eksempel på skripter og noder i Dynamo (Kapittel 3.2.2).

Dette systemet er arbeidsflyten i prosjekteringen, og kan også bli omtalt som "skript". Det er satt sammen av prefabrikkerte hjelpemidler (noder) til å utføre diverse oppgaver. Om de eksisterende hjelpemidlene ikke kan gjøre jobben er det også mulig å utvikle sine egne, samt publisere dem eller laste ned hjelpemidler utviklet av andre.

Det finnes egne programmer som anvender visual programming, og et av disse er Dynamo for Revit som vil bli brukt i denne oppgaven.

3.1.8 Samhandling mellom teoriene

Som tidligere nevnt bygger mange av disse teoriene på hverandre, hvor disse kan sammen utgjøre oppgaven. På neste side er en figur som viser hvordan disse teoriene kan anvendes for å utføre oppgaven.

Figur 6: Samhandling mellom teoriene

3.2 Programmer

I dette kapittelet vil de relevante programmene bli gjennomgått. Programmene omtales også som verktøy. Dette kapittelet er ment for å gi en god forståelse for funksjonene til de forskjellige programmene, i tillegg til hvordan de brukes generelt. En mer detaljert forklaring for hvordan de brukes til å utføre oppgaven vil bli gitt i kapittel 4.

3.2.1 Autodesk Revit (v2018)

Revit er et modelleringsverktøy brukt av (bl.a.) ingeniører og arkitekter, som lar brukeren lage, endre og presentere prosjektmodeller. Det har et sterkt fokus på parametrisk design, hvor alle elementer (fysisk eller ikke) har parametere implementert. På denne måten bygges det informasjon inn i enkelt elementene og hele prosjektet, som videre gjør Revit til et BIM-verktøy. Revit kan brukes til å modellere flere fag (arkitektur/konstruksjon, VVS, elektrisk, osv.), og kombinere disse i samme prosjekt. Dette gir et helhetlig syn på prosjektet, og gir muligheten for å oppdage kollisjon mellom fag tidlig i prosjekteringsfasen. Prosjekter kan presenteres både i 2D og 3D, samt muligheter for 4D simuleringer. I tillegg kan prosjekter også bli "rendered" for å gi en virkelighetsnær representasjon av prosjektet.

Det er en rekke elementer (dører, vinduer, vegger, osv.) inkludert i programmet som standard, samtidig som det gir brukeren mulighet til å utvikle sine egne. Disse kan bli enten bli laget "in-place" (direkte inn i prosjektet), eller som en egen modell slik at den kan bli brukt i flere prosjekter om ønskelig.

Revit gir altså en god representasjon av prosjekter, men vil i denne oppgaven kun brukes til å lage geometrien som brukes i dagslysberegningene. I tillegg til å lage dør og vinduselementer med ønskede parametere som også brukes i beregningene.

3.2.2 Dynamo for Revit (v1.3.2)

Dynamo er et gratis tilleggsprogram for Revit. Det tar i bruk visual programming og lar brukeren lage en arbeidsflyt for Revit-prosjekter. Systemene som settes opp i Dynamo blir også omtalt som "skripter", hvor parametere i Revit kan styres ved hjelp av noder (også omtalt i kapittel 3.1.7). I tillegg kan disse systemene (skriptene) implementeres inn i "custom nodes", som ser ut som vanlige noder, men inneholder disse egenutviklede systemene som lar dem utføre diverse oppgaver. På denne måten kan et helt prosjekt styres gjennom Dynamo, hvor det samtidig er kommunikasjon mellom Dynamo og Revit. Dette betyr at parameterne til elementene i Revit kan styres og endres i sanntid, samtidig som de kan settes opp i komplekse systemer hvor alle parameterne har et forhold til hverandre.

Dynamo lar brukere som ikke har kompetanse innenfor programmering modellere med algoritmer, og ta i bruk de muligheter som de byr på. Det vil være lurt å unngå norske tegn (*æ, ø* og *å*) i skriptene, siden de ikke brukes av enkelte programmeringsspråk hvor de da ikke vil kunne leses av programmet.

Dynamo tillater også tredjepartsutvikling av "custom nodes" som kan lastes opp og ned av alle brukere gjennom Dynamo, som biblioteker med noder. Utfra hvilke oppgaver som skal utføres finnes det ofte et bibliotek som kan hjelpe. I denne oppgaven brukes fire slike tredjeparts-pakker: Optimo, Ladybug, Honeybee og Archilab. Disse vil bli gjennomgått i de neste kapitlene (3.2.2.2 – 3.2.2.4).

Mot slutten av denne oppgaven ble det også utgitt en ny versjon av Dynamo (2.0.0). Ved første øyekast ser den ut til å ha vesentlige forskjeller fra forrige versjon, slik at potensielle komplikasjoner ikke nødvendigvis vil være et problem lenger. Nye oppdateringer er nødvendigvis ikke *kun* positivt, hvor enkelte eksisterende skripter og noder kan bli ubrukelige etter oppdateringen.

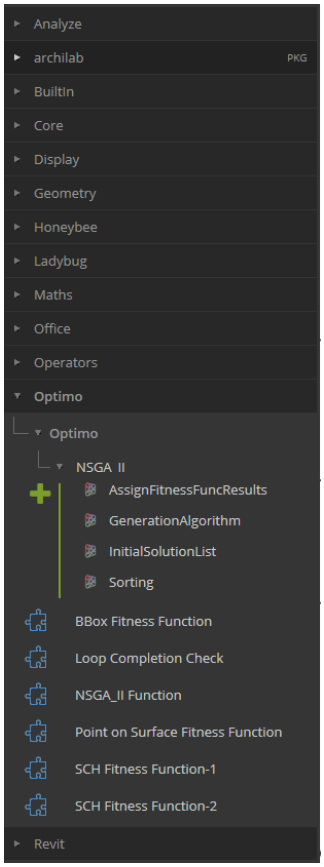
For mer informasjon om Dynamo, se http://primer.dynamobim.org/en/

3.2.2.1 Generelle funksjoner i Dynamo

Dynamo blir levert med en rekke funksjoner, hvor noen av disse vil bli brukt i denne oppgaven. I dette kapittelet vil de mest vesentlige bli forklart:

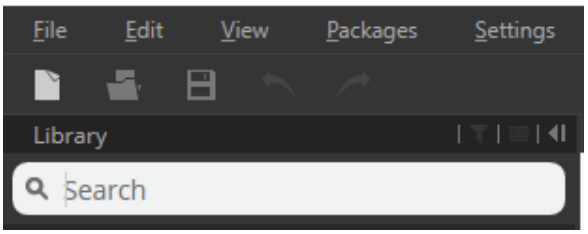
Generelle funksjoner i Dynamos grensesnitt:

Alle eksisterende noder i Dynamo kan finnes i biblioteket på venstreside av skjermen. Disse er gruppert i forskjellige kategorier og under-kategorier. I tillegg vil tredjeparts-pakker som blir lastet ned legges til i biblioteket som egne kategorier.



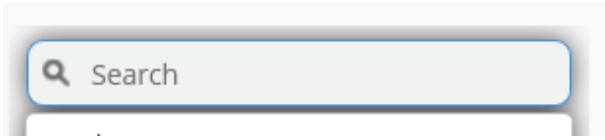
Figur 9: Node-bibliotek i Dynamo

Dynamo inneholder også en søkefunksjon, slik at brukeren raskt kan finne riktige noder uten å måtte lete gjennom biblioteket. Denne søkefunksjonen kan finnes på to måter; enten ved å bruke søkefeltet over biblioteket.



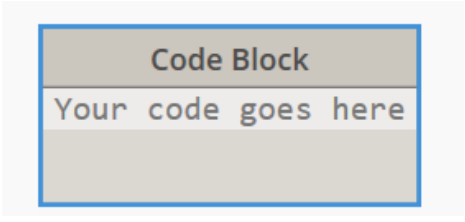
Figur 7: Søkefunksjon over bibliotek

Ellers kan søkefunksjonen finnes ved å høyre-klikke i arbeids-området, hvor søkefeltet vises over menyen.



Figur 8: Søkefunksjon ved høyre-klikk

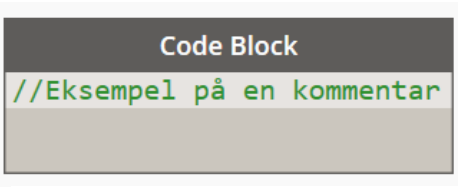
Andre funksjoner i grensesnittet er rask plassering av «Code Blocks» (Se neste delkapittel) ved å dobbeltklikke inne i arbeids-området. Code Blocks er mye brukt i skriptene, og denne snarveien kan dermed hjelpe på flyten når disse skriptene blir utviklet.



Figur 10: Code Block noden

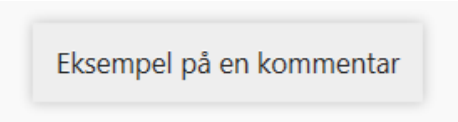
Code Block

Code blocks tillater brukeren å sette inn verdier eller utføre enkle funksjoner i Dynamo. Disse nodene kan kobles opp mot andre noder utfra hvilke algoritmer de inneholder. Brukeren kan sette inn verdier i disse Code Blockene som kan brukes videre av andre noder. Tall verdier kan skrives direkte inn i noden, mens tekst må startes og avsluttes med enten anførselstegn eller apostrofe (“ eller ‘). Disse verdiene kan lagres som lister i denne noden, og dette gjøres ved å starte og avslutte med krøllparantes ({}) i versjon 1.3.3 og lavere, og hakeparenteser ([]) i versjon 2.0.0. Flere verdier kan legges i samme liste ved å bruke komma (,) som skille. For å legge til flere linjer med verdier eller funksjoner må hver linje avsluttes med semikolon (;).



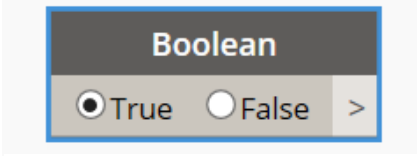
Figur 11: Kommentar i en Code Block

Inne i disse Code Blockene kan brukeren også legge inn kommentarer. Dette gjøres ved å legge til to skråstreker (//) først på linjen. Denne noden kan fortsatt brukes til å utgi verdier til skriptet, men disse verdiene kan ikke ligge på samme linje som kommentaren. Teksten vil vises som grønn når teksten er registrert en kommentar.



Figur 12: Kommentar som et notat

En annen måte å legge til kommentarer er å bruke kommentarnoder. Disse kan plasseres på to måter; Enten ved å velge «Create Note» under «Edit»-fanen, eller ved å klikke «Ctrl+W» på tastaturet. Disse kommentarene er selvstendige i forhold til resten av skriptet, og kan ikke kobles opp mot andre noder



Figur 13: Boolean-noden

Boolean

Boolean er en Sann/Usann-funksjon som tillater brukeren å velge enten *sann* eller *usann*, som vil bli gitt ut av noden som output. Enkelte noder trenger et slikt sann/usann input for å aktiveres, hvor da denne noden kan brukes. Om en input *alltid* skal være sann eller usann kan også Code Blocks brukes, hvor brukeren skriver inn «true» eller «false» i noden.

Number-/Integrer Slider

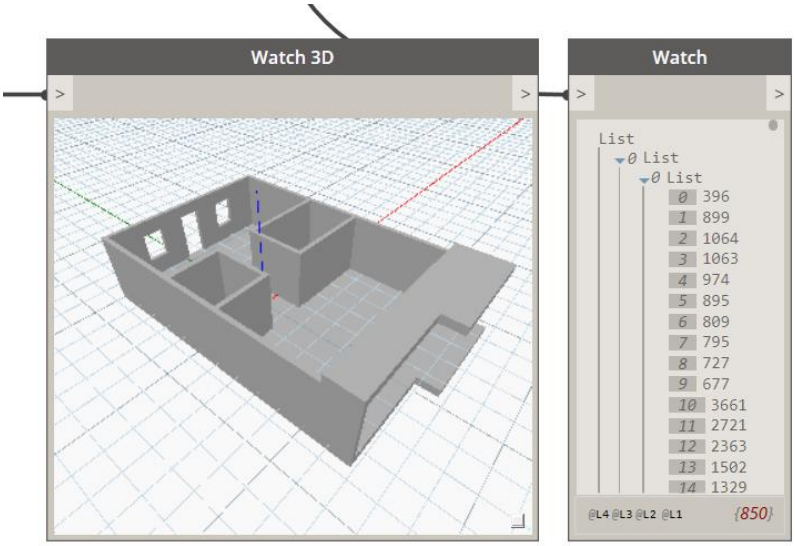
Number- og Integrer Sliders brukes til å gi ut tall verdier (Desimaltall for Number, og heltall for Integrer), samtidig som det er enkelt å endre disse verdiene om man ønsker. Det er også mulig å sette grenser og intervaller for disse sliderne, og innstillingene til dette gjøres vises/skjules ved å klikke på pilen til venstre i noden.

Figur 14: Number- og Integrer Slider nodene

Watch

Om man ønsker å se resultatet fra en node kan man enten holde musepekeren over noden, slik at et vindu vil vises under noden (ved å klikke på pinnen vil denne vises permanent). En annen måte å vise resultatet på er å bruke «Watch»-noder. Resultat som man ønsker å vise kobles til denne noden, som vil vise dette etter at skriptet har blitt uført. Denne noden kan flyttes på, i motsetning til vinduet under nodene.

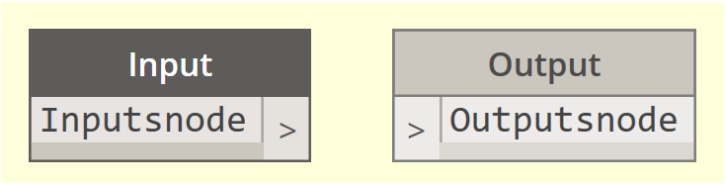
Det er også mulig å se 3D resultater fra Dynamo ved å bruke «Watch 3D»-noden



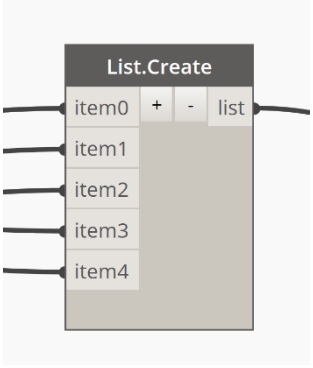
Figur 15: Watch 3D- og Watch nodene

Input/Output

Når man bruker *Custom Nodes* vil man ofte at funksjonen skal ta inn og gi ut data. Til dette brukes Input- og Output nodene. Når disse blir koblet opp mot andre noder i skriptet vil de vises på custom noden når den er lagret (Enkelte ganger må den plasseres på nytt i hoved-skriptet før disse vises). Navnene på disse in- og outputene kan settes av bruker ved å skrive det inn i verdifeltet. Disse kan kun brukes *inne* i custom noder, ikke i hovedskripter.



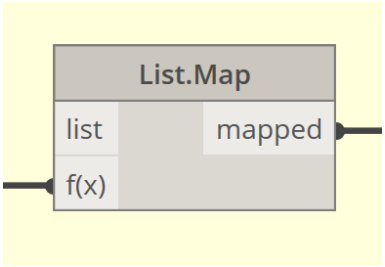
Figur 16: Input- og Output nodene



Figur 17: List.Create noden

List

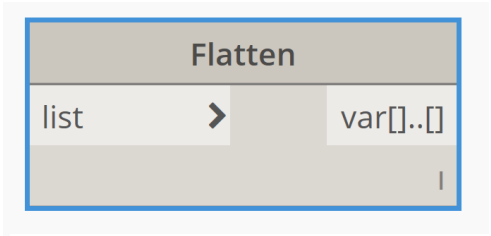
Dynamo operer ofte med lister. Disse listene kan inneholde diverse verdier (som tall, tekst-strings, funksjoner, osv.), som kan være plassert på forskjellige nivåer i listene. Man kan legge til eller trekke fra antall elementer i en liste ved å bruke «+»- og «-«tegnene. Enkelte funksjoner kan også regne ut resultater for hele lister, men enkelte ganger må man heller bruke List.Map (Se neste funksjon/node)



List.Map

Enkelte funksjoner klarer ikke å kalkulere lister, slik at ett og ett variabelsett på regnes ut om gangen. I slike situasjoner brukes List.Map til dette. Denne funksjonen vil mate inn ett og ett variabelsett, for å så utgi resultatet i en liste av samme dimensjon som inngående liste.

Figur 18: List.Map noden



Flatten

Flatten noden «flater ut» nivåene i en liste. Enkelte ganger kan trenger ikke verdiene i en liste å være oppført under forskjellige nivåer i listene. Da kan flatten-noden brukes til å flate ut disse nivåene, slik at man får mindre nivåer, eller kun ett.

Figur 19: Flatten noden

Custom Node

Til sist er Custom Node. Custom Node er lik hovedskriptene på mange vis, men har noen vesentlige forskjeller. Custom Nodes kan lages og importeres inn i hovedskripter, men ikke via versa. I tillegg kan custom noder legges inn i andre custom noder. Utfra hvilke funksjoner, inputs og outputs en custom node inneholder, har den mulighet til å kobles opp mot eksisterende noder i hovedskriptet. Skriptene blir lagret som .dyn-filer, mens custom nodene blir lagret som .dyf-filer. Når custom nodene blir lagret, kan de legges til i node-biblioteket for rask tilgang. En annen forskjell mellom custom noder og hovedskripter er at skriptene kan kjøres (utføre skriptene), mens nodene ikke kan det. En enkel måte å se hvilken man jobber i er å se på bakgrunnsfargen; I hovedskriptet er bakgrunnsfargen *hvit*, mens i custom noden er den *gul*. Når et hovedskript inneholder custom noder kan de raskt åpnes ved å dobbeltklikke på dem.

3.2.2.2 Optimo

Optimo-pakken er den første tredjeparts-pakken for Dynamo som blir omtalt, og inneholder noder som brukes i en optimaliserings-funksjon. Optimaliseringsfunksjonen som tas i bruk her er NSGA II (Non-dominated Sorting Genetic Algorithm), som enkelt sagt vil generere flere generasjoner med variabelsett og måle opp resultatene mot eksisterende generasjoner. Funksjonen starter med å generere en første generasjon med variabelsett (hvert sett med tilfeldige variabler innenfor brukersatte grenser). Videre vil funksjonen lage nye generasjoner, og sammenligne de nye generasjonene mot de gamle i et forsøk på å finne den sterkeste av dem. Dette kan nærmest ses på som evolusjonsteori, hvor "only the strongest survive". Med dette menes det at når funksjonen genererer nye generasjoner med variabelsett, vil den regne ut resultatet av disse variablene gjennom kontroll-funksjoner (som i denne oppgaven vil være dagslysberegninger og materialbruk), og så sammenligne resultatet med tidligere generasjoner. Funksjonen vil så forkaste den generasjonen med svakest resultat, for og så generere en ny generasjon for å sammenligne med vinneren.

Når funksjonen genererer nye generasjoner bruker den også verdiene til de eksisterende generasjonene, men muterer disse til å lage nye og se etter hvordan den kan forbedre tidligere generasjoner. Når generasjoner genereres brukes noen evolusjonsbegreper: Først blir de variabelsettene med best resultat tatt med videre (1. Selection), så blir variabler som begge variabelsett har til felles brukt i neste generasjon som genereres (2. Cross-over). For andre variabelsett kan (3. Mutation) bli brukt, hvor et variabelsett brukes til å generere helt nye verdier. På denne måten forhindres optimaliseringsfunksjonen fra å sette seg fast i et lokalt-minimum, istedenfor å finne absolutt minimum (lavest mulig). På denne måten vil hver generasjon strekke seg mot bedre resultater og, gitt mange nok generasjoner, muligheten til å finne det beste resultatet.

For å utføre denne optimaliseringen trengs input satt av brukeren. Disse er "Population", som er antallet variabelsett som skal genereres per generasjon. For best resultat bør det være mange (helst over 100) variabelsett per generasjon, siden dette gir funksjonen flere variabler å evaluere utfra, slik at de nye generasjonene vil gi bedre resultater.

Bruker må også velge antall "Objectives", som er antall mål for optimaliseringen. I denne oppgaven er det to mål som skal optimaliseres; Mest mulig dagslys og minst mulig materialbruk.

Bruker må også sette "Lower- og Upper Limits". Dette er begrensingene for variablene, hvor funksjonen vil generere tilfeldige variabler innenfor de satte grensene. For å lage flere variabler i variabelsettene kan disse begrensingene legges inn som *lister* i Dynamo.

Med denne informasjonen vil funksjonen generere den første generasjonen. Variabelsettene i denne generasjonen er nå klar til å kjøres gjennom "Fitness" funksjonene. Antallet fitnessfunksjoner må samsvare med antallet "Objectives". Fitnessfunksjonene bruker variablene som input, og beregner resultater for hvert mål. I denne oppgaven for eksempel vil de beregne dagslysmengde og materialbruk. Disse resultatene blir så koblet opp mot hvert variabelsett i generasjonen.

Nå har funksjonen laget ferdig den første generasjonen som den vil bruke som grunnlag videre.

Videre i funksjonen vil bruker måtte velge "Iteration Number". Dette er antallet med generasjoner som skal genereres. Jo flere generasjoner som blir generert, jo nærmere vil sluttresultatet være til optimalet. Dermed vil det også være sikrest med høyt antall her (helst 50 eller mer).

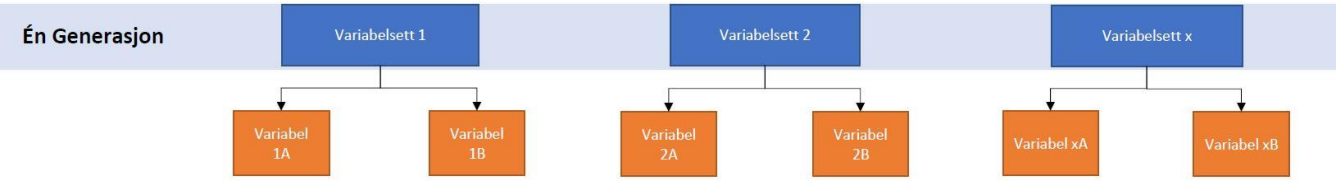
Når disse innstillingene er satt vil programmet automatisk generere nye generasjoner, og måle opp resultatene fra fitnessfunksjonene mot hverandre. Til slutt vil funksjonen stå igjen med de beste resultatene den har funnet. Disse kan presenteres som en Pareto-front ved å legge til noen ekstra noder.

Potensielle komplikasjoner med denne metoden er maskinytelsen ved høyt antall "Population" og "Iteration Number", kombinert med avanserte fitnessfunksjoner. Dette kan kreve mye maskinkraft og tid. Ved simple fitnessfunksjoner vil det være mulig å bruke høyt antall "Population" og "Iteration Number".

3.2.2.2.1 NSGA_II Funksjon

Non-dominated Sorting Genetic Algorithm II er en flermåls (brukt i Optimo) som vil bli brukt i denne oppgaven. Når vi sier flermål mener vi motstridene mål. Non-dominated betyr; Av de endelige løsningene vil alle være like bra. Mer presist kan dette forklares som en løsning (x) kan ikke være bedre enn en annen (y) for ett mål, uten at (x) er dårligere enn (y) for det andre målet (Calle, 2017).

Først må brukeren bestemme hvor mange variabelsett som skal testes om gangen, i tillegg til grenser for disse variablene. Brukeren bestemmer også antall generasjoner som funksjonen skal generere nye variabler for.

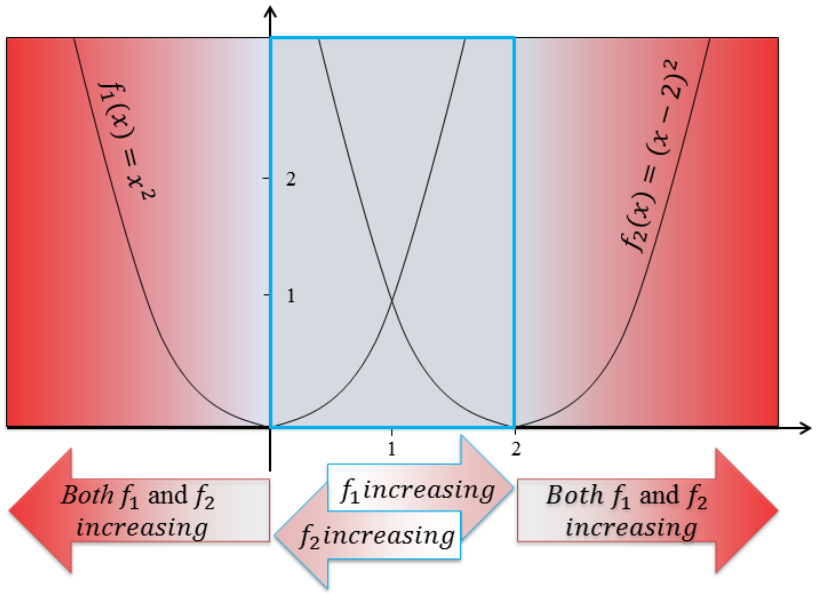


Figur 20: Eksempel på variabelsett fra en generasjon, hvor hvert variabelsett har to variabler

Resultatene for slik flermåls optimalisering kan så presenteres i en Pareto-front.

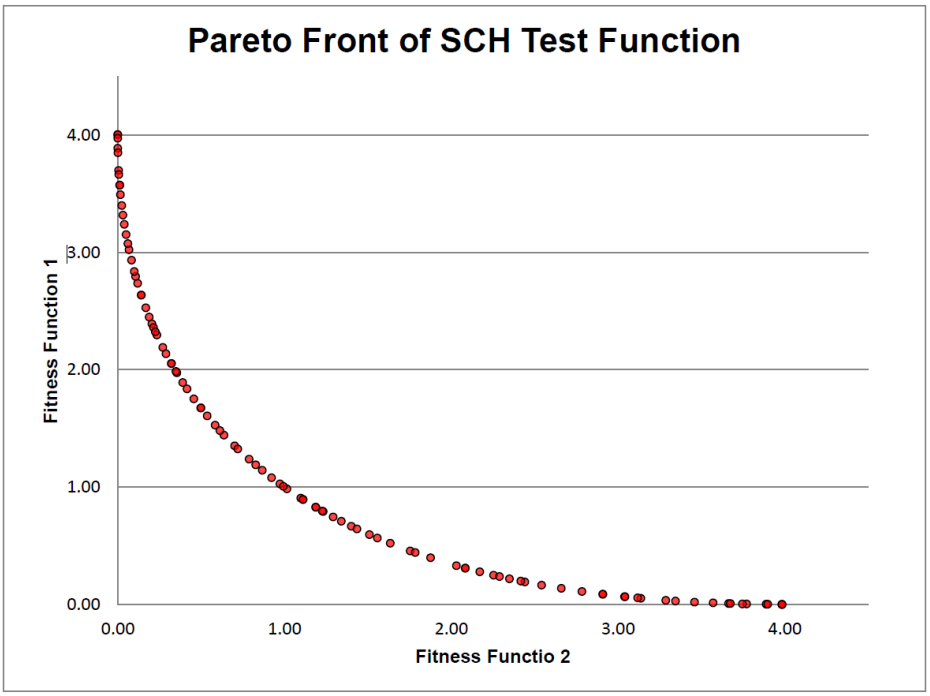
Det neste eksempelet er ment å forklare flermåls optimalisering og Pareto fronten:

Målet her er å minimere variabelen (x) som blir brukt i fitnessfunksjonene x^2 og $(x-2)^2$. Dette er Schaffer funksjon nr. 1 (SCH N.1), som blir brukt til å teste optimaliseringsmetoder.



Figur 21: De to funksjonene som dette eksempelet baserer seg på. (dynamobim.org)

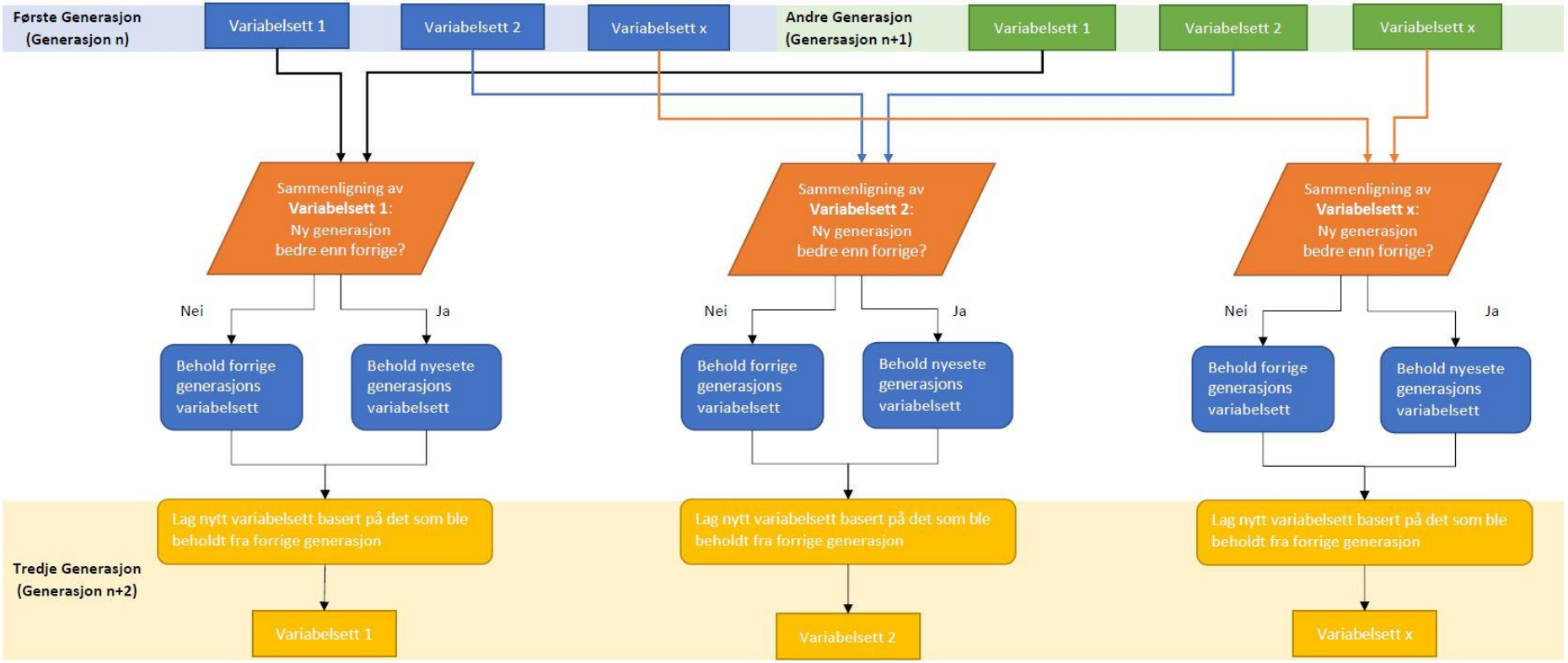
Når x er lavere enn 0 eller større enn 2 vil begge funksjonene stige, dermed ligger de optimale mellom 0 og 2. Denne funksjonen inneholder altså to mål (fitnessfunksjoner): x^2 og $(x-2)^2$, som funksjonen vil prøve å minimere. Innenfor intervallet {0,2} er i alle løsningene like gode, men brukeren kan selv velge variabelsett utfra hvilket av målene som er mest verdsatt.



Figur 22: Pareto-front av funksjonen (dynamobim.org)

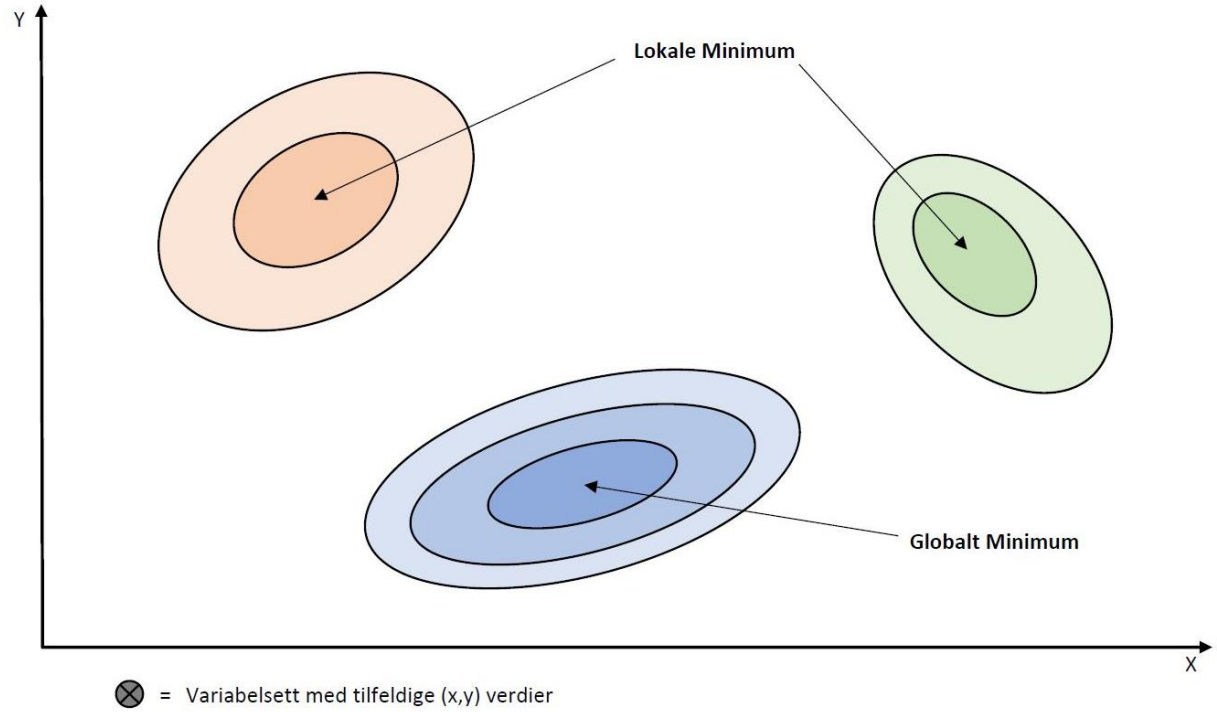
Over ser vi de mest optimale resultatene som ble kalkulert av funksjonen, og vi ser at når fitnessfunksjon 1 er 0 vil fitnessfunksjon 2 være 4, og via versa. Ingen av løsningene er dominerende, men brukeren kan utfra dette diagrammet velge ut en løsning som er nøytral (når x =1) eller veier mer mot ett av de satte målene (fitnessfunksjonene).

For å nå se hvordan funksjonen kommer frem til målene vil vi se nærmere på «genetic»-delen av algoritmen. Denne algoritmen baserer seg på evolusjonsteorien, hvor «only the strongest survive». For denne algoritmen betyr det at den vil generere tilfeldige variabler, som den bruker til å kalkulere resultatene for målene. Etter dette vil den evaluere, og generere nye tilfeldige variabler basert på resultatene fra første gruppe. Disse nye variablene vil så brukes til å kalkulere resultatet for målene. Etter dette vil variablene med resultatene fra første og andre generasjon bli evaluert opp mot hverandre. Her vil de beste (i.h.t. alle satte målene) bli behold, mens de dårligste vil bli forkastet. Neste sett med tilfeldige variabler vil bli basert på de som «vant» forrige runde. På denne måten vil denne funksjonen lete seg frem til de mest optimale variablene med tanke på de satte målene. Dette er som sagt kun *Genetic* delen av algoritmen.



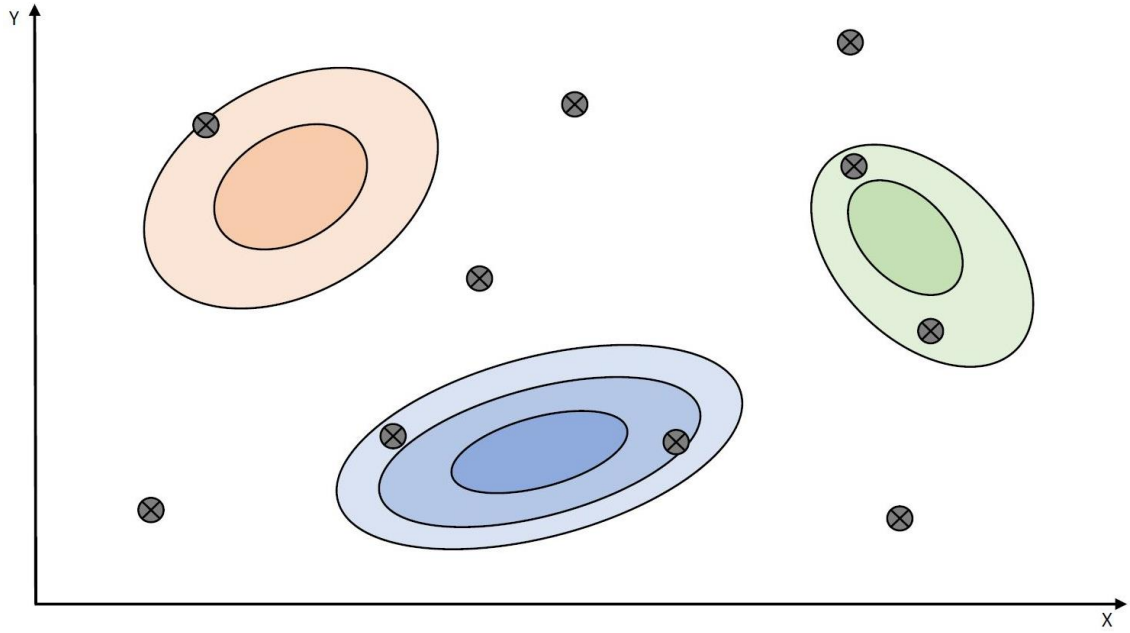
Figur 23: Flowchart som viser hvordan Genetic Algoritmen lager nye generasjoner

Under er et eksempel på en genetic algoritme, hvor funksjonen skal finne absolutt minste z-verdi basert på x- og y verdiene som variabler. Funksjonen starter med å lage et sett med tilfeldige x- og y variabler, og regner ut z-verdien for hver.



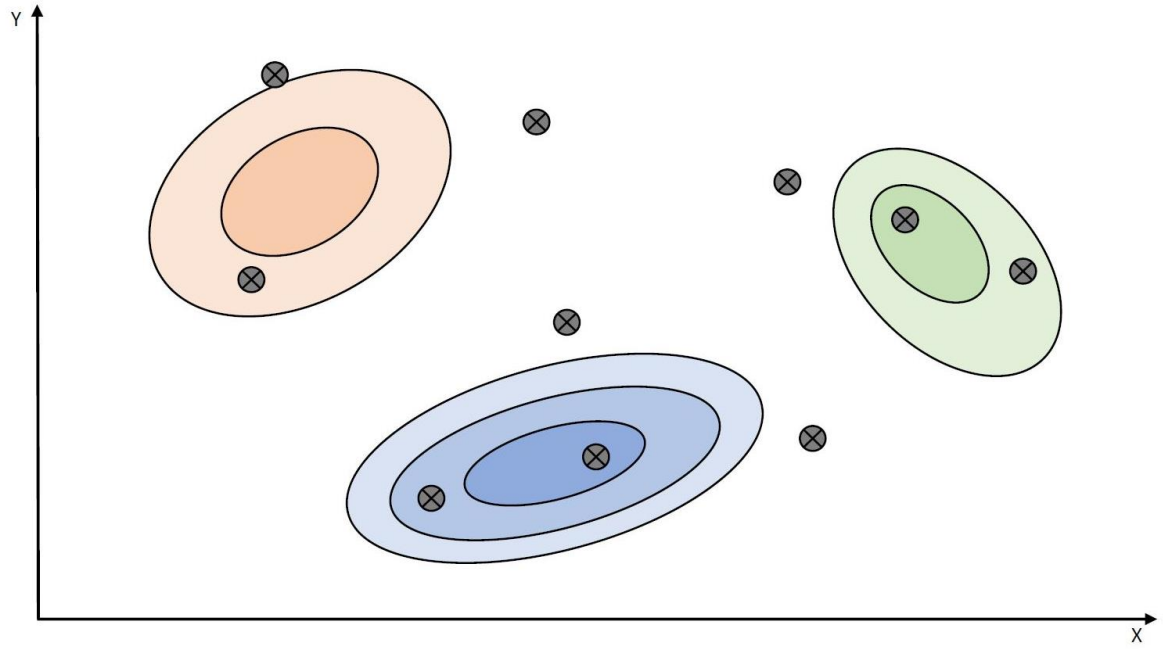
Figur 24: Forklaring av eksempel for genetic algoritme

Dette diagrammet inneholder tre lokale minima, hvor et av dem i tillegg er det globale minimum (blå).



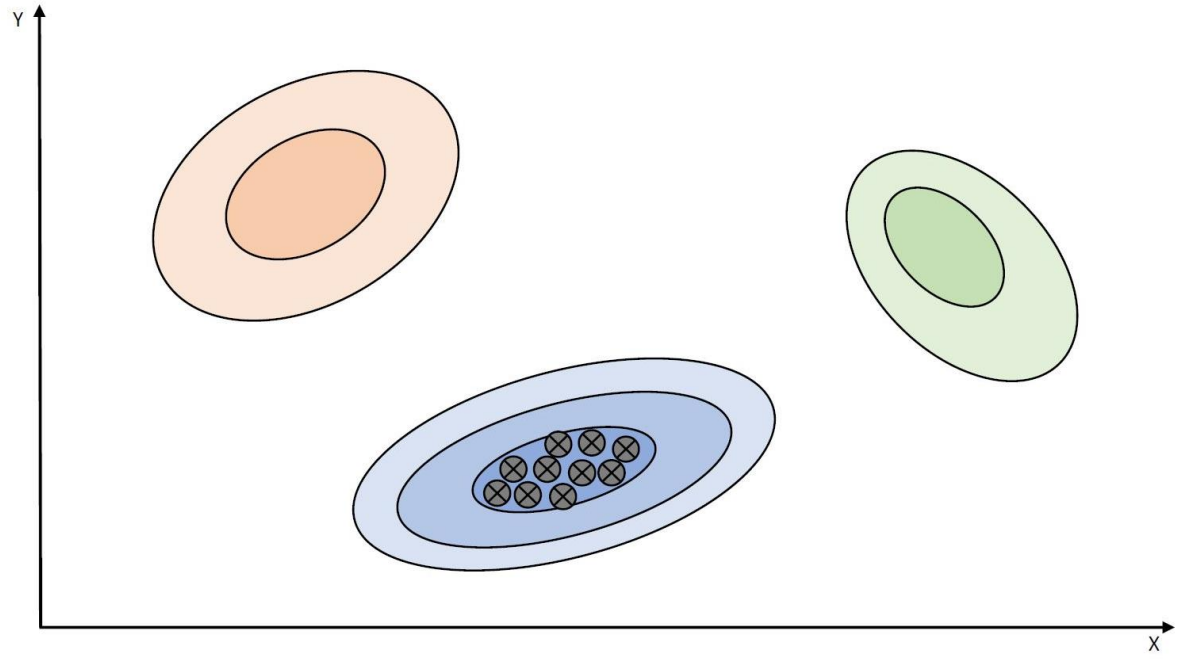
Figur 25: Første steg i genetic algoritme eksempelet

Først lages ti tilfeldige sett med x- og y variabler. Z-verdien for disse blir så regnet ut, som blir brukt til å lage neste generasjon med variabelsett.



Figur 26: Andre steg i genetic algoritme eksempelet

Denne generasjonen er har nå begynt å sikte seg inn på de forskjellige minima, og neste generasjon blir å så være nærmere enn forrige.



Figur 27: Tredje og siste steg i genetic algoritme eksempelet

Gitt stor nok populasjon (og flaks) vil hvert fall *ett* av variabelsettet ha funnet det globale minima tidligere. Etter hvert som variabelsettet sikter seg inn på de forskjellige minima vil til slutt det globale minima gi lavere z verdi enn de lokale. Dermed vil funksjonen gå bort fra disse, for å fortsette å lete etter det aller laveste minimum. *Merk* at dette kun er et eksempel for en genetic algoritme, som kun er en del av NSGA II-funksjonen. Dette er fremgangsmetoden som funksjonen benytter for å finne det optimale resultatet.

3.2.2.3 Ladybug

Ladybug-pakken inneholder noder som blir brukt til å konstruere lokasjon- og værdata, som videre blir brukt av nodene i Honeybee-pakken til å utføre dagslysberegningene.

I tillegg inneholder den noder som brukes til å presentere resultatet fra dagslysberegningene i Dynamo. Denne presentasjonen er i form av et varmekart, som bruker fargekoder i et rutenett for rommet til å gjengi belysningsstyrke for hver rute i rommet.

3.2.2.4 Honeybee

Honeybee-pakken inneholder noder som brukes til å utføre diverse dagslysberegninger. Den gir også muligheten til å legge til refleksjon verdier til Revit-geometri for mer detaljerte beregninger. For å utføre analysen må Revit-modellen ha rominndeling (rommet kan tagges ved å bruke "Room" kommandoen under "Architechure"-fanen i Revit). Dette rommet vil bli delt opp i et rutenett i brukersatte dimensjoner, hvor beregningen vil beregne daglysmengde (lux) for hver enkelt rute. Vinduer, dører og vegger registreres automatisk av Revit når man legger til rom-taggen, men de kan også legges til som Honeybee dører og -vinduer for å manipulere G-faktoren (lysmengde som slippes gjennom) deres. Dørene må lages som vindusfamilier, siden Honeybee ikke registrer dører som lysgjennomslippende (selv om det er en balkongdør som for det meste består av glass).

Dagslysberegningen blir gjort i Radiance (se kapittel 3.2.3). Et selvstendig program som må installeres induviduelt på maskinen. En av Honeybee-nodene ("Run Radiance") aktiverer programmet gjennom cmd (Windows kommando program), hvor Radiance utfører beregningene og sender resultatene tilbake til Dynamo. Radiance utfører kun beregning for ett variabelsett om gangen, og må aktiveres for hvert enkelt variabelsett som skal beregnes.

Denne beregningen vil være en fitnessfunksjon som blir brukt av Optimo. Dette betyr at Optimo vil mate inn ett og ett variabelsett for å få et resultat ut fra hver av dem. Dette resultatet blir videre brukt til å verdsette hvert variabelsett i Optimo.

3.2.2.5 Archilab

Archilab inneholder noder som brukes til å presentere resultatet fra daglysberegningene i Revit, og kombineres med Ladybug-pakken for å gjøre dette. Presentasjonen er, i likhet med presentasjonen i Dynamo av Ladybug-pakken, presentert som et rutebasert varmekart. Hvor hver rute fra beregningen er fremstilt i farger basert på hvor mye lux som treffer ruten.

3.2.3 Radiance (v5.2.0)

Radiance er et dagslysberegningsprogram utviklet av Greg Ward i 1985 (Wikipedia, 2018). Det brukes av Honeybee-pakken for å utføre daglysberegningene i Dynamo, og må installeres separat fra Dynamo. For at Honeybee-pakken skal kunne aktivere Radiance *må* programvaren være installert i "C:\radiance". Om Radiance ikke er installert i denne mappen vil ikke Honeybee klare å finne programmet, og videre føre til feilmelding i Dynamo.

Radiance er validert som simuleringsverktøy for dagslysberegninger i.h.t. CIE 171:2006 (Danish Building Research Institute, 2013). Dette betyr at beregninger utført av Dynamo som bruker Radiance er validert for å dokumentere krav til dagslys for boliger.

Radiance benytter seg av såkalte «Raytracing» kalkulasjoner for dagslys, hvor vektorer brukes til å beregne lysstyrken gjennom rommet. Det er også vel å merke seg at Radiance bruker en stokastisk beregningsmetode (radiance-online, 2016), som betyr at to resultater av samme variabler ikke nødvendigvis vil være identiske. Forskjellen mellom disse resultatene er ikke forventet å være over 2%, slik at det fortsatt anses som brukbar metode for å utføre beregningene.

3.2.4 Programmeringsspråk

Først, generelt om programmeringsspråk; Programmeringsspråk brukes til å kommunisere med datamaskiner for å styre dem. Disse språkene er bygget opp av algoritmer som programvarer og datamaskiner igjen er bygget opp av. Alle programmer er bygget opp av programmeringsspråk, men de fleste programmer gjømmer disse språkene og lar brukerne bruke grensesnittet til programmene for å styre dem. Når vi bruker datamaskiner bruker vi programmeringsspråkene selv om vi kanskje ikke er klar over det. Når brukeren kun bruker knapper og kommandoer ved å klikke på ikonene på skjermen bruker de programmeringsspråkene, hvor hvert ikon sender kommandoer i form av algoritmer gjennom disse programmeringsspråkene i bakgrunnen.

Det finnes mange diverse programmeringsspråk der ute, og i denne oppgaven kommer vi over to: C# (C-sharp) og Python.

C# er et programmeringsspråk utviklet av Microsoft (Wikipedia, 2018), som brukes i en rekke moderne programvarer (bl.a. Autodesk Revit).

Python er et programmeringsspråk utviklet av Python Software Foundation (Wikipedia, 2018). I likhet med C# brukes dette språket også til å utvikle programmer, men brukes også direkte i programvarer (i form av algoritmer) for å utføre spesifikke funksjoner i programmer.

Selv om disse begge er programmeringsspråk har de begge veldig vesentlige forskjeller, kapasiteter, styrker og svakheter. Begge har forskjellige anvending- og bruksmetoder. I denne oppgaven blir begge blitt undersøkt for å forbedre utførelsen av oppgaven, men grunnet begrenset kompetanse innenfor programmering har dette ikke vært undersøkt til det fulleste.

Som tidligere nevnt er Revit sin API (programmets grensesnitt) bygget opp med C#, men både C# og Python algoritmer kan anvendes i Dynamo. Selv om begge det er mulig å bruke begge i Dynamo er bruken deres veldig forskjellig. Python kan brukes direkte i Dynamo, ved å legge til "Python"-noder, kan algoritmene legges direkte inn i skriptene. Samtidig er samtlige eksisterende noder i Dynamo bygget opp med C#, men for å legge til nye C# algoritmer i Dynamo må disse lages eksternt og importeres inn. C# algoritmene kan lages bl.a. lages i Microsoft Visual Code Studio (se kapittel 3.2.5).

I denne oppgaven brukes C# i nodene fra Optimo-pakken, mens Python brukes i Ladybug- og Honeybee-pakkene. Algoritmene i disse nodene er klare til bruk, men vil bli undersøkt for å se etter forbedrings potensial i algoritmene.

3.2.5 Microsoft Visual Studio Code

Microsoft Visual Code Studio brukes til å lage og endre algoritmer. Disse algoritmene kan være bygget i diverse programmeringsspråk (bl.a. C# og Python). I denne oppgaven vil dette programmet brukes til å undersøke algoritmene i Dynamo-pakkene, for å se etter eventuelle feil eller forbedringspotensialer. For å se C#-algoritmene brukes dette Microsoft Visual Code Studio (med tilleggspakker som tillater Visual Code Studio å lese C#). C#-algoritmene som eventuelt lages i dette programmet kan lagres som .dll filer, som videre kan bli importert i Dynamo.

3.2.6 Andre programmer som kunne blitt brukt

Dette kapittelet går gjennom andre potensielle programmer som kunne blitt brukt til å løse oppgaven. Disse har ikke blitt fokusert på i denne oppgaven på grunn av begrenset kompetanse i forhold til programmene nevnt tidligere i dokumentet. Med tanke på dette ses det som mest effektivt å videreutvikle eksisterende kompetanse innen programmer som jeg allerede kan, i motsetning til å lære meg disse programmene fra bunnen av. Denne beslutningen er tatt med tanke på tidsrammen jeg har til å utføre oppgaven.

3.2.6.1 Rhinoceros 3D

Rhinoceros er et modelleringsprogram på lik linje med Revit. Selv om begge programmene har deres likheter, har de fortsatt forskjellige styrker og svakheter. På grunn av manglende kompetanse innen denne programvaren kan jeg ikke gå for mye inn i detalj angående disse forskjellene.

3.2.6.2 Grasshopper for Rhinoceros

Grasshopper brukes på lik linje som visual programming verktøy for Rhinoceros, slik Dynamo brukes for Revit. En spesifikk funksjon som kommer med Grasshopper som standard er «Galapagos»-funksjonen. Dette er en optimaliserings funksjon som fungerer likt som Optimo-funksjonen for Dynamo, men siden dette er en del av grunnprogramvaren kan det antas å ha en mer stabil utførelse.

3.2.6.3 Autodesk Insight 360 for Revit

Autodesk Insight 360 for Revit er Autodesk sitt eget optimaliseringsverktøy. Det er cloud-basert som gir god ytelse til å utføre simuleringene, men gir i gjengjeld ikke god mulighet til kommunikasjon mellom simuleringsverktøy, geometri og variabler. Dette gjør muligheten til å automatisere prosessen og kartlegge alle alternative design vanskeligere.

3.2.6.4 Matlab

Matlab er et skripting-program som brukes til å lage og utføre algoritmer. Det bruker tekstbaserte algoritmer, og klarer også å lese de fleste programmeringsspråk. Manglende funksjoner til å modellere geometri er hovedårsaken til at dette programmet ikke ble tatt i bruk.

4. Parametrisk design som beslutningsstøtte

Dette kapittelet vil gjennomgå hvordan jeg løser oppgaven, ved anvending av teori og programmer som ble gjennomgått i tidligere kapitler.

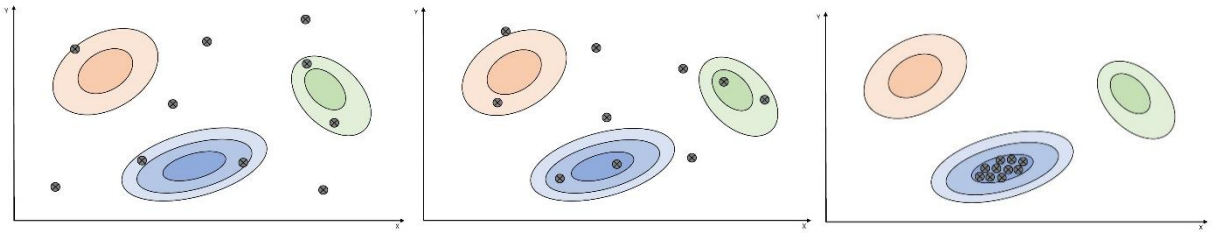
Målet er å finne en metode som kan identifisere de mest optimale kombinasjonene av input utfra ønsket output. Dette vil si at parametrisk design brukes til å undersøke alle mulige alternativer, og presentere de mest optimale slik at brukeren kan ta beslutningen for hvilket design som vil bli brukt i prosjektet.

4.1 Generelt

Optimo-pakken i Dynamo vil spille en av hovedrollene for hvordan parametrisk design kan brukes som beslutningsstøtte. Som nevnt i kapittel 3.2.2.2 må brukeren sette verdier "Population", "Objectives", "Lower Limits", "Upper Limits" og "Iteration Number", i tillegg til å lage en funksjon for hvert "Objective". Inputene til disse funksjonene vil være tilfeldige variabler, generert av Optimo innenfor de brukersatte "Lower Limits" og "Upper Limits". Optimo vil prøve å minimere resultatene for fitness funksjonene, og videre basere sin forkast/behold beslutning på disse resultatene. Ved å ha to funksjoner med motstridende mål vil Optimo lete etter et kompromiss når den simulerer gjennom forskjellige variabler.

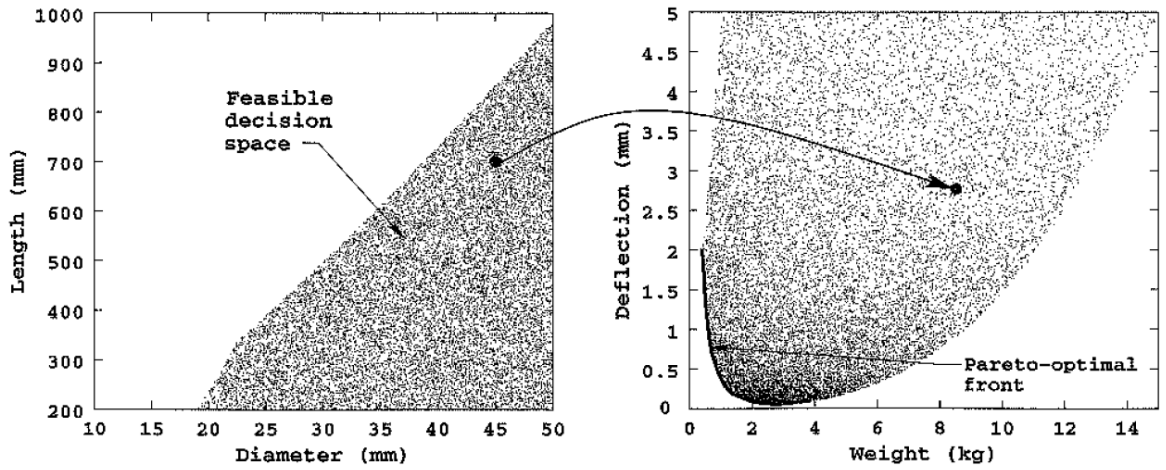
Fitnessfunksjonene inneholder formler som beregner et resultat ut fra inputene. For eksempel vil en fitnessfunksjon som beregner et areal inneholde to inputs; lengde-x og lengde-y, og formelen $x \times y$. Optimo vil så basere beslutningen sin over å beholde eller forkaste variabler på dette resultatet (arealet). Dette gjør den ved å gå etter variabler som gir minst mulig resultat for fitnessfunksjonen (minst mulig areal). Om man ønsker at Optimo skal maksimere et resultat multipliseres resultatet med «-1» inne i funksjonen, slik at Optimo vil søke etter «høyeste» verdi i negativ skala.

Når disse fitness funksjonene er laget, samt alle brukersatte verdier for Optimo er gitt vil Optimo simulere gjennom tilfeldige variabelsett. Gitt nok "Population" (Antall variabelsett som testes om gangen) og "Iteration Number" (Antall generasjoner med variabel sett som vil bli generert), i tillegg til nok maskinkraft og tid, vil Optimo kunne simulere seg fra til de beste løsningene.



Figur 28: For hver generasjon vil funksjonen komme nærmere det optimale målet. Globalt minimum i dette eksempelet

Etter at simuleringene er ferdig kan Optimo fremstille resultatene i en Pareto-front (kapittel 3.2.2.2.1), hvor de beste løsningene er fremstilt i et diagram. I dette diagrammet vil alle løsningene vil alle alternativene være like gode, men med forskjellige kompromiss i.h.t. målene som er satt. Dette er en oversiktlig måte å fremstille resultatene, og tillater arkitekt å kunne velge ut ønskede løsning utfra hvilket kompromiss som er å foretrekke.

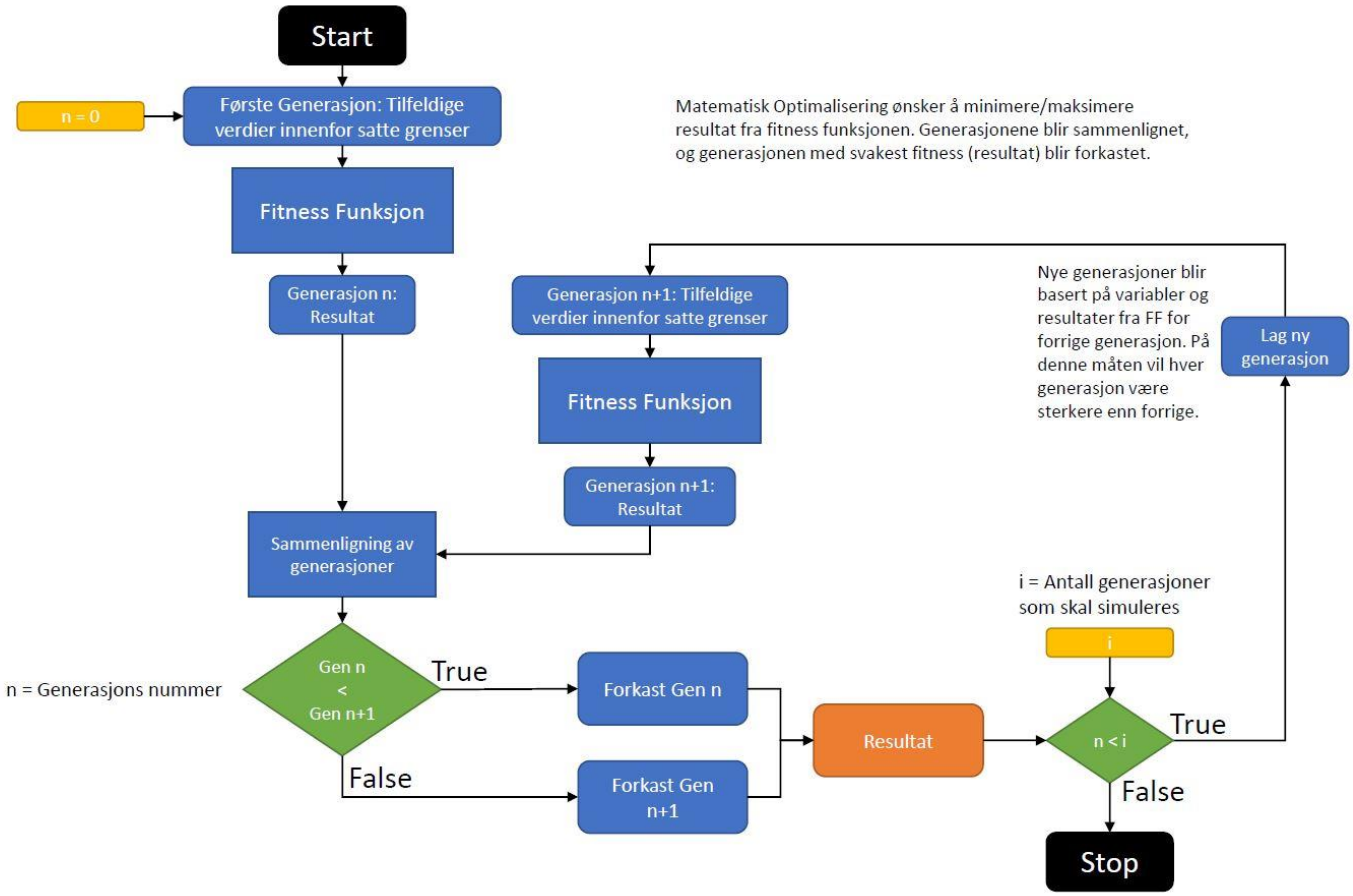


Figur 29: Eksempel på fremstilling av brukbare resultater (venstre) som Pareto-Front (Høyre) (oklahomaanalytics.com).

I figuren over ser vi at det finnes utallige alternativer, men kun de langs Pareto-Fronten er optimale.

Dette er tanken over hvordan parametrisk design kan brukes som beslutningsstøtte, hvor man bruker maskiner til å simulere gjennom mulige løsninger for og så presentere de beste.

Optimo-pakken i Dynamo vil utføre denne optimaliseringen:



Figur 30: Flowchart for Optimo-noden

4.2 Parametrisk design som beslutningsstøtte for dagslysberegninger

Hvordan skal jeg utføre oppgaven?

Jeg lager en BIM-modell av en konseptbolig. Denne bruker jeg til å gjengi geometrien av boligen til Dynamo for å utføre dagslysberegningene.

Jeg vil sette opp skripter i Dynamo som gjør både dagslysberegninger og optimalisering av resultatene. For å lage disse skriptene vil jeg bruke tredjeparts-"pakker" i Dynamo; Optimo, Honeybee, Ladybug og Archilab. *Vel å merke seg* at Dynamo operer i *meter* som standard, mens Revit operer i *millimeter* som standard. Av denne grunnen må én av dem endres, og i denne oppgaven blir Revits enheter endret til *meter*. Dette blir gjort gjennom «Project Units»-kommandoen under «Manage»-fanen i Revit.

Ladybug og Honeybee vil brukes til å utføre dagslysberegningene. Disse beregninger blir brukt som fitnessfunksjon i Optimo, men jeg må multiplisere resultatet med «-1» for at Optimo skal prøve å *maksimere* lysmengden. I tillegg vil jeg lage en fitnessfunksjon for å minimere materialbruken. I denne funksjonen bruker jeg refleksjonsverdier (ubehandlet overflate antas å ha lavere refleksjonsverdi enn behandlet i denne oppgaven) og dimensjoner, hvor Optimo vil minimere resultatet fra funksjonen.

Optimo vil så simulere gjennom kombinasjonene for å finne de mest optimale alternativene utfra variablenes satte grenser.

Optimo bruker *Pareto-front* til å fremstille de beste alternativene slik at man fortsatt kan velge utfra preferanser.

Ønskede alternativer kan så plukkes ut for å bli presentert med «varmekart», ved hjelp av Ladybug- og Archilab-nodene.

Hvordan dette blir laget blir forklart i de neste kapitlene. Merk at koblingene mellom gruppene vil bli listet opp til slutt av skriptgjennomgangene.

Variablene som vil bli brukt i denne oppgaven er følgende:

Navn	**Minimum**	**Maksimum**	**Intervall**
Orientering	-2°	2°	1°
Brystningshøyde	0mm	500mm	100mm
G-faktor	0.4	0.8	0.1
Sideheng Refleksjonsverdi	0.5	0.7	Kun to valg
Innervegger Refleksjonsverdi	0.4	0.7	Kun to valg
Balkongdybde	2000mm	3000mm	100mm

Tabell 1: Variabler som blir brukt i oppgaven.

I første omgang er det ønskelig å inkludere alle disse variablene, *men* enkelte av disse kan droppes hvis metoden sliter med å simulere alle disse variablene. Til sammen gir disse variablene 6600 forskjellige kombinasjoner.

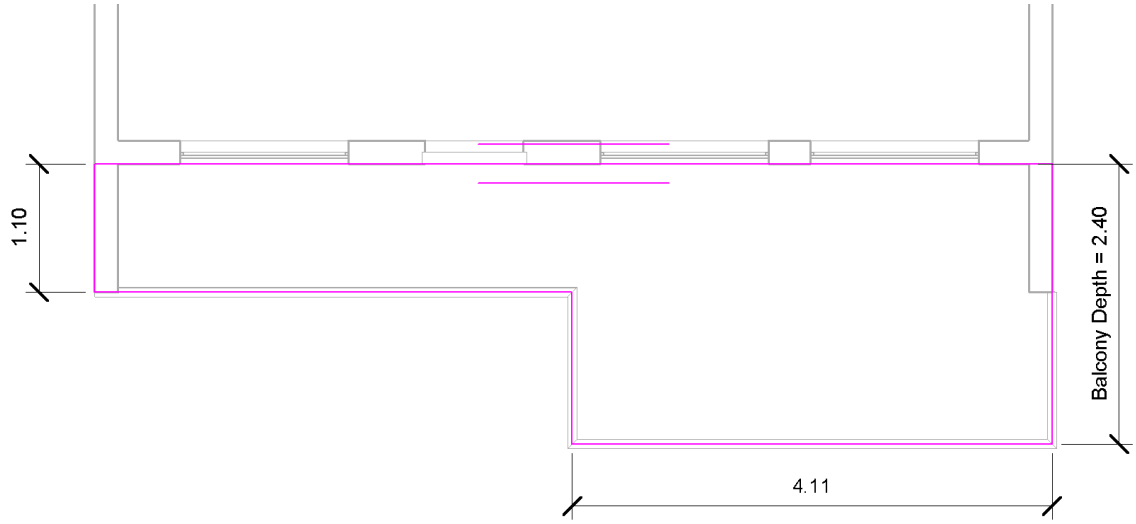
4.2.1. Gjennomgang

For denne oppgaven, med dagslysberegninger og konseptboligene, er det en del ekstra som må inkluderes i forhold til den generelle løsningen.

Revit modell

For å utføre dagslysberegningene trengs geometri og rom fra Revit, og dette blir laget fra bunnen av ved hjelp av tegninger og IFC-filer for konseptbolig A1a i Revit. Når geometrien er laget, er det viktig legge til et rom, ved å bruke kommandoen "Room", under Architechture fanen i Revit. Dette rommet vil bli brukt av Honeybee for dagslysberegningene, og vil føre til feilmelding om det ikke er lagt til.

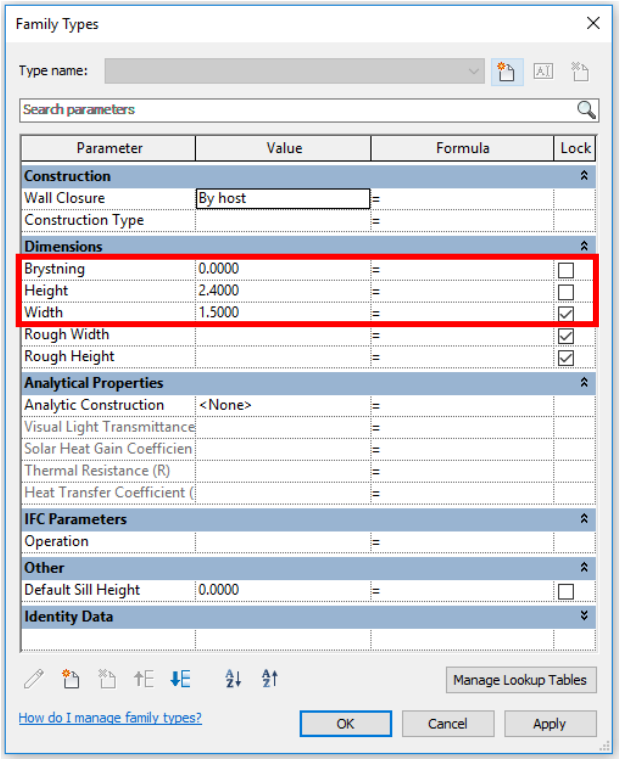
Siden Balkong er variabel, er de viktig å legge en parameter til lengden dens i Revit (Helst i "sketching mode") slik at denne kan kobles til i Dynamo.



Figur 31: Sketch av balkong, med "Balcony Depth"-parameteren

Vindu og balkongdør

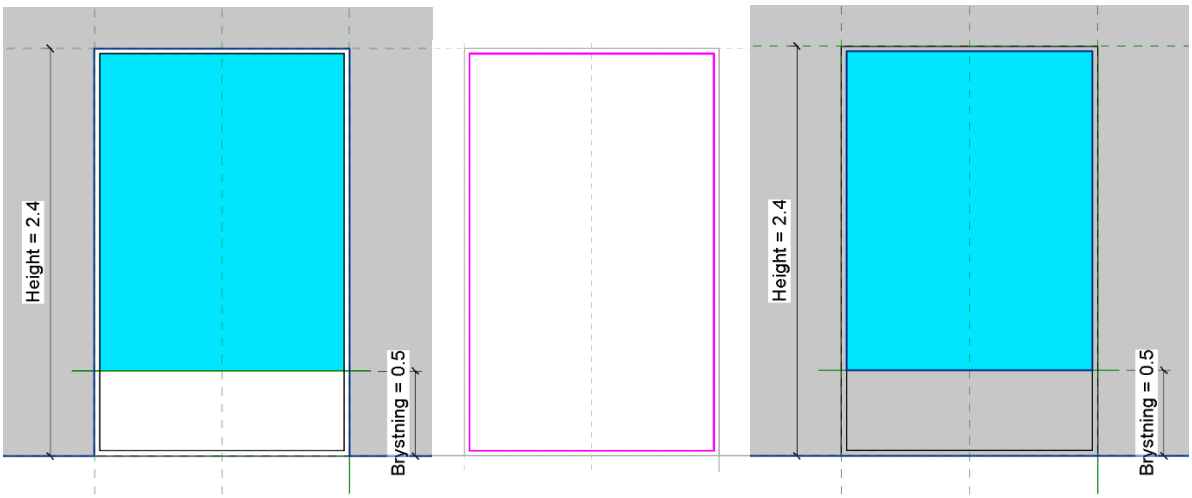
Vinduene og balkongdøren lages også fra bunnen, siden disse trenger spesifikke parametere for å oppføre seg som ønsket. Blant disse parameterne er høyde, bredde slik at disse enkelt kan settes i prosjektet eller er ønsket å koble opp mot Dynamo etter hvert, men i førsteomgang vil disse være konstant. En annen parameter som vil være med er brystningshøyde. Denne vil være konstant for balkongdøren, men variabel for vinduene. Dette vil si at Optimo-pakken i Dynamo vil bruke tilfeldige verdier for denne parameteren, som videre brukes i dagslysberegningene. For å modellere disse familiene brukes «Dimensions»- og «Reference Lines» kommandoene aktivt, for å sikre robuste og sikre modeller. Uten dette kan enkelt elementer ligge uten referanser og føre til feilmeldinger når noen prøver å utføre endringer i modellen.



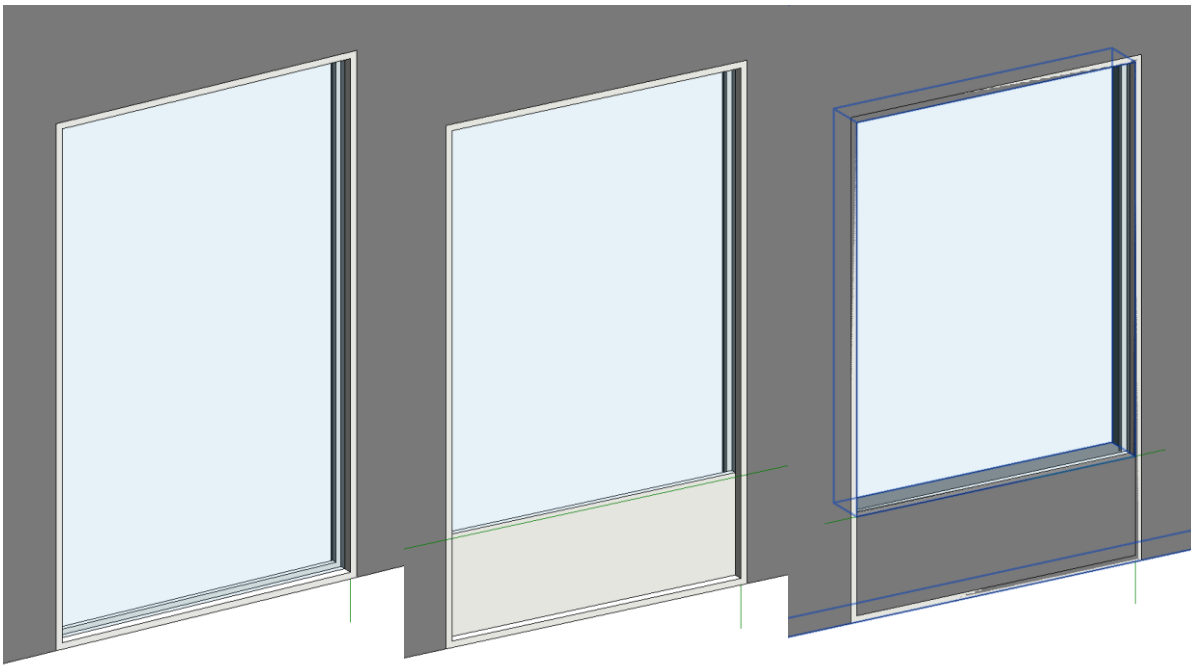
Figur 32: Parameterliste for vindu- og dørfamiliene

Vindu

Først lages vindusfamilien, og dette starter med «generic window family»-templaten i Revit. *Vel å merke seg er at Honeybee (Dagsanalysen) kun bruker «Opening Cut» som vinduet i beregningen, og dette vil si at ingen av geometrien i familien blir brukt.* Dermed vil all geometri lagt til kun være estetisk i Revit-modellen. For en mer detaljert beregning er vinduskarm inkludert slik at «Opening Cut» må legges på innsiden av karmen. På denne måten vil Honeybee beregne lyset gjennom vinduene som utfra glassarealet.

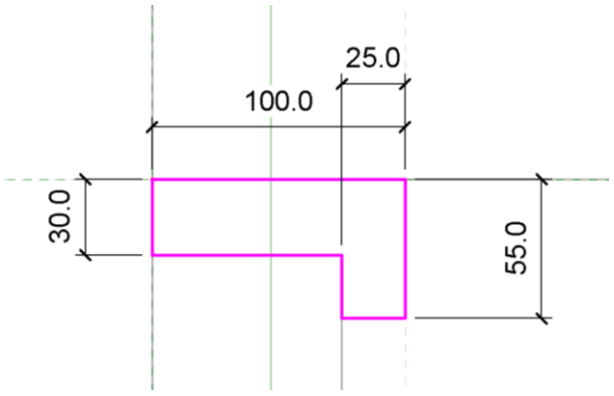


Figur 33: Endring av "Opening Cut" slik at den følger glassarealet. (Høyre) «Opening Cut» følger også glassarealet når vinduet har brystning.

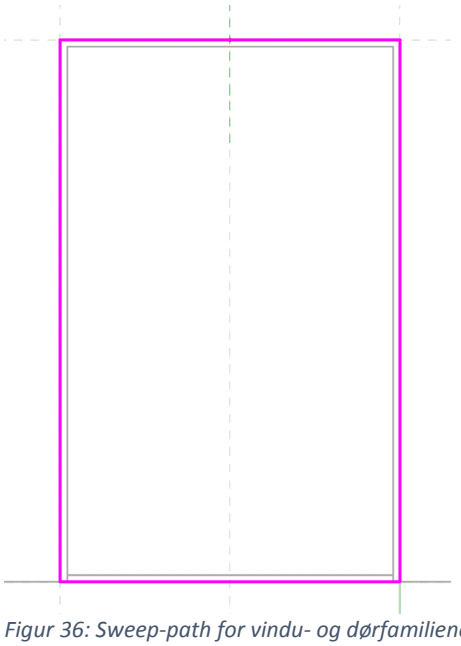


Figur 34: 3D visning av vindu, med brystning (Midten), hvor "Opening Cut" blir lagt rundt glassarealet (Høyre).

For denne oppgaven brukes «Sweep»-kommandoen til å lage en simpel karm. Sweep-kommandoen finnes under «Architecture»-fanen. Denne sweep-geometrien er ikke nødvendig for beregningene, men gir et mer realistisk resultat når «Opening Cut» blir lagt rundt glassarealet siden lyset ellers hadde gått gjennom den virtuelle karmen.



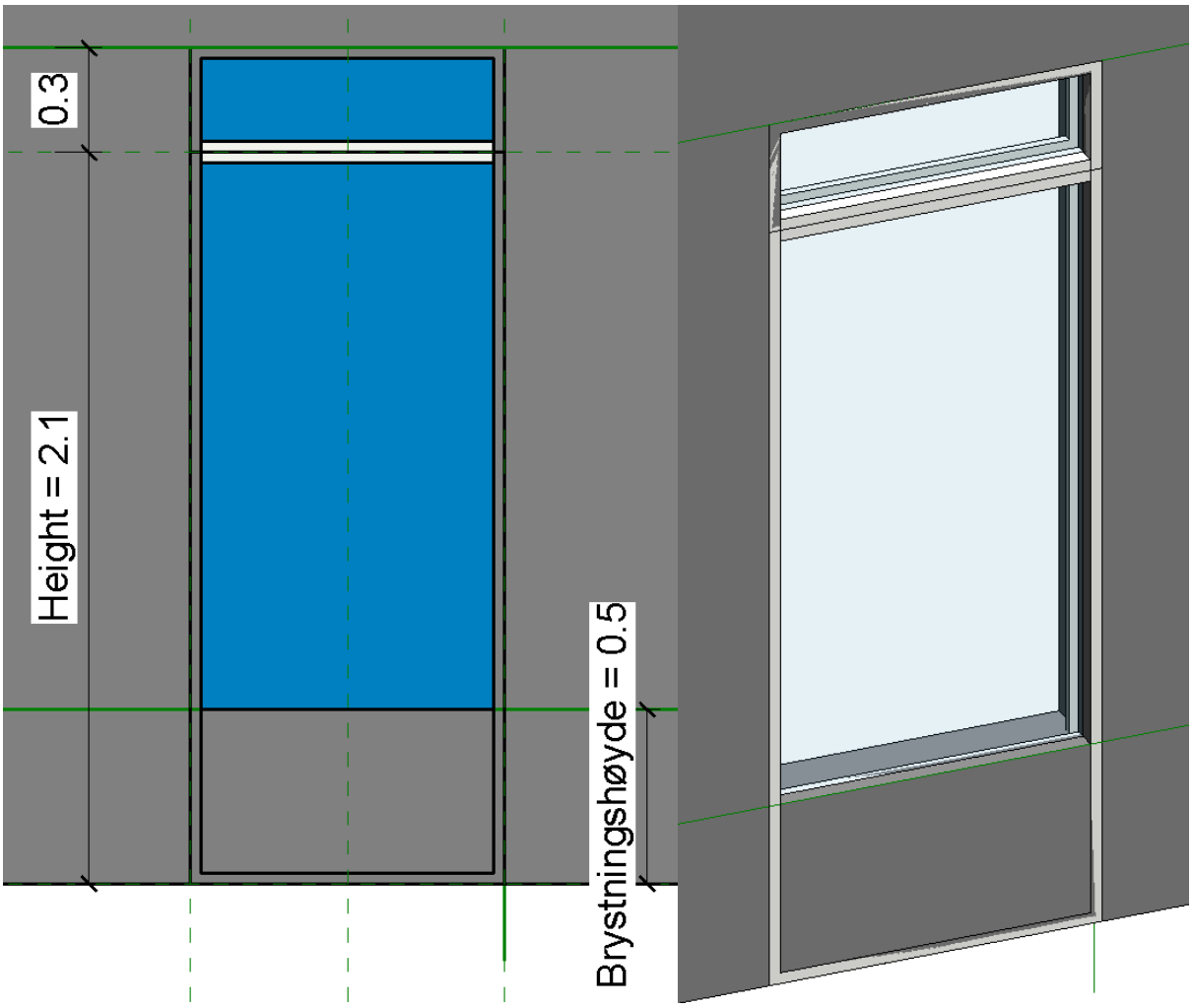
Figur 35: Sketch av Sweep-profilen for vindu- og dørfamiliene



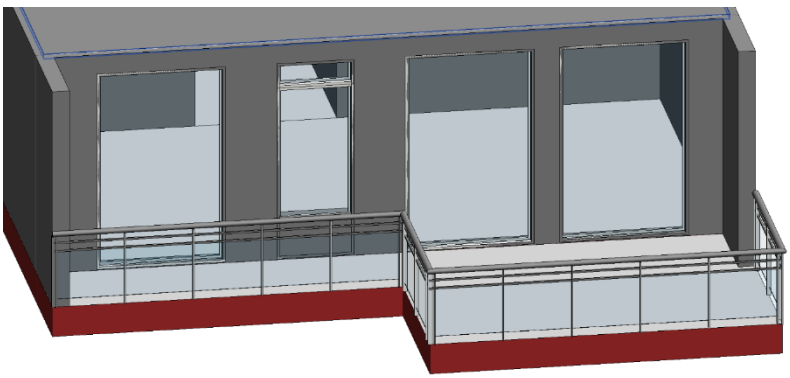
Figur 36: Sweep-path for vindu- og dørfamiliene

Balkongdør

Honeybee regner ikke dør-familier som lysgjennomslippende, og balkongdøren må dermed bli laget som en vindus-familie. Istedenfor å lage denne fra bunnen av kan den tidligere vindusfamilien modifiseres og lagres under et nytt navn. Den mest vesentlige forskjellen fra vinduene er vinduet over, ellers er vindus- og dør familien lik (sett bort fra dimensjonene). Dette vinduet kan bruke samme sweep-profil som resten.



Figur 37: Fasade-view- (Venstre), og 3D view av balkongdøren (Høyre)



Figur 38: Tre vinduer og én balkongdør i fasaden til boligmodulen

Skriptene i Dynamo

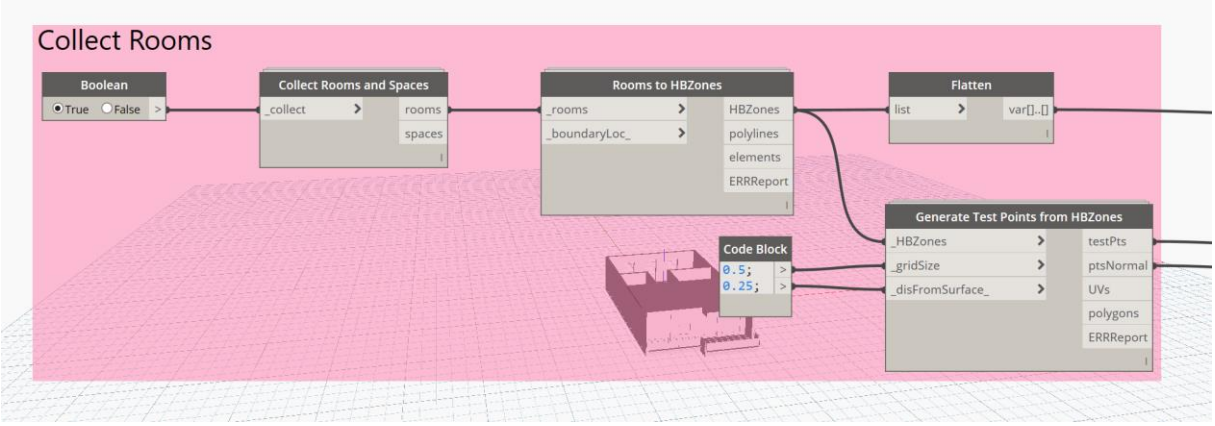
Når Revit modellen er klar er det på tide å starte med Dynamo. Den åpnes i Revit, i «Visual Programming»-kategorien under "Manage" fanen. Det kan være greit å starte med skriptet som skal brukes for dagslysberegninger å lage dette som et eget skript i første omgang (ikke som en custom node, som den vil bli laget som senere), slik at skriptet kan testes underveis.

Når man lager skripter i Dynamo er det mulig å gruppere nodene for å få en oversikt hvor forskjellige funksjoner blir utført. Dette er ikke nødvendig, men være til stor hjelp i store, kompliserte skripter.

Dagslysberegning

Collect Rooms

For å gå gjennom de forskjellige bitene av daglysberegningsskriptene vil jeg gå gjennom hver av disse gruppene, og først, "Collect Rooms"-gruppen:



Figur 39: "Collect Rooms"-gruppen

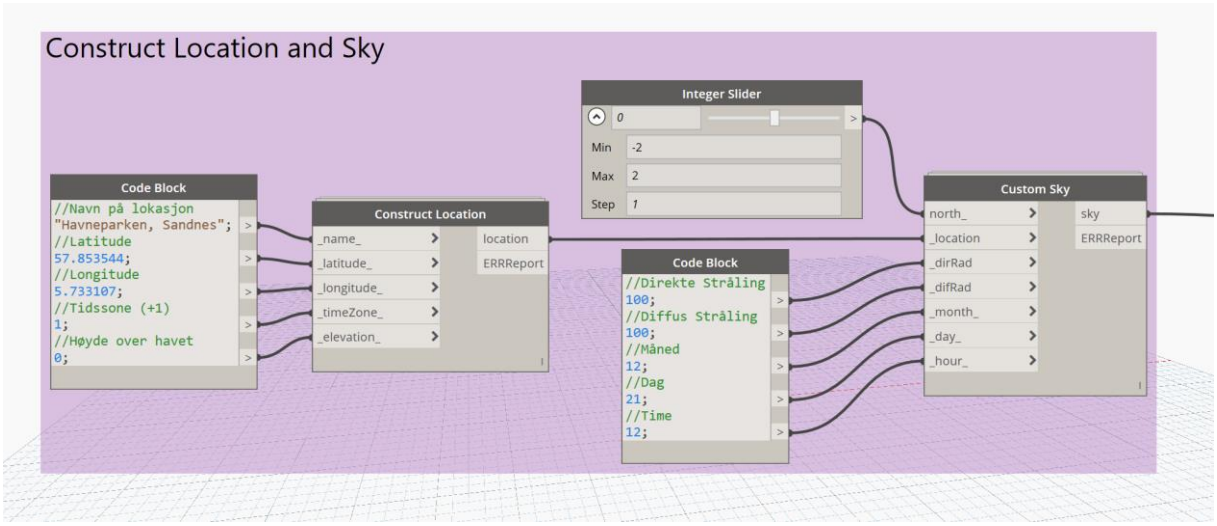
Denne gruppen vil samle inn rommene fra Revit modellen, og konvertere dem til testpunkter som blir brukt til å simulere dagslysmengden.

Denne gruppen starter med en boolean-node som brukes til å aktivere eller deaktivere funksjonen for å samle rommene. Når den er aktiv vil funksjonen hente ut rommene, med mål, i "Collect Rooms and Spaces"-noden fra Honeybee. Videre vil de bli konvertert til såkalte "Honeybee Zones", som Honeybee funksjonen trenger for å utføre beregningene. Disse sonene blir så delt opp i et rutenett ved hjelp av "Generate Test Points from HBzones"-noden, hvor brukeren kan sette inn ønskede dimensjoner. Disse dimensjonene er størrelsen på rutene (Gridsize), og høyde over gulv (disFromSurface). Dette gjøres ved hjelp av en code block-node

Videre vil resultatene fra disse nodene bli brukt i beregningsgruppen.

Construct Sky

Før vi kan gå videre til beregningsgruppen er det flere innstillinger som må settes for en detaljert simulering. En av disse gruppene er "Construct Location and Sky":



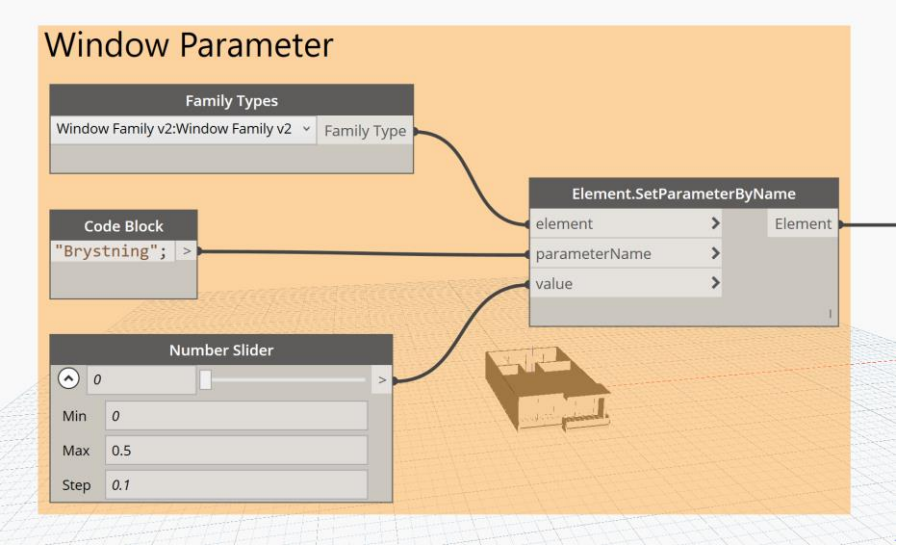
Figur 40: "Construct Location and Sky"-gruppen

Til dette brukes Ladybug nodene "Construct Location" og "Custom Sky". Disse trenger brukerinput som blir anvendt ved hjelp av code block-noder. I "Contruct Location"-noden settes ønskede koordinater inn, i tillegg til navn.

I "Custom Sky"-noden settes strålingsdata og årstid inn, slik at analysen vil basere seg på disse forholdene. Location vil i tillegg være koblet til denne noden, slik at output fra denne noden vil gi beregningsfunksjonen et oversiktlig bilde over hvilket solforhold som blir beregnet. *Fant selv ikke noen gode kilder for hvilken Direkte- og Diffus stråling, så disse er satt til 100 for denne oppgaven.* I tillegg setter man inn årstid, hvor 21 desember kl. 12 blir brukt i denne oppgaven (Lyseste tidspunkt, på den mørkeste dagen i året). Resultatet fra denne noden blir så koblet videre til beregningsgruppen.

Window Parameter

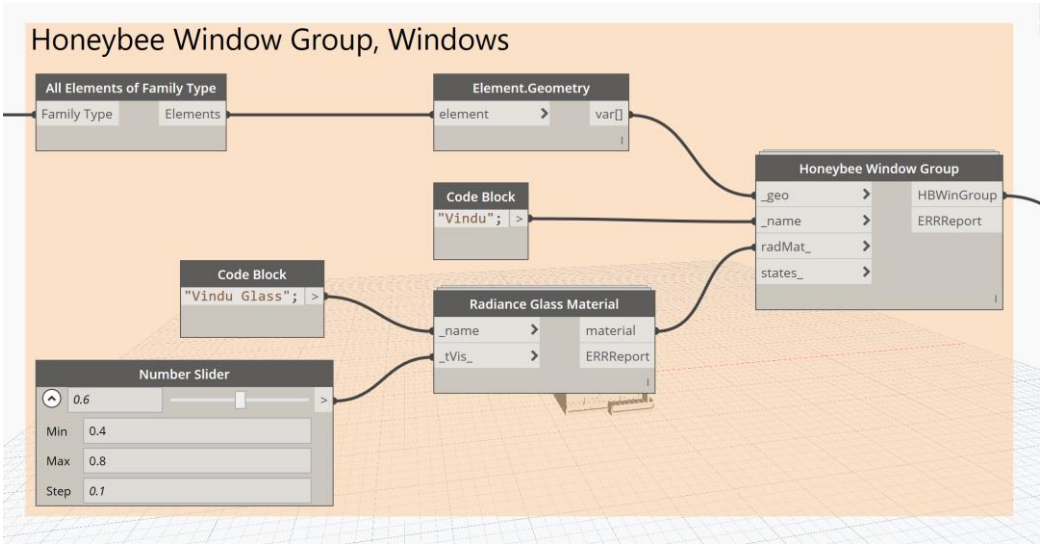
Nå er rom og lokasjons data lagt klart for beregningen, men vi trenger også å legge inn detaljer for vinduene (mer konkret glassene). I tillegg må parameterne som styrer brystning til vinduene må legges til, slik at de kan bli inkludert i beregningene. Først er gruppen "Window Parameters": Vi starter med å hente vindusfamilien fra Revit modellen med "Family Types"-noden. Så brukes "Element.SetParameterByName"-noden for å gi muligheten til å endre (type-)parameterne til familien. Legg til "Brystning" (Navnet på parameteren) ved hjelp av en "Code Block"-node, og en "Number Slider" som styrer verdien til parameteren.



Figur 41: "Window Parameter"-gruppen

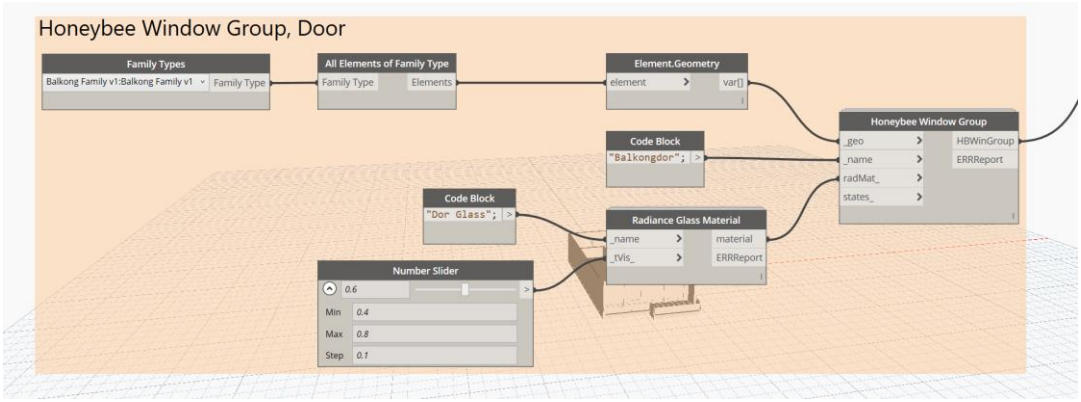
Honeybee Window Group

Videre skal vinduene og balkongdøren konverteres til en "Honeybee Window Group", som brukes i dagslysberegningen. Først må alle elementene av familietypen hentes ut, og dette gjøres med "All Elements of Family Type"-noden. Dette vil identifisere alle tilfellene av denne vindusfamilien i Revit-modellen, slik at disse kan konverteres til Honeybee vinduer. Før de kan konverteres må de gjøres om til geometri i Dynamo, og dette blir gjort med "Element.Geomtry"-noden. Videre trengs to Honeybee-noder for å konvertere vinduene; "Radiance Glass Material" og "Honeybee Window Group". "Radiance Glass Material" tillater brukeren å sette en g-faktor på vinduene, og siden dette er en variabel vil vi koble til en "Number Slider"-node til "_tVis". Nå vil alle inputene som "Honeybee Window Group" trenger være klare og kan kobles til.



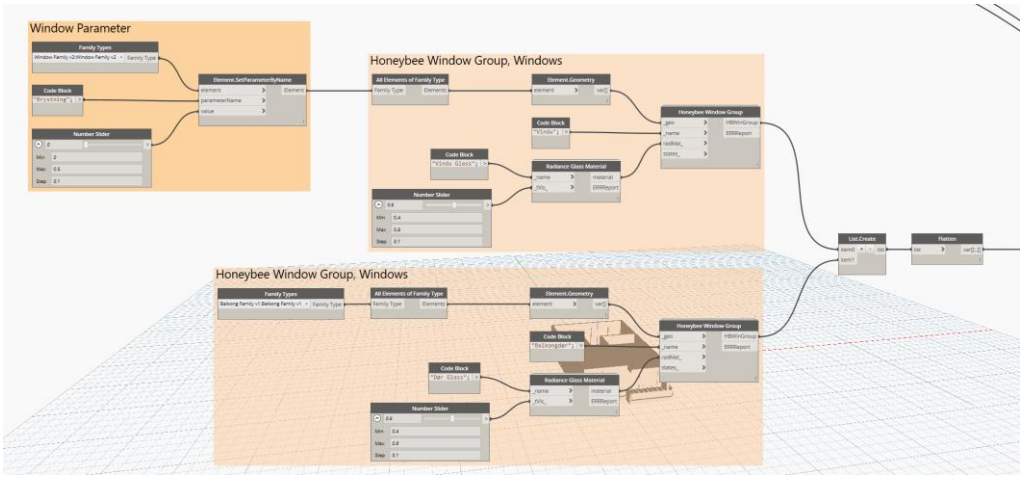
Figur 42: "Honeybee Window Group, Window"-gruppen

I tillegg til denne må også balkongdøren konverteres til en "Honeybee Window Group", og dette gjøres på samme måte som for vinduene, men uten "Window Parameters"-gruppen og riktig familietype valgt.



Figur 43: "Honeybee Window Group, Door"-gruppen

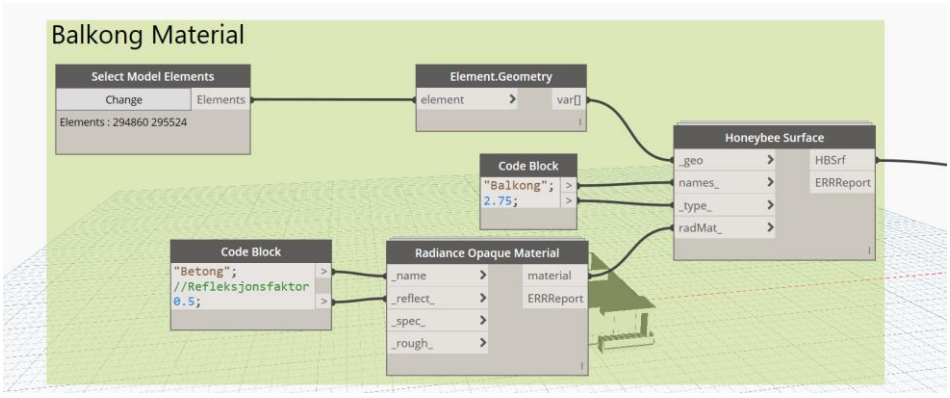
Når disse to gruppene er ferdig, samles resultatene i en liste, som flates ut og kobles opp mot beregningsgruppen.



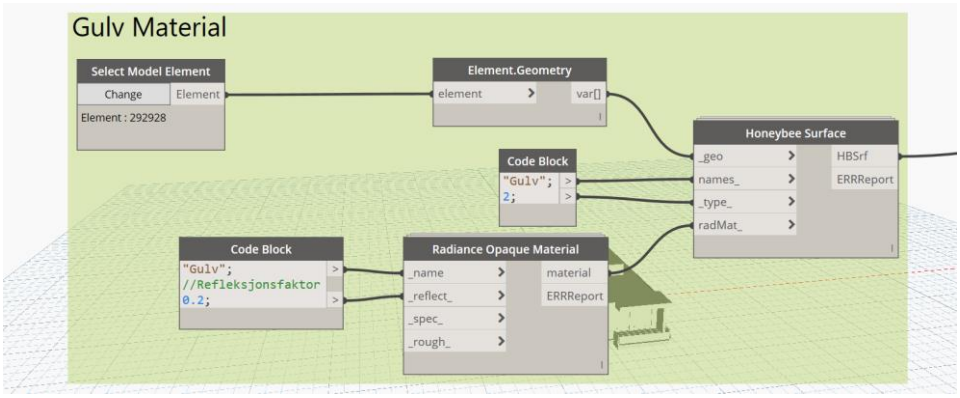
Figur 44: "Honeybee Window Group"-gruppene kombinert til en liste

Radiance Material Scene

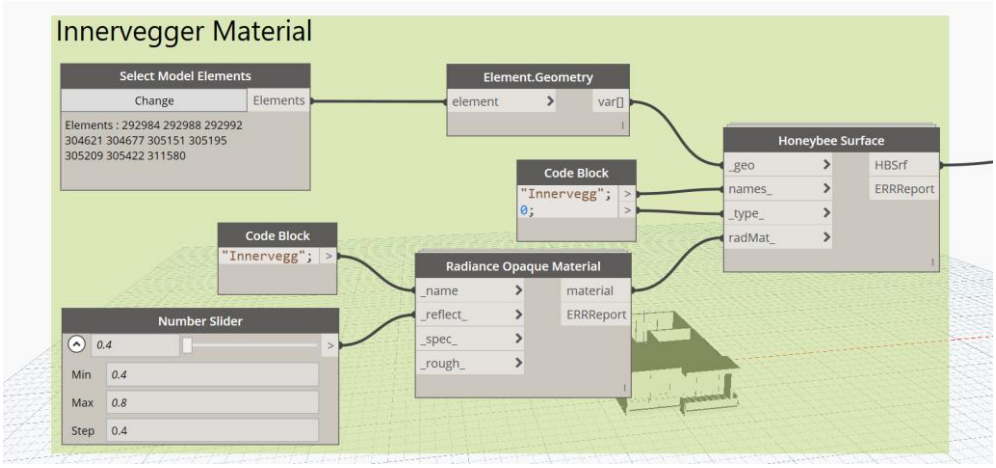
Siste innstillingsgruppe er "Radiance Scene", og brukes for å legge til refleksjonsfaktor til geometrien i prosjektet.



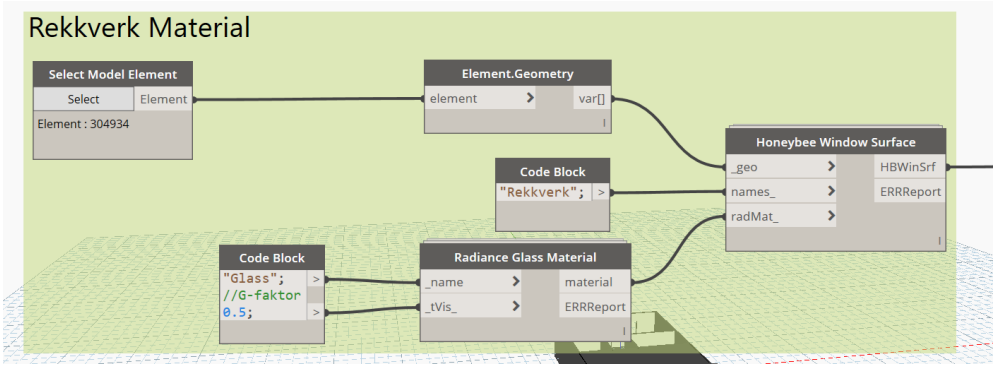
Figur 45: "Balkong Material"-gruppen



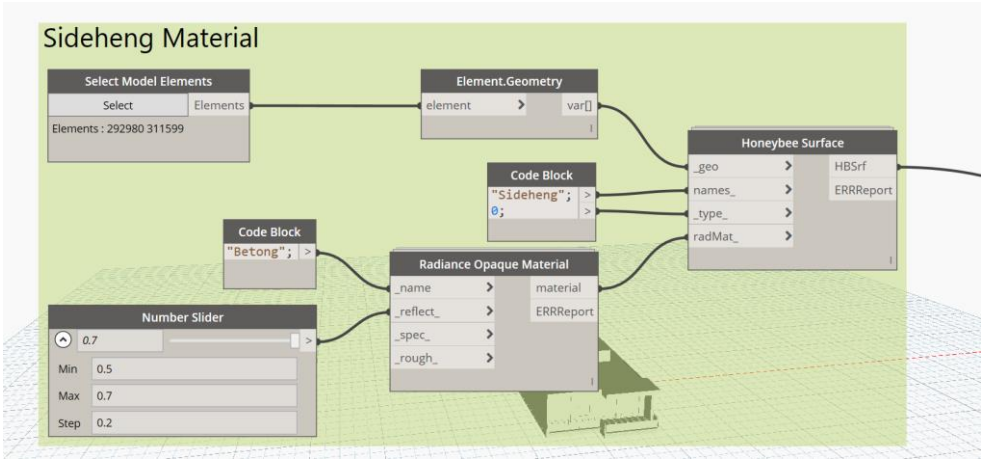
Figur 46: "Gulv Material"-gruppen



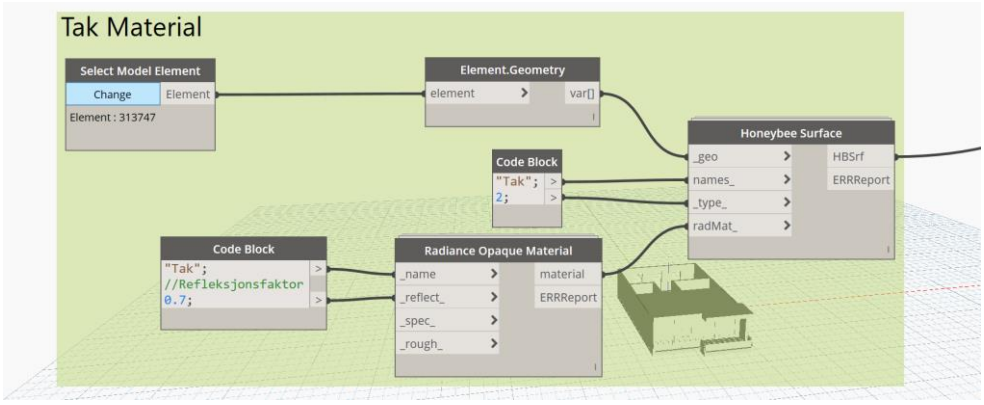
Figur 47: "Innervegger Material"-gruppen



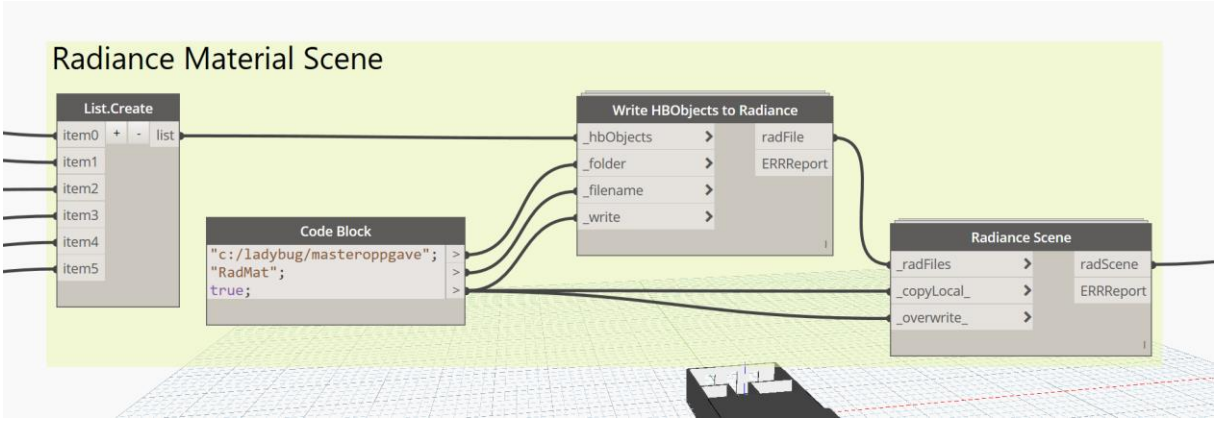
Figur 48: "Rekkverk Material"-gruppen



Figur 49: "Sideheng Material"-gruppen



Figur 50: "Tak Material"-gruppen

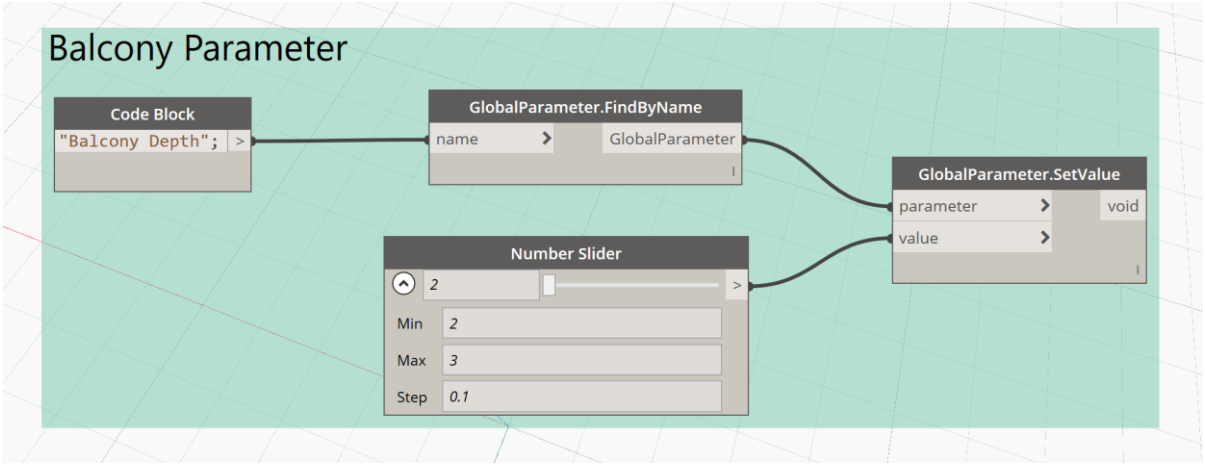


Figur 51: "Radiance Material Scene"-gruppen, hvor alle materialgruppene blir satt sammen

Dette fungerer likt for all geometri, bortsett fra enkeltforskjeller mellom noen av elementene (Se figurer for referanse). Først må geometri-elementet velges, og dette kan gjøres ved å bruke "Select Model Element" («Select Model Element*s*» for flere elementer), for å så klikke "Change" i noden og klikke på ønskede elementer i Revit. Når et element er valgt vil Revit-nummeret deres vises i noden. Videre skal disse elementenes geometri bli gjengitt i Dynamo, og til dette brukes "Element.Geometry"-noden. Videre trenger vi Honeybee nodene "Radiance Opaque Material" og "Honeybee Surface". "Radiance Opaque Material" noden brukes for å legge til materialverdier, blant annet refleksjonsfaktor (_reflect_) som blir satt med en "Number Slider"-node (de variable verdiene). "Radiance Opaque Material"-noden kobles så opp mot "Honeybee Suface"-noden, som nå kan genere denne geometrien med satte verdier. Når disse gruppene er laget for all relevant geometri, kan de samles i en liste, som videre kobles til Honeybee-noden "Write HBObjects to Radiance". Denne noen vil klargjøre geometrien til å brukes av Radiance. Videre kobles denne noden opp mot Honeybees "Radiance Scene"-node, som videre kobles opp mot beregningsgruppen.

Balcony Parameter

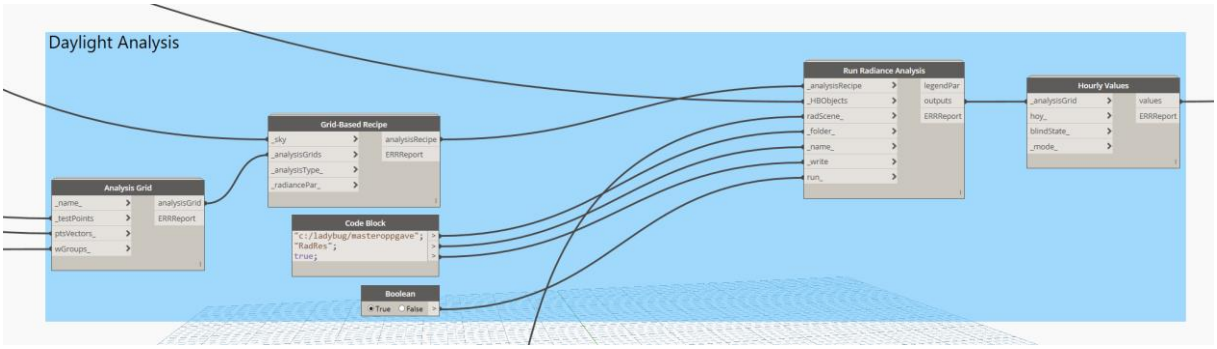
Vi trenger også en gruppe for å styre parameteren som bestemmer dybden til balkongen. Først legger vi til «GlobalParamter.FindByName»-noden, som vi mater med navnet på prosjektparameteren (Balcony Depth). Så legger vi til noden «Global.Parameter.SetValue», som brukes for å sette verdien til parameteren i Revit. For å sette denne parameteren bruker vi en «Number Slider»-node, hvor vi i tillegg kan sette inn ønskede grenser for parameteren.



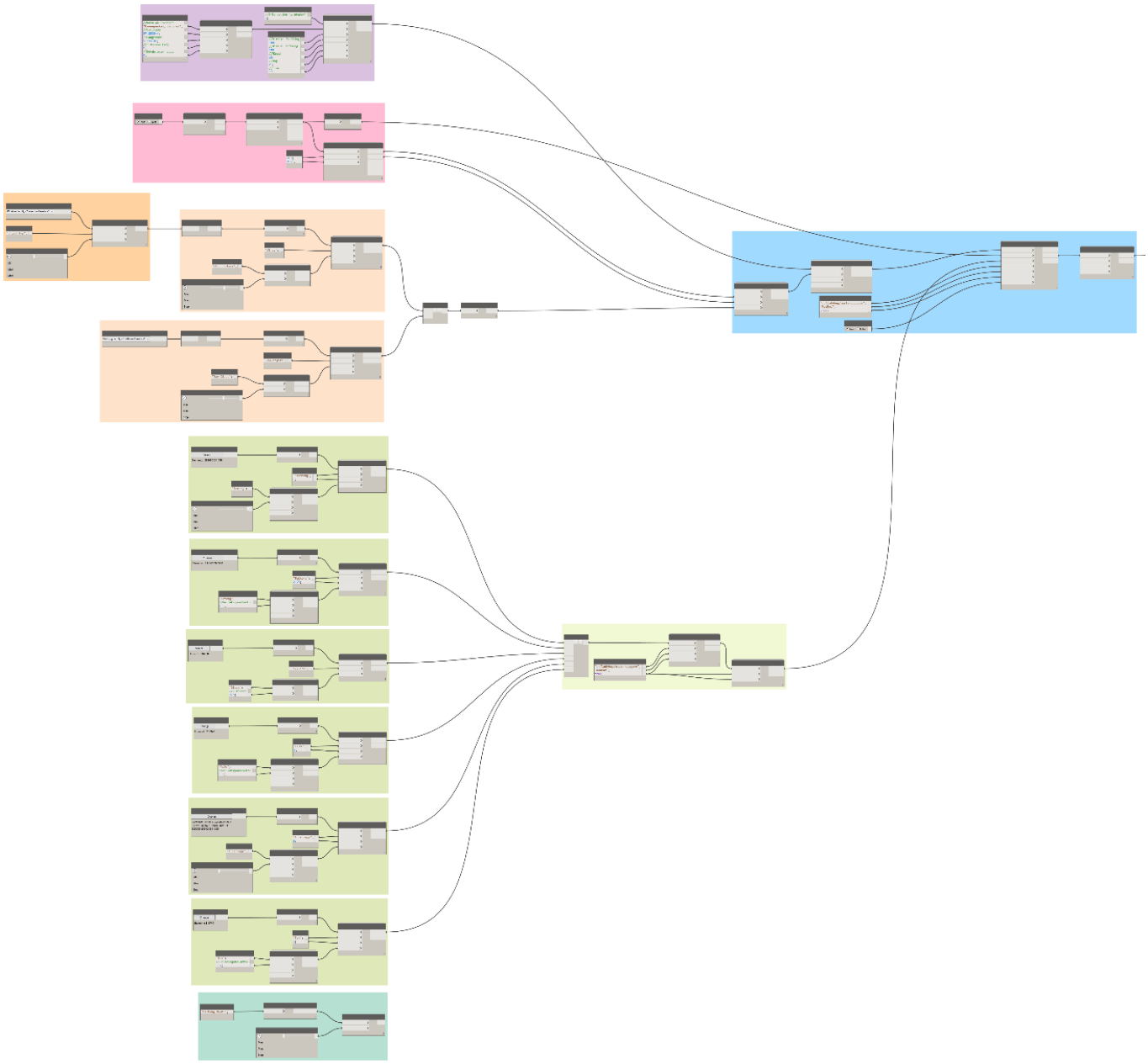
Figur 52: "Balcony Parameter"-gruppen

Daylight Analysis

Nå skal alt ligge klart for beregningene, og dermed flytter vi oss over til "Daylight Analysis"-gruppen. I denne gruppen vil resultatene fra alle tidligere noder bli brukt til å utføre dagslysberegningene ved hjelp av Radiance-programmet. Først trenger vi "Analysis Grid"-noden, som vil kombinere rutenettet og vindusgruppene fra "Room Collecting" og "Honeybee Window Group"-gruppene henholdsvis. Videre legger vi til en "Grid-Based Recipe"-node, som bruker resultatet fra "Create Location and Sky"-gruppen. Videre legger vi til "Run Radiance Analysis"-noden, og det er denne noden vil utføre analysen. Denne noden trenger en boolean-node som brukes til å aktivere og deaktivere noden. Resultatet fra denne noden kobles til en "Hourly Values"-node. Dette gir ut resultatet fra beregningene som Lux-verdi for hver enkelt rute i en liste. Hvordan dette resultatet blir tatt videre til Optimo-skriptet er valgfritt, men i denne oppgaven blir gjennomsnittlig Lux regnet ut og sendt ut som output.



Figur 53: "Daylight Analysis"-gruppen



Figur 54: Full oversikt av daylightanalysis-skriptet

Koblinger mellom gruppene:

Fra			Til		
Gruppe	**Node**	**Output**	**Gruppe**	**Node**	**Input**
Construct Location and Sky	Custom Sky	Sky	Daylight Analysis	Grid Based Recipe	_Sky
Collect Rooms	Flatten (Rooms to HBZones)	var[]..[]	Daylight Analysis	Run Radiance Anylisis	_HBObjects
Collect Rooms	Generate Test Points from HBObjects	testPts	Daylight Analysis	Analysis Grid	_testPoints
Collect Rooms	Generate Test Points from HBObjects	ptsNormal	Daylight Analysis	Analysis Grid	ptsVectors_
Window Parameter	Element. SetParameterByName	Element	Honeybee Window Group, Window	All Elements of Family Type	Family Type
Honeybee Window Group	Flatten (List.Create < Honeybee Window Group)	var[]..[]	Daylight Analysis	Analysis Grid	wGroups_
Alle "… Material"-grupper	Honeybee Surface	HBSrf	Radiance Material Scene	List.Create	item…
Radiance Material Scene	Radiance Scene	radScene	Daylight Analysis	Run Radiance Anylisis	radScene_

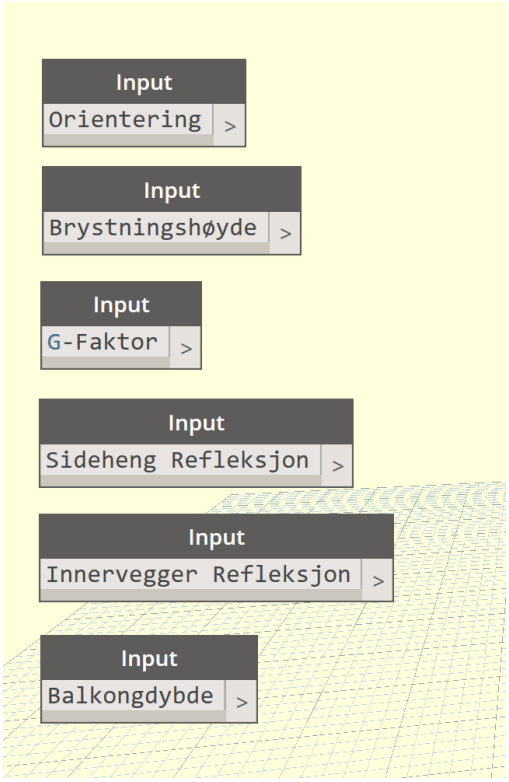
Tabell 2: Koblinger mellom gruppene i dagslysberegningsskriptet.

Oversikt over noder som *ikke* er del av Dynamos grunnleggende bibliotek:

Gruppe	Node	Pakke
Construct Location and Sky	Construct Location	Ladybug
Construct Location and Sky	Custom Sky	Honeybee
Collect Rooms	Collect Rooms and Spaces	Honeybee
Collect Rooms	Rooms to HBZones	Honeybee
Collect Rooms	Generate Test Points from HBZones	Honeybee
Honeybee Window Group	Radiance Glass Material	Honeybee
Honeybee Window Group	Honeybee Window Group	Honeybee
Alle "… Material"-grupper	Radiance Opaque Material	Honeybee
Alle "… Material"-grupper	Honeybee Surface	Honeybee
Radiance Material Scene	Write HBObjects to Radiance	Honeybee
Radiance Material Scene	Radiance Scene	Honeybee

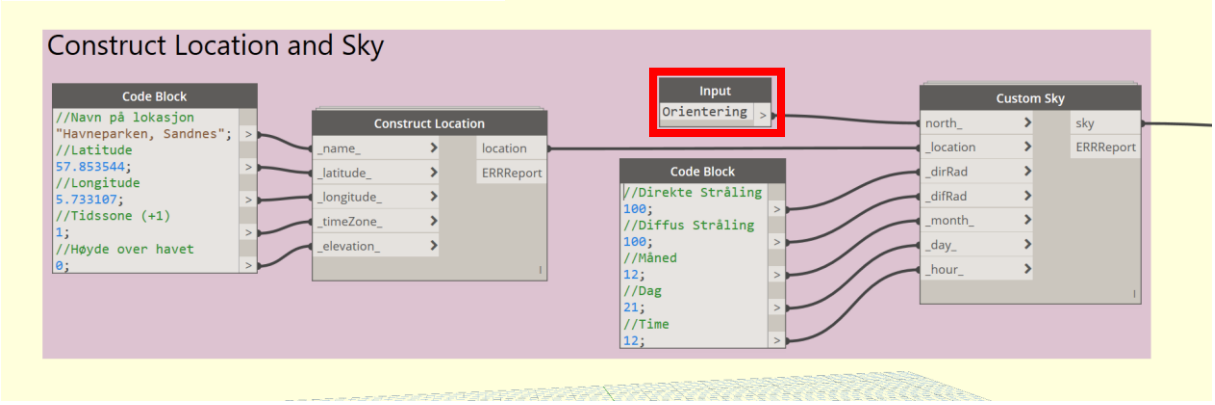
Tabell 3: Oversikt over node-produsent for dagslysberegningsskriptet.

Daglysberegning, fra skript til Custom Node

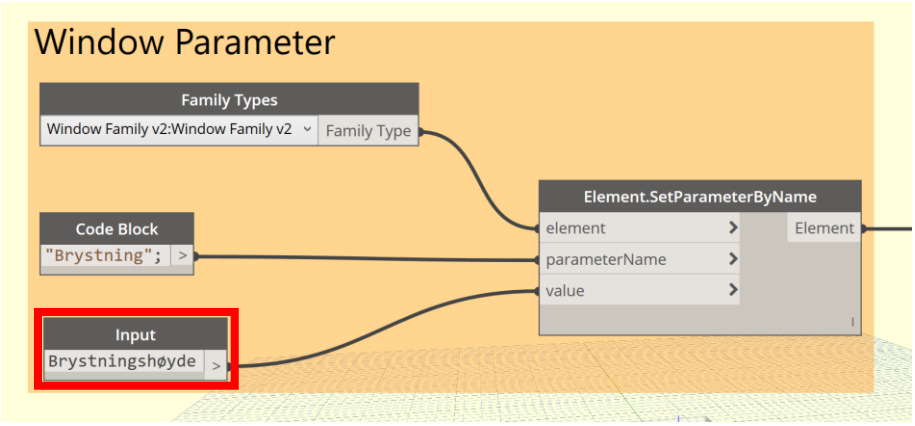


Når dagslysberegningsskriptet er ferdig og testet kan det lages en tilsvarende "Custom Node". Dette kan raskt gjøres ved å kopiere nodene fra skriptet og lime dem inn i ny "custom node". Det er viktig å legge til *inputs* og *outputs* til denne custom noden, slik at øvrige skripter kan legge inn og hente ut verdier fra denne. Husk å legge til en multipliseringsfunksjon på slutten, slik at "-1" multipliseres med resultatet fra beregningen. På denne måten vil resultatet bli gitt ut i negativ skala, hvor Optimo vil forsøke å maksimere resultatet fra funksjonen utfra variablene den setter inn.

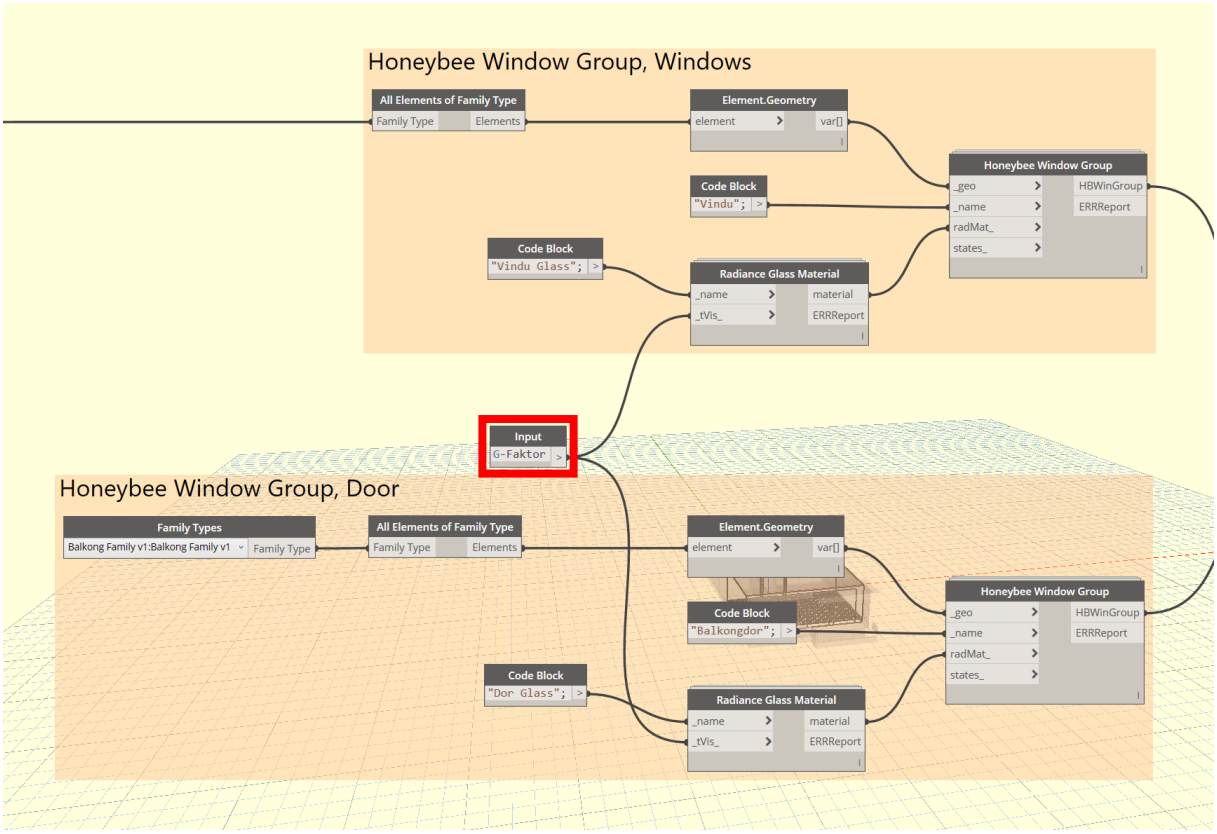
Figur 55: Inputene som vil bli brukt



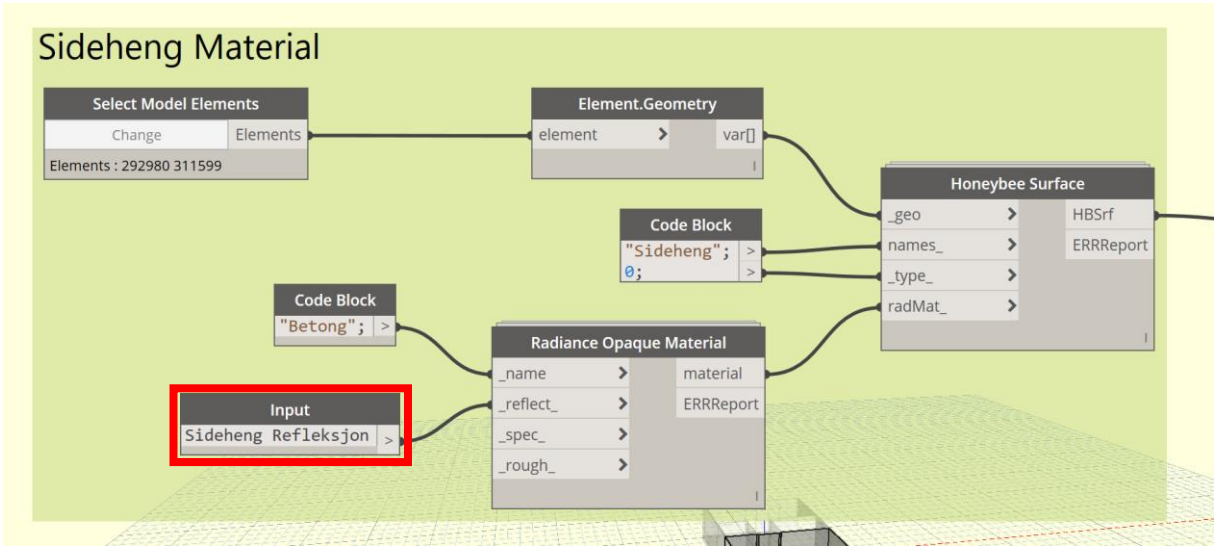
Figur 56: "Construct Location and Sky"-gruppen med input



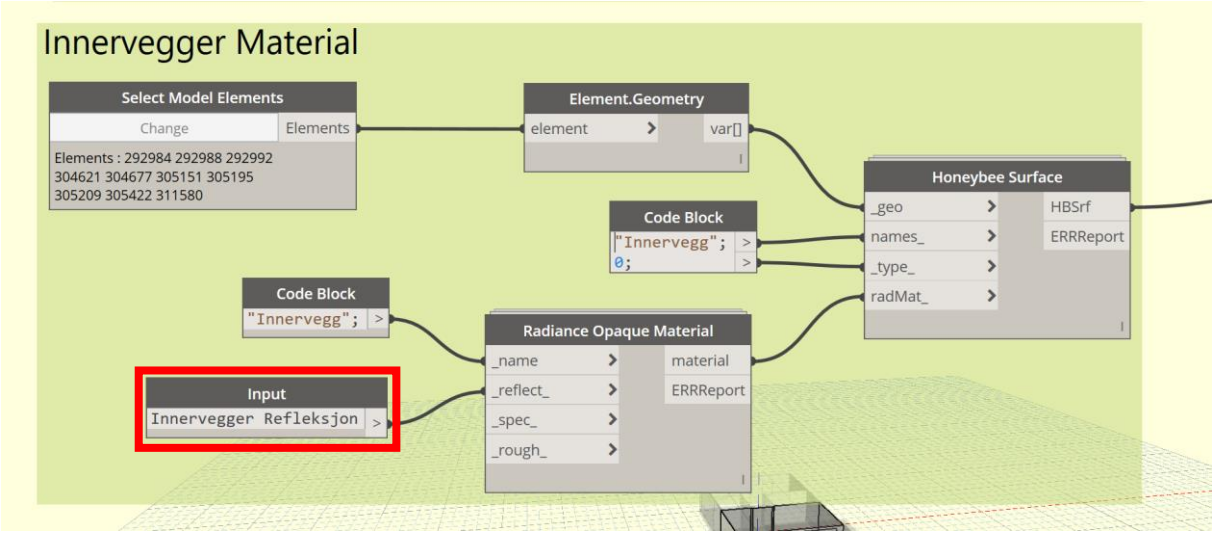
Figur 57: "Window Parameter"-gruppen med input



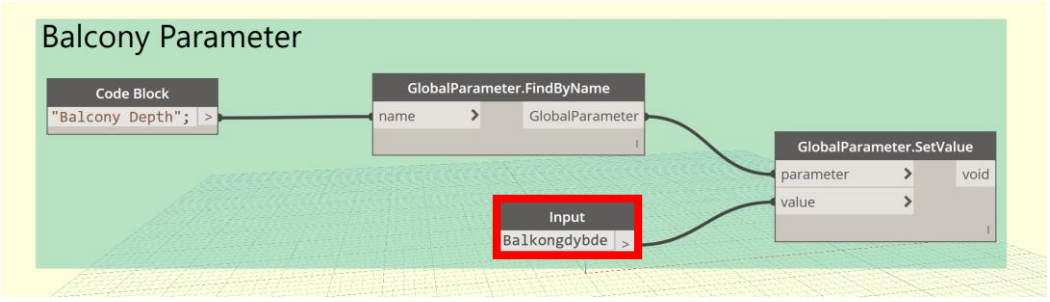
Figur 58: "Honeybee Window Group"-gruppene med input



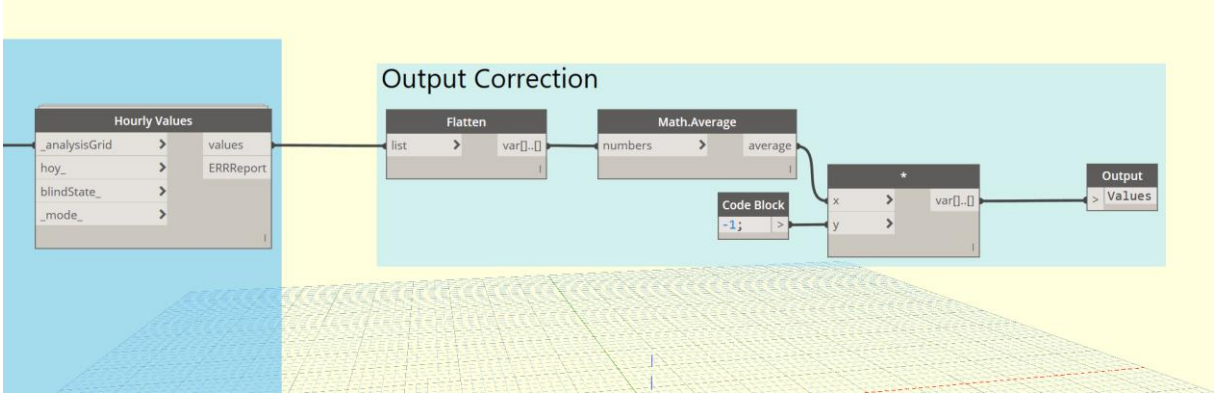
Figur 59: "Sideheng Material"-gruppen med input



Figur 60: "Innervegger Material"-gruppen med input



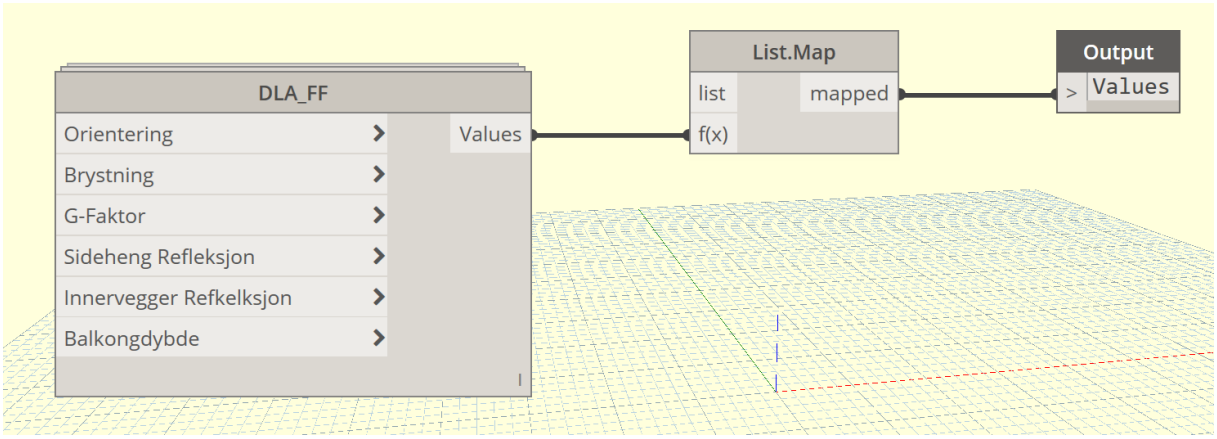
Figur 61: "Balcony Parameter"-gruppen med input



Figur 62: Output, som utgir resultatet fra "Daylight Analysis"-gruppen, som gjennomsnitt i negativ skala.

List.Map

Ikke alle funksjoner kan kalkulere lister som input. Dette vil si, bruke listene som input data for og så gi ut resultatet for hver input i form av en liste. Dette gjelder blant annet dagslysberegningsfunksjonen. Siden Optimo-pakken vil generere flere alternativer for hver variabel må denne funksjonen bruke List.Map-funksjonen for å beregne hvert alternativ. List.map vil mate ett og ett alternativ gjennom funksjonen, for og så utgi resultatene i en liste.



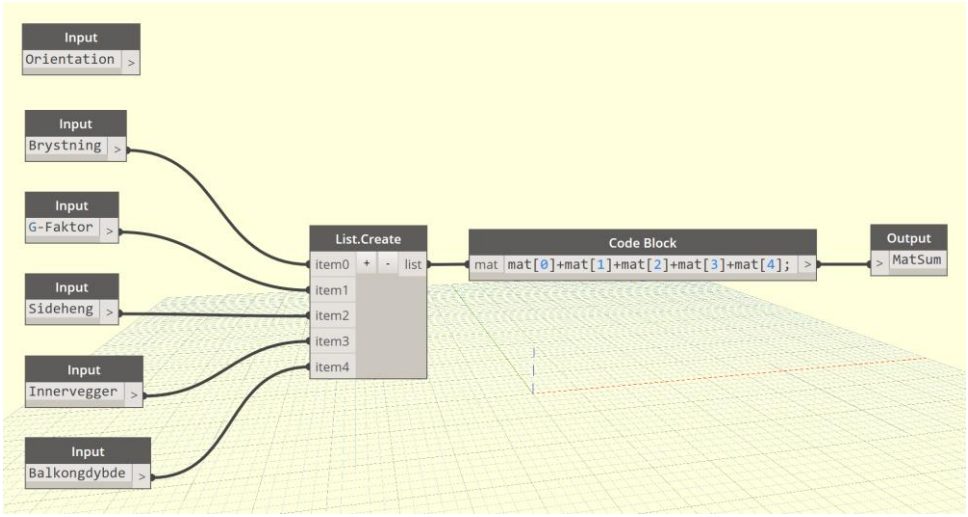
Figur 63: DLA_FF (Daylightanalysis Fitness Function) med List.Map

Dermed vil dagslysberegnings-noden bli lagt inn i en *ny* Custom node, som vil bli representere fitnessfunksjonen i Optimo-skriptet.

Materialbruks-fitnessfunksjon

Når dagslysberegningsnoden er ferdig, som vil fungere som en fitness funksjon, er det på tide å lage den andre fitness funksjonen, *Materialbruk*. Lag en ny "Custom Node" og legg til alle seks inputs (husk rekkefølge), og én output. Orientering vil ikke ha noe å si på materialbruken, så denne blir ikke brukt inne i selve noden. Legg så til en funksjon som adderer inputene, for å så koble den videre opp mot outputen. Denne fitnessfunksjonen vil altså bli minimert av Optimo, hvor den vil se etter variablene som gir minst mulig materialbruk.

Addisjonsfunksjonen klarer å beregne lister som input, slik at List.Map *ikke* behøves for denne fitnessfunksjonen.



Figur 64: Materialbruks Fitnessfunksjonen

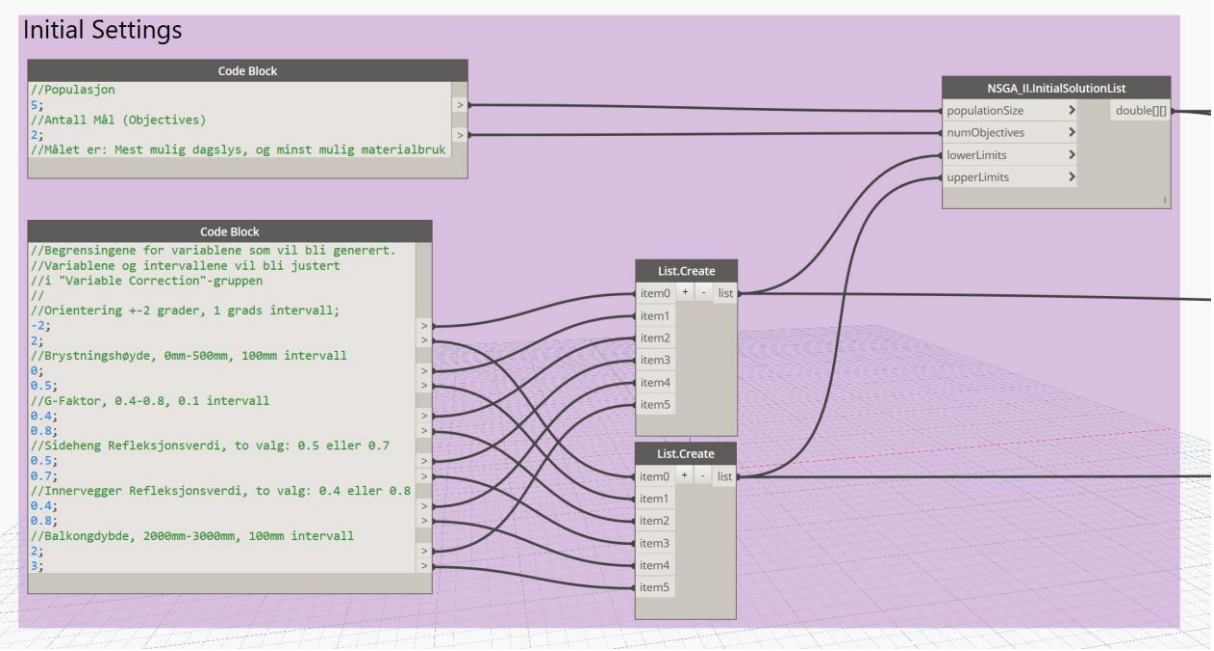
Optimo (Hovedskript)

Når alle fitnessfunksjonene er ferdige er det klart for å starte på hovedskriptet. Dette skriptet vil brukes til å de optimale alternativene.

Initial Settings

I denne gruppen vil brukeren legge inn de første innstillingene for skriptet. Her må brukeren bestemme populasjon, antall mål (fitnessfunksjoner), min- og maks grenser for variablene. Disse legges enkelt inn med code block noder, som kobles til «Initial Solution List».

Min og maks grensene må legges inn som lister til «Initial Solution List». Om flere variabler skal brukes kan disse simpelthen legges til i listen.

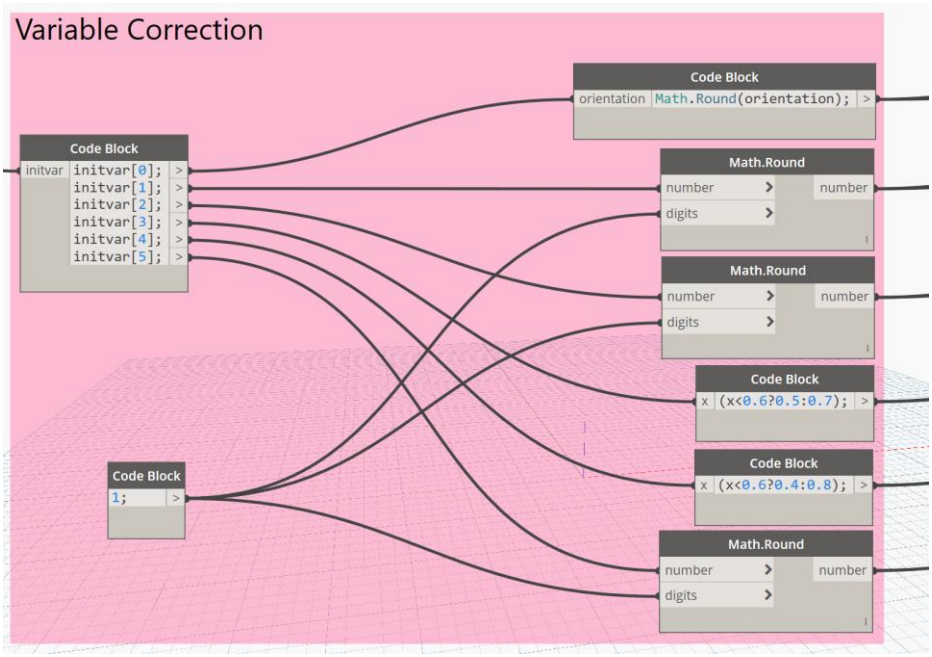


Figur 65: "Initial Settings"-gruppen

I denne gruppen vil Optimo så generere første generasjon med variabelsett, av tilfeldige variabler.

Variable Correction

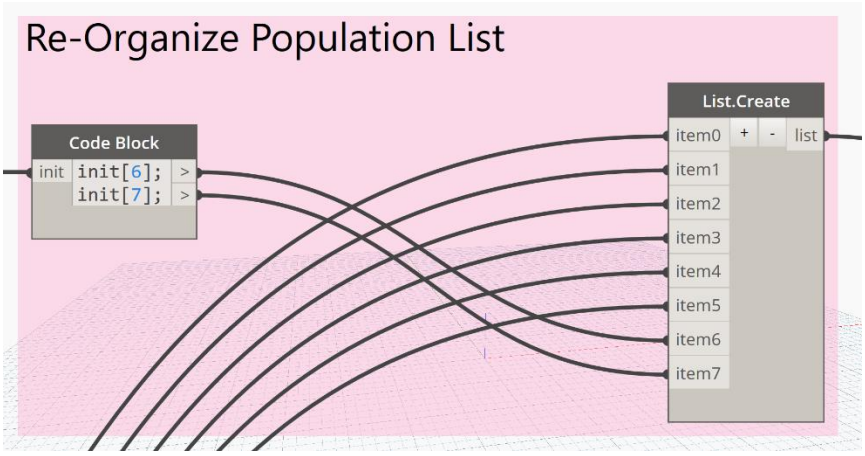
Variablene generert i «Initial Settings» er tilfeldige verdier innenfor de satte grensene, men i dette tilfellet er enkelte av disse variablene kun ment å ha to valgt eller spesifikke intervaller innenfor grensene. Dermed brukes nodene i «Variable Correction» gruppen til å korrigere disse.



Figur 66: "Variable Correction"-gruppen

Re-Organize Population List

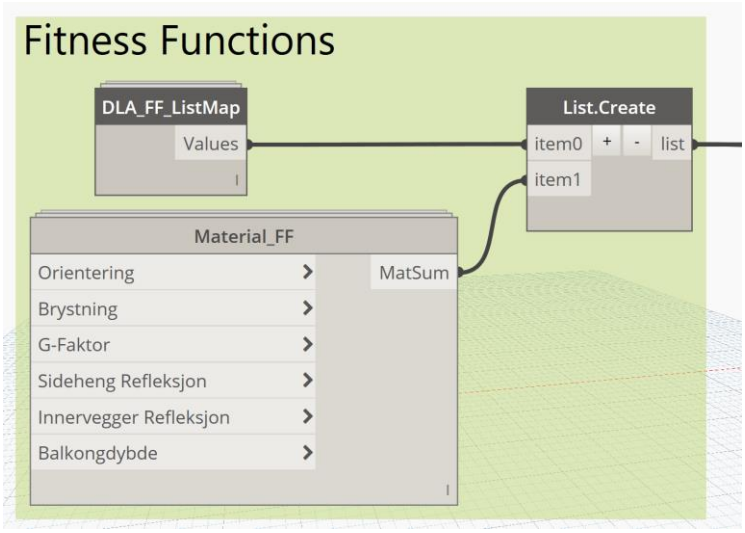
Etter at variablene har blitt korrigert kan de settes tilbake inn i populasjonslisten for å bli brukt videre. I tillegg må de to siste feltene fra den første populasjonen settes tilbake inn i populasjonslisten. Funksjonen blir overskrive verdien i disse feltene med resultatene fra fitnessfunksjonene.



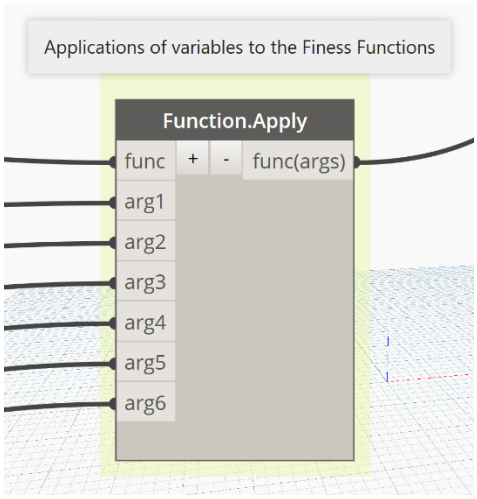
Figur 67: "Re-Organize Population List"-gruppen

Fitness Functions

Så er det på tide å legge inn fitnessfunksjonene, og disse må legges inn i en liste.



Figur 68: "Fitness Functions"-gruppen



Figur 69: Function.Apply-noden

Application of Fitness Functions

Etter korrigeringen vil variabelen også bli koblet opp mot fitnessfunksjonene gjennom noden «Function.Apply». I denne noden vil alle variablene, som er koblet inn som argumenter, bli brukt i fitnessfunksjonene som er koblet inn som en liste. Denne noden vil så utgi resultatene fra fitnessfunksjonene for alle variabelsettene, som sendes videre til «Generation Loop»-gruppen.

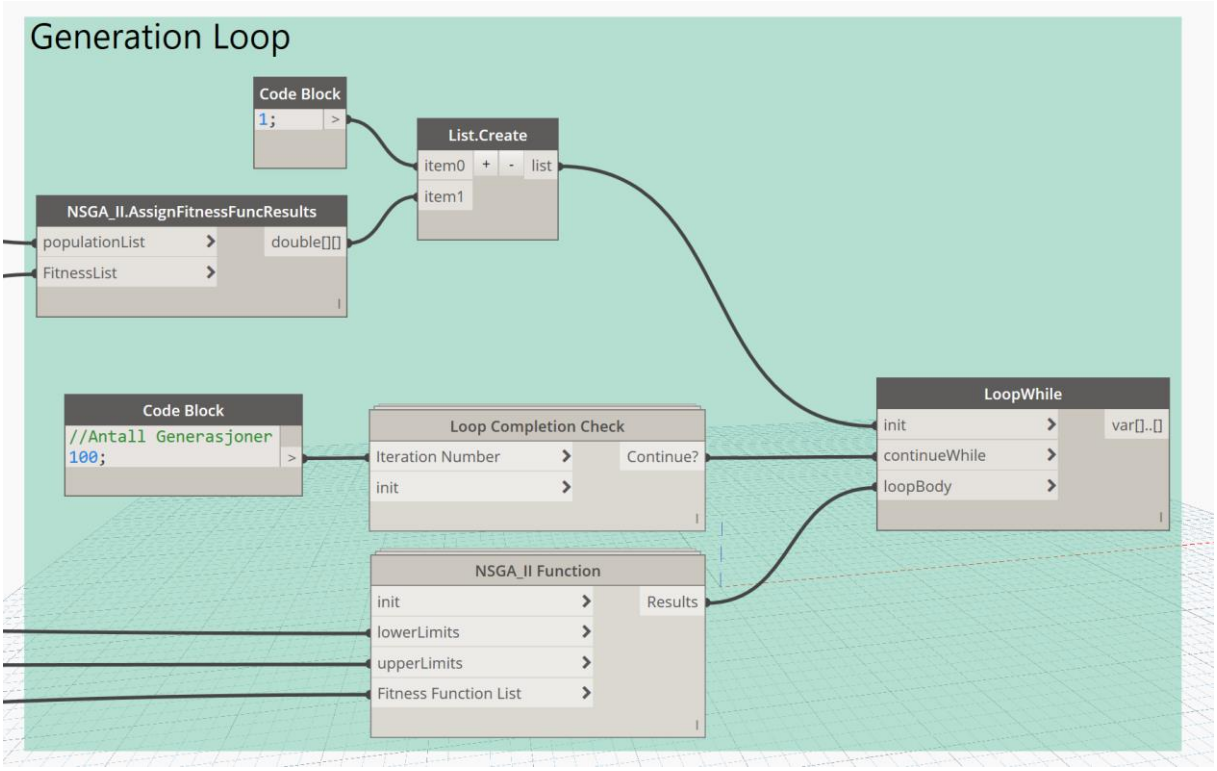
Generation Loop

Her kobles de første variabelsettene opp mot deres resultater fra fitnessfunksjonene. Dette gir Optimo et grunnlag for neste generasjon med variabelsett, hvor disse resultatene blir evaluert og de beste blir videreutviklet for neste generasjon.

Brukeren må sette inn et «Iteration Number» i denne gruppen, og dette vil si antall generasjoner som vil bli generert.

De nye generasjonene vil bli generert i «NSGA_II Function», hvor de nye variabelsettene vil bli testet ut i fitnessfunksjonene, og videre evaluert med tanke på resultatene.

Gitt stor nok starts-populasjon («Population» i «Initial Settings»-gruppen), og antall generasjoner (Iteration Number) vil dette skriptet kunne beregne seg frem til de mest optimale løsningene ved hjelp av NSGA_II funksjonen.

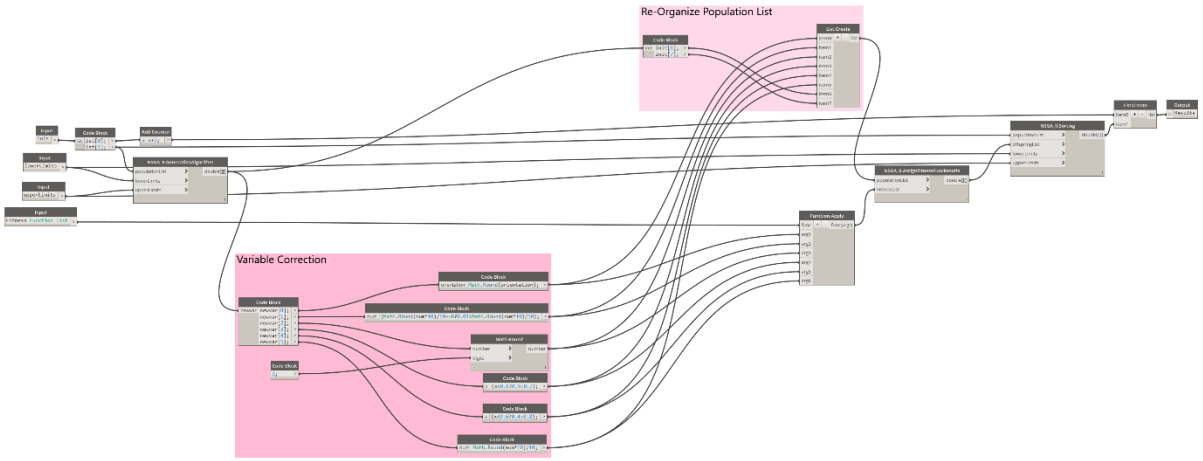


Figur 70: "Generation Loop"-gruppen

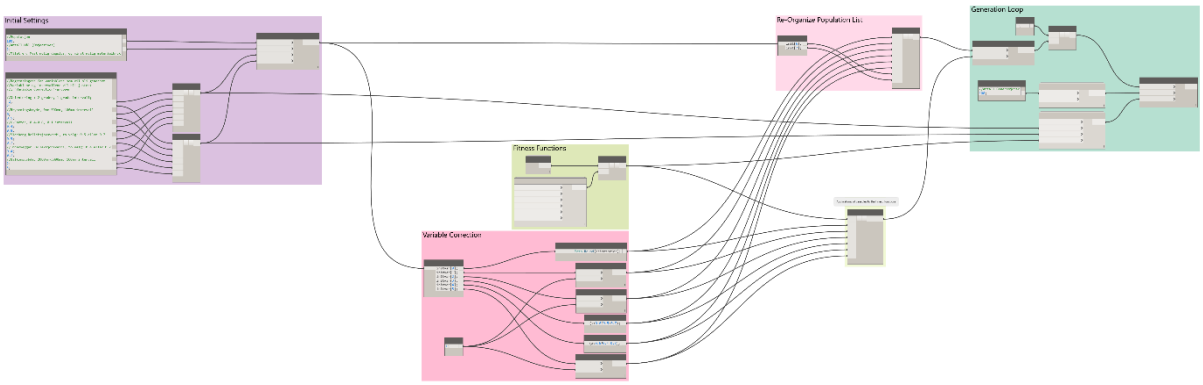
Endring av NSGA_II Funksjonen

Den originale NSGA_II funksjonen som følger med Optimo-pakken må justeres for å kunne brukes i denne oppgaven. Den kan enten modifiseres direkte, eller kan nodene kopieres inn i en ny custom node slik at den originale fortsatt kan brukes i andre skripter.

Modifikasjonene er variabel korrigeringen fra «Variable Correction»-gruppen, «Re-organization Population List» og utvidelsen av function.apply-noden. De to gruppene kan kopieres fra hovedskrpitet, og limes inn i NSGA_II funksjons-noden.



Figur 71: Innsiden av den nye "NSGA_II Function"-noden



Figur 72: Fullt skript for optimaliseringen

Koblinger mellom gruppene:

Fra			Til		
Gruppe	**Node**	**Output**	**Gruppe**	**Node**	**Input**
Initial Settings	NSGA_II. InitialSolutionList	Double[][]	Re-Organize Population List	Code Block	initvar
Initial Settings	NSGA_II. InitialSolutionList	Double[][]	Variable Correction	Code Block	initvar
Variable Correction	Alle korrigeringsnodene	…	Function Apply	Function.Apply	arg…
Variable Correction	Alle korrigeringsnodene	…	Re-Organize Population List	List.Create	item (0-5)
Fitness Functions	List.Create	list	Function Apply	Function.Apply	func
Fitness Functions	List.Create	list	Generation Loop	Custom NSGA_II Function	Fitness Function List
Re-Organize Population List	List.Create	list	Generation Loop	NSGA_II.Assign FitnessFunc Results	PopulationList
Function Apply	Function.Apply	Func(args)	Generation Loop	NSGA_II.Assign FitnessFunc Results	FitnessList

Tabell 4: Koblinger mellom gruppene i optimaliseringsskriptet.

Oversikt over noder som *ikke* er del av Dynamos grunnleggende bibliotek:

Gruppe	Node	Pakke
Initial Settings	NSGA_II.InitialSoultionList	Optimo
Generation Loop	NSGA_II.AssignFitnessFuncResults	Optimo
Generation Loop	Loop Completion Check	Optimo
Generation Loop	NSGA_II Function	Optimo *(Endret fra orginalen)*
Fitness Function	DLA_FF_ListMap	*Egenutviklet*
Fitness Function	Material_FF	*Egenutviklet*

Tabell 5: Oversikt over node-produsent for optimaliseringsskriptet.

Presentasjon av resultat som varmekart

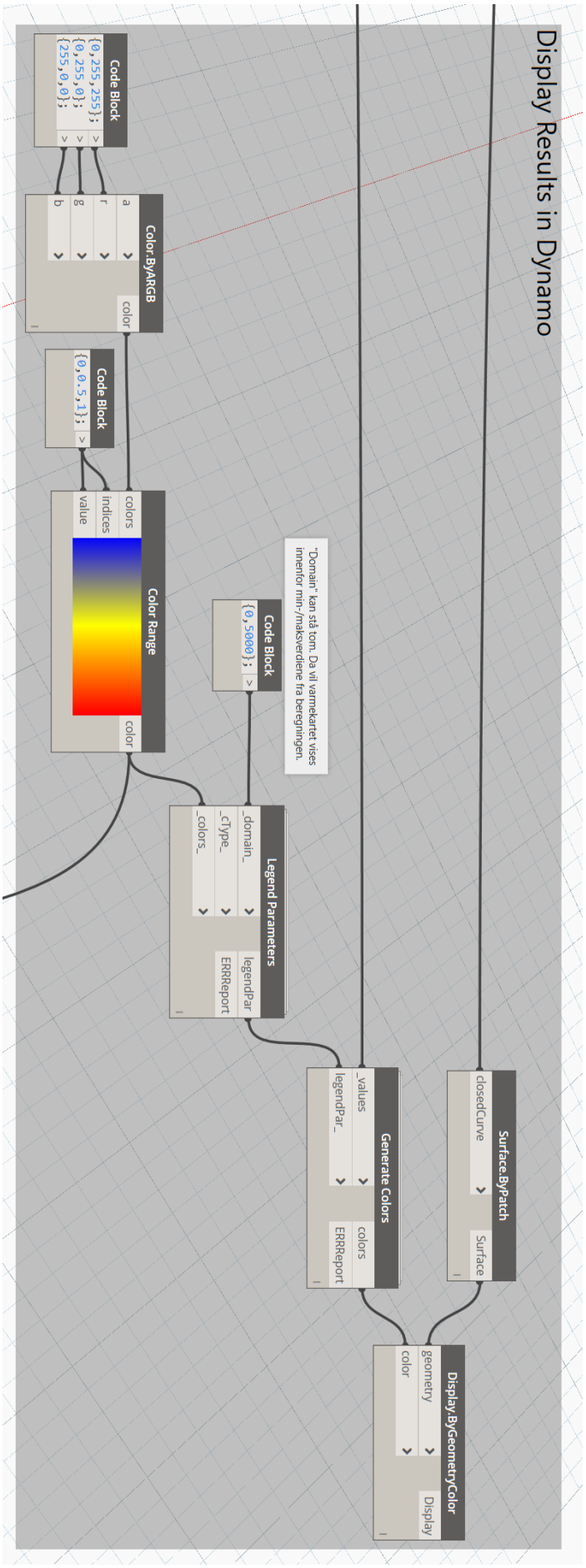
Til slutt er det mulig å lage et skript som kan presentere det endelige resultatet. Dette skriptet vil presentere resultatet av et valgt variabelsett som varmekart i både Dynamo og Revit, som viser lysforholdene i hele rommet.

For å gjøre dette kan det tidligere skriptet (*ikke noden*) fra dagslysberegningen modifiseres. Presentasjonene i Dynamo og Revit blir laget i to separate grupper.

NB! Først er det viktig å slette «Rekkverk Material»-gruppen, og redusere antallet inputs i påfølgende Create.List-node. Hvis ikke dette blir gjort vil ikke varmekartene være synlige.

Først lager vi gruppen som vil presentere resultatet i Dynamo.

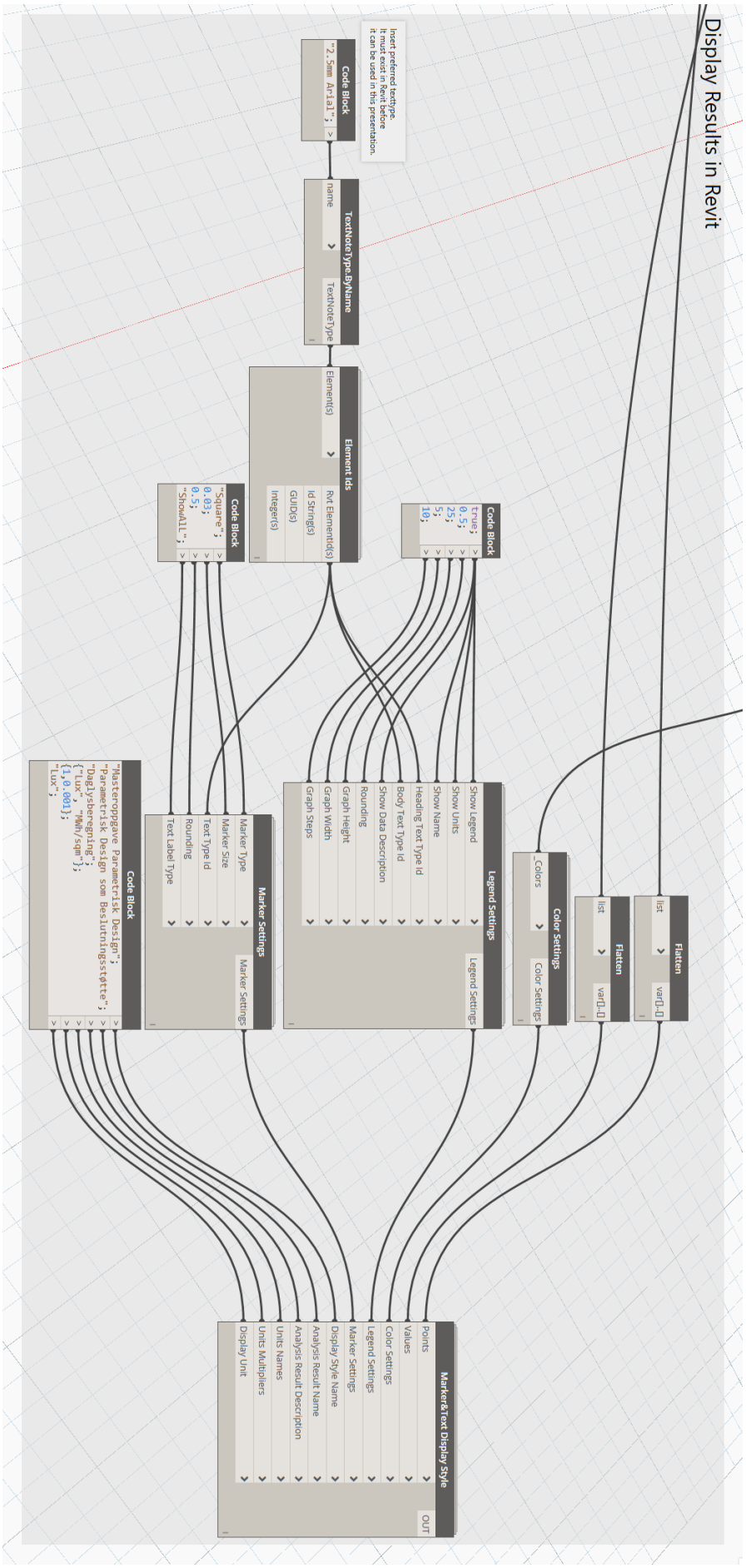
Til dette brukes først noden «Surface.ByPatch» som vil lage en overflate av rutenettet i rommet. Så legger vi til fargene. Dette kan gjøres på forskjellige vis, men i denne oppgaven blir dette gjort fra bunnen av. Først «Color.ByARGB», med verdier i form av RGB tall for rød, grønn og blå. Flere farger kan legges til ved å bruke lister. “Color Range” blir brukt til å vise fargene med gradientene, hvor inputene til noden brukes til å velge rekkefølge/plassering av fargene (på en skala fra 0.0 til 1.0). Videre legges noden «Legend Parameters» inn, hvor man også kan legge inn grensene for resultatene. Om denne inputen blir stående tom vil grensene bli minimum- og maksimum verdiene fra analysen. Videre trenger vi noden «Generate Colors». Denne anvende fargene til resultatene. Til slutt setter vi inn noden «Display.ByGeometryColor», som vil vise rutenettet med fargene i Dynamo. Se figur på neste side for referanse.



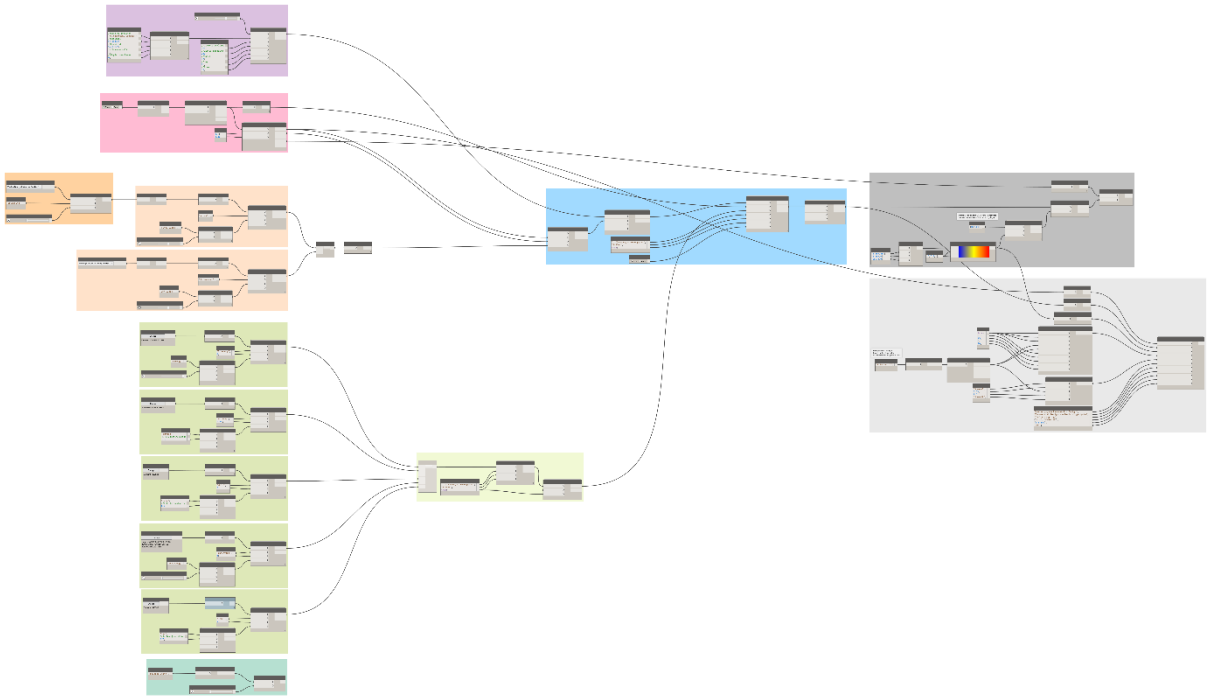
Figur 73: Display Result in Dynamo

Når resultatet skal presenteres i Revit trenger vi først to flatten noder, slik at verdiene fra input-listene kan leses av funksjonen. I tillegg trenger vi en color settings node, som vil bruke fargene fra «Display Results in Dynamo»-gruppen. Så trenger vi TextNote.ByType noden som vil finne tekst typer som eksisterer i Revit, og vi bruker en Code Block til å skrive inn ønsket teksttype. Denne kobles opp mot Elements Ids noden, som brukes for å finne id-koden til teksttypen. Videre legger vi inn «Legend Settings»-noden. Til denne brukes også en Code Block for å legge inn verdiene, bortsett fra teksttypen som vi får fra forrige node. Denne noden brukes for å sette innstillingene for legenden til presentasjonen. Vi trenger også en «Marker Settings»-node, som også bruker teksttypen fra «Elements Ids»-noden. I tillegg trenger denne input som vi setter inn med en Code Block. Denne noden brukes til å sette innstillingene for markørene i presentasjonen. Til slutt legger vi til «Marker&Text Display Style»-noden. Vi kobler de eksisterende nodene til denne, i tillegg til verdier som vi gir ved hjelp av en Code Block. Når alle nodene er koblet opp vil resultatene presenteres som et varmekart i Revit-modellen. Både i 2D eller 3D, etter hvilket view som er aktivt når skriptet kjøres.

Se figur på neste side for referanse.

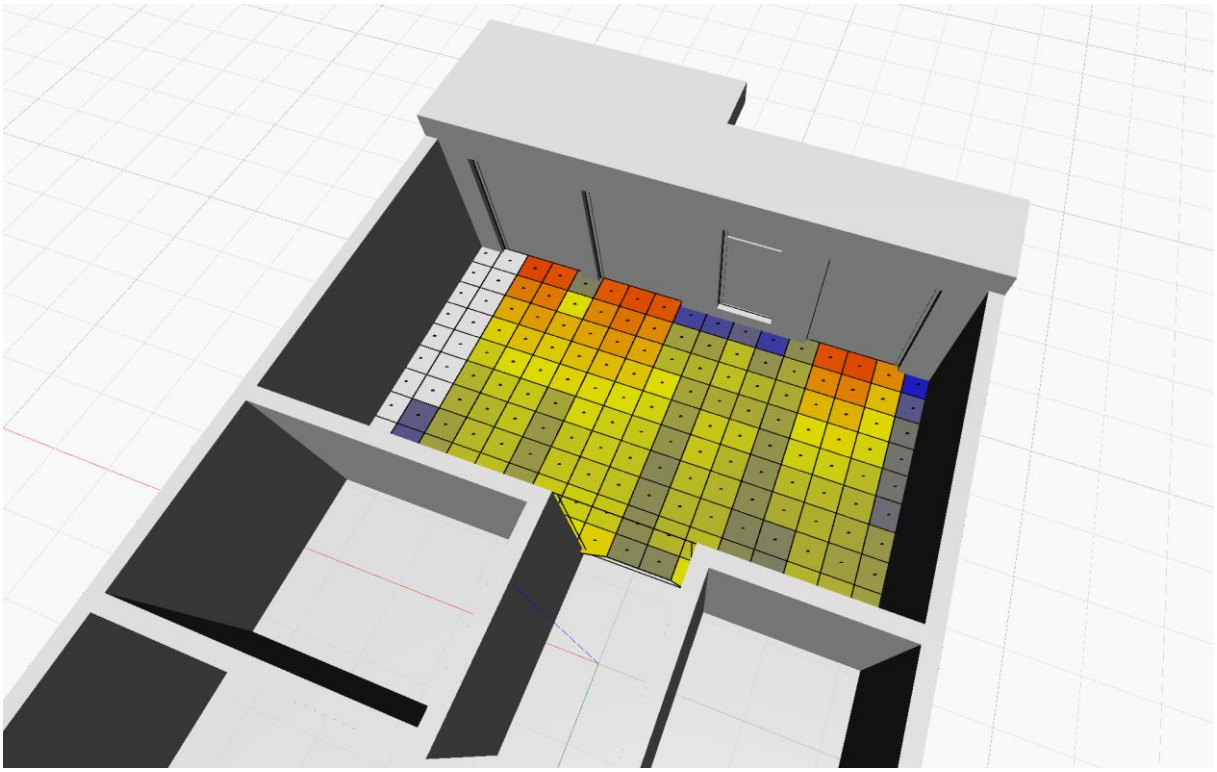


Figur 74: Display Results in Revit



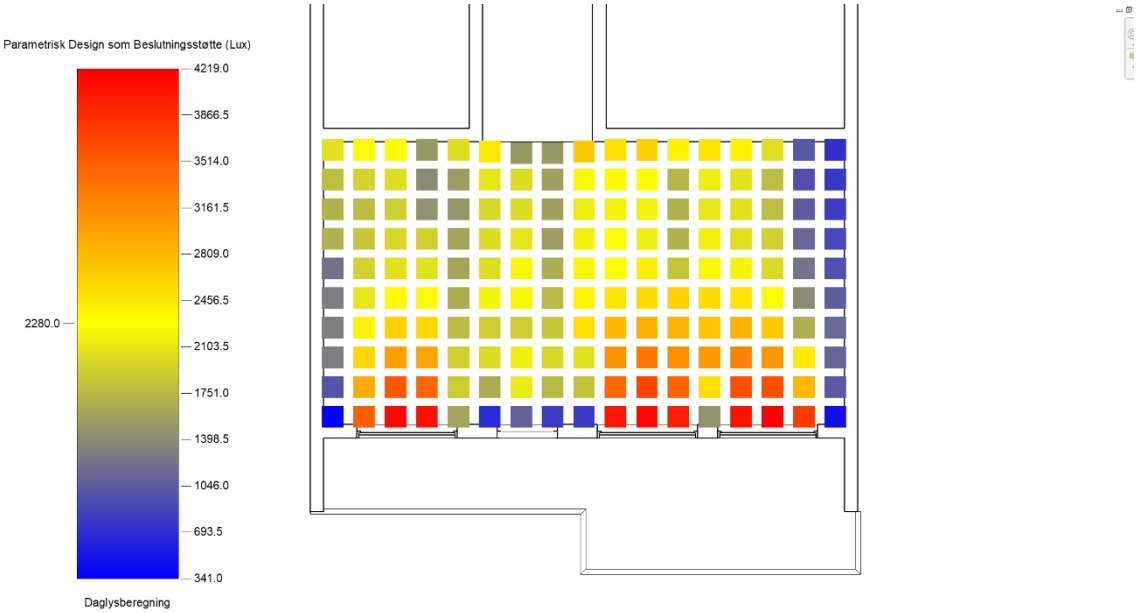
Figur 75: Hele skriptet for å vise resultatet i Dynamo og Revit

Enkelte ganger klarer ikke presentasjonen i Dynamo å legge til farge for alle rutene i Dynamo. Merk i figuren under at *kun* døren har brystning, kombinert med rutenett 25mm over gulvet. Brukeren kan selv sette hvilken høyde over gulvet som blir beregnet. Denne kan verdien kan endres i «Collect Rooms»-gruppen i noden «Generate Test Points for HBZones».

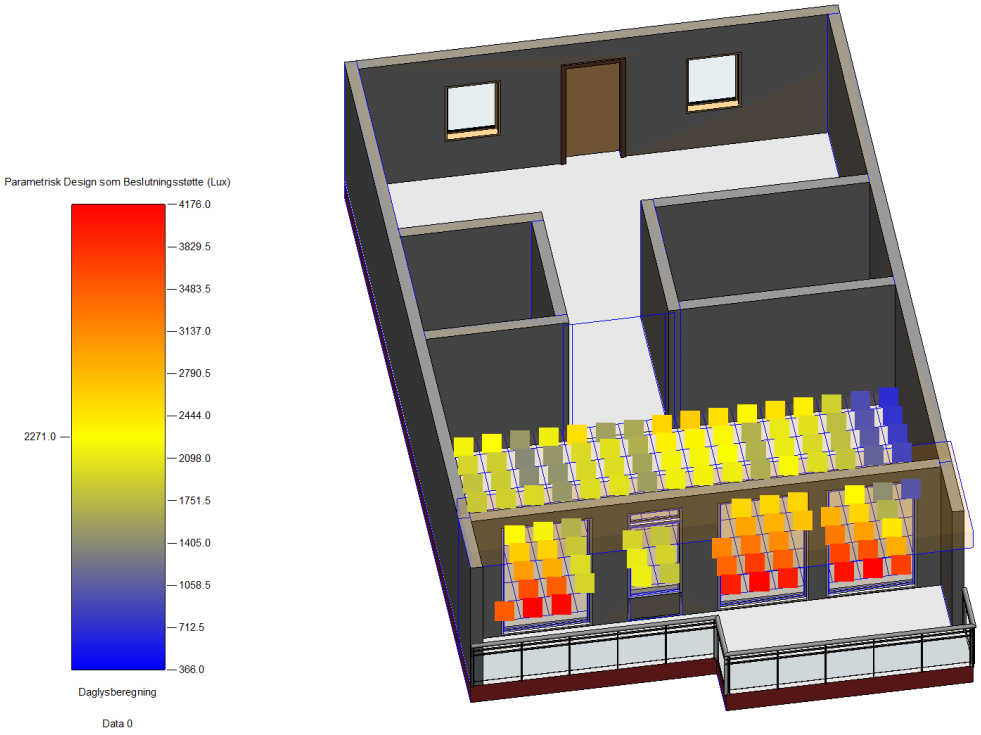


Figur 76: Resultat i Dynamo

Under ser vi presentasjonene i Revit (med en skala på 1:50). Formene på markørene (firkantene i rutenettet) kan også endres av brukeren. Dette kan gjøres ved å skrive «Circle» eller «Triangle» istedenfor «Square» i Code Blocken som er koblet til «Marker Settings» i «Display Results in Revit»-gruppen.



Figur 77: Resultat i Revit (2D)



Figur 78: Resultat i Revit (3D)

Koblinger mellom gruppene:

Fra			Til		
Gruppe	**Node**	**Output**	**Gruppe**	**Node**	**Input**
Collect Rooms	Generate Test Points from HBObjects	polygons	Display Results in Dynamo	Surface.ByPatch	ClosedCurve
Daylight Analysis	Hourly Values	values	Display Results in Dynamo	Generate Colors	_values
Collect Rooms	Generate Test Points from HBObjects	testPts	Display Results in Revit	Flatten (Points)	list
Daylight Analysis	Hourly Values	values	Display Results in Revit	Flatten (Values)	list
Display Results in Dynamo	Color Range	color	Display Results in Revit	Color Settings	_Colors

Tabell 6: Koblinger mellom gruppene i presentasjonsskriptet.

Oversikt over noder som *ikke* er del av Dynamos grunnleggende bibliotek:

Display Results in Dynamo	Generate Colors	Ladybug
Display Results in Dynamo	Legend Parameters	Ladybug
Display Results in Revit	Color Settings	Archi-lab
Display Results in Revit	Element Ids	Archi-lab
Display Results in Revit	Legend Settings	Archi-lab
Display Results in Revit	Marker Settings	Archi-lab
Display Results in Revit	Marker&Text Display Style	Archi-lab

Tabell 7: Oversikt over node-produsent for presentasjonsskriptet.

5. Resultat, Refleksjon & Videre Arbeid

Dette kapittelet vil gjennomgå resultatene fra oppgaven, hva som kunne bli gjort annerledes oppgaven, og hva kan gjøres videre (videreutvikling) innenfor parametrisk design som beslutningsstøtte.

5.1 Resultat

Skriptene nevnt i forrige kapittel klarer å utføre oppgaven til en viss grad, men denne metoden klarer ikke å utføre kalkulasjonene som først antatt. Simuleringene tar veldig lang tid, og bruker *veldig* mye minne (RAM). I tillegg krasjet ofte Revit og/eller Dynamo tilfeldig mens skriptet kjørte.

Under er en tabell resultatene fra seks av simuleringene, utført på en maskin med 64GB RAM:

#	Variabler	Population	Iteration Num.	Tid	RAM
1	5	5	5	Ca. 20 min	21GB RAM
2	5	5	5	Ca. 20 min	22GB RAM
3	5	5	5	Ca. 20 min	21GB RAM
4	5	10	5	Krasj	Ca. 50GB RAM
5	5	10	5	Krasj	Ca. 40GB RAM
6	5	10	5	Over 6 timer (Krasj)	Ca. 60GB RAM

Tabell 8: Resultater fra simuleringene.

Merk at når populasjon er 5 og iteration number er 5 er det maksimalt 25 kombinasjoner som blir sjekket, mens populasjon på 10 og iteration number på 5 sjekker maks 50 kombinasjoner. Variablene som ble brukt var balkongdybde (11 alternativer), Refleksjonsfaktor Sideheng (to alternativer), Refleksjonsfaktor Innervegger (to alternativer), G-faktor (fem alternativer) og brystningshøyde (seks alternativer. Totalt finnes det $(11 \times 2 \times 2 \times 5 \times 6 =)1320$ alternativer, slik at veldig få av disse faktisk blir sjekket. Tidligere simuleringer med seks variabler (orientering med fem alternativer: 358°, 359°, 0°, 1° og 2°) gir en total på 6600 mulige kombinasjoner. Etter hvert som Dynamo må regne ut flere kombinasjoner går tidsbruken eksponentielt opp, slik at tidsbruken for å beregne *alle* kombinasjonene ikke er gjennomførbart.

En av grunnene til problemet med simuleringene ble antatt å være algoritmene som ble brukt i tredjeparts-pakke. Denne antagelsen baserte seg på at resultatene fra simuleringen (fra hver generasjon) ble liggende i minnet istedenfor å frigjøre det etter at simuleringen var over. Dermed ble algoritmene sjekket ved hjelp av Microsoft Visual Code Studio, men uten hell. Dette ble videre antatt å være på grunn av grensesnittet til Revit og Dynamo, som begge er under fortløpende utvikling. Slik programvarene operer i skrivende stund rensker de ikke minnet før programmet er lukket, men dette kan potensielt bli løst av utviklerne i fremtidige versjoner av programvarene.

Hovedproblemet med denne metoden blir altså minnebruken. Siden Dynamo helst skal kunne simulere gjennom alle alternativene, ville det vært best om programmet hadde forkastet brukte resultater (fra minnet) for å frigjøre minnet til å kalkulere nye resultater. Tiden programmene bruker på å kalkulere resultatene er antatt å være knyttet til minnet, hvor tidsbruken for kalkulasjonene går eksponentielt opp etter hvert som minnekapasiteten til maskinen fylles. Beregningene fra «Dagslysberegnings»-skriptet sto for mye av denne minnebruken.

En ny versjon av Dynamo ble utgitt under denne oppgaven, versjon 2.0.0. Denne versjonen tar et stort hopp fra sin forgjenger, men denne kunne ikke bli brukt på grunn av tredjeparts-pakkene. Siden den nye versjonen var slik et stort hopp fra sin forgjenger var det blant annet store endringer i grensesnittet i programvaren, og enkelte av tredjeparts-pakkene ble ubrukelige i den nye versjonen. Disse pakkene vil antakeligvis bli oppdatert av utviklerne etterhvert, slik at de vil fungere med den nyeste versjonen av Dynamo.

5.2 Refleksjon

Denne oppgaven var mitt personlige «første dykk» innen parametrisk design som beslutningsstøtte, og selv om metoden som ble brukt i oppgaven ikke ga ønsket resultat mener jeg fortsatt at parametrisk design som beslutningsstøtte kan i stor grad revolusjonere prosjekteringsprosessen i fremtiden. Det ble fokusert på dagslysberegninger i denne oppgaven, men dette seg jeg kun på som en liten bit av et helt system som kan dannes i fremtiden; Hvor man vil ha muligheten til å utføre lignende simuleringer for flere fag og mål under prosjekteringsfasen. På denne måten kan man, utfra en rekke variabler og regelsett, kunne få utgitt optimal plassering og høyde på bygg for å optimalisere tomteutnyttelse, boligtyper/størrelser for å finne optimal BRA%, i tillegg til mange flere. På denne måten kan nesten hele designprosessen kunne automatiseres, hvor alle krav og forskrifter vil følges samtidig som *alle* alternativer kan tas i betrakting for å kunne lage de mest kost-nytte effektive byggene som er mulig.

Til å gjøre slike simuleringer for prosjektet trenger man kun sette inn begrensinger for variablene (i form av regelsett), så simulerer verktøyet gjennom alle kombinasjoner og alternativer, og utgir til slutt de (f.eks.) 100 beste alternativene som arkitekt kan velge mellom. Dette vil si at man kan komme frem til bedre design enn tidligere, i tillegg til at mindre tid og ressurser blir brukt til prosessen.

Det er mulighet for å koble optimaliserings-skriptene og resultatene i Dynamo opp mot modellering i Revit. På denne måten kan programmene både simulere seg frem *og* modellere det mest optimale designet.

Denne oppgaven viser i det minste at det vil være mulig å bruke parametrisk design som beslutningsstøtte, men at det krever god kompetanse innenfor programmering, skripting, i tillegg til tilgang til høyt-ytende simuleringsmaskiner og -verktøy. Revit og Dynamo ble valgt til å løse denne oppgaven på grunn av kompetansen min innen disse, i forhold til alternativene. Med tanke på dette ble det ansett at det ville gi bedre resultat for oppgaven, hvis jeg videreutviklet min eksisterende kompetanse i motsetning til å lære meg de andre programmene fra bunnen av. Resultatet fra oppgaven kunne kanskje vært annerledes om andre verktøy hadde blitt tatt i bruk (f.eks. Rhinoceros og Grasshopper).

Det har vært veldig tidskrevende å lære seg disse teoriene og programmene, spesielt Dynamo-pakkene som ikke har mye eksisterte lærestoff på internett. Dermed ble dette lært primært gjennom prøving og feiling.

De nye oppdateringene til programvarene har potensiale til å gjøre tidligere skripter ubrukelige (som ble opplevd i denne oppgaven), hvor de endrer i algoritmene til hovedprogrammet slik at skriptene ikke lenger kan operere som de skal. Ved slike tilfeller er det fortsatt mulig å endre skriptene slik at de vil fungere igjen, men dette kan kreve kompetanse innenfor programmering i enkelte tilfeller. Ellers kan man bruke tidligere versjoner av programvaren, slik det ble gjort i denne oppgaven.

5.3 Lignende metoder og oppgaver

Lignede metoder for å effektivisere prosjekteringsfasen ved hjelp av parametrisk design har lenge blitt utforsket. Spacemaker AS, basert i Oslo, viser at det er mulig. De kan simulere diverse kombinasjoner for hele prosjekter, med flere forskjellige variabler. De bruker egne programvarer, i tillegg til sky-baserte maskiner for å utføre simuleringene. På denne måten støter de ikke på de samme problemene som jeg har gjort, hvor Revit og Dynamo ikke spesifikt er laget for å utføre slike oppgaver, samtidig som den sky-baserte maskinkraften gir mulighet til å simulere større prosjekter. Jeg har dessverre ikke tatt kontakt med Spacemaker grunnet tidsdisposisjonen mot slutten av oppgaven, men ser utfra deres arbeid at konseptet er mulig.

5.4 Videre Arbeid

Med tanke på at kun to programmer ble utforsket i denne oppgaven, vil det være mulighet for å oppnå andre resultater ved å utforske andre programmer. Som tidligere nevnt i oppgaven er det mulig å sette opp lignende skripter i Grasshopper for Rhinoceros, men grunnet forskjell oppbygning i forhold til Dynamo og Revit kan denne kombinasjonen gi et annet resultat. Funksjonen som brukes av Optimo, NSGA II, virker lovende for denne typen kalkulasjoner. Bruken av «evolusjonsteorien» til å forkaste dårlige alternativer, hvor man vil stå igjen med de beste virker fordelaktig. Gitt god nok minnehåndtering vil denne metoden kunne simulere *alle* alternativer, hvilket var et av hovedpunktene ved bruk av parametrisk design. Med dette sagt kan denne metoden utforskes med andre programmer og verktøy. Om man har programmer med bedre minnehåndtering kan forbedre utførelsen av denne metoden, samt resultatene.

Andre kriterier bør også utforskes, hvor neste skritt bør omhandle noe annet enn dagslys. Det har blitt understreket flere ganger i oppgaven at dette konseptet ikke kun omhandler *ett* kriteria, men at det skal kunne ta for seg flere forskjellige. I tillegg til at dagslysberegningene sto for mye av minnebruken, virker det lurt å ta for seg et annet kriteria slik at selve metoden kan utforskes videre og forbedres.

Det kan også være lurt å få kontakt med andre som jobber med lignende systemer og parametrisk design, slik at man kan få en peilepinne for hvordan oppgaven kan løses med tanke på hva som allerede er visst. Utfra erfaringer i denne oppgaven er mange av programmene og teoriene veldig tidskrevende å lære, slik at dialog med erfarne mennesker innenfor feltet kan vise seg å være verdifullt. Spacemaker er en av de som allerede har jobbet med dette i noen år, og kan være et godt sted å starte. Ikke kun for spesifikke løsningsforslag for denne type oppgave, men også for generell kunnskap om parametrisk design med alt det innebærer.

6. Konklusjon

Den valgte metoden for å utføre oppgaven, med Revit og Dynamo som frontfigurer, ga ikke ønskelig resultat. Denne metoden krever både mye tid og maskinkraft slik at det anses som uvesentlig for større prosjekter. Dette er hovedsakelig et problem med Revit og Dynamo, hvor programmenes primære funksjon ikke er rettet mot denne typen oppgaver. Alt i alt klarte programmene å løse oppgaven, men brukte mye tid og maskinkraft i gjengjeld til få resultater.

Det er kun noen få programmer som ble undersøkt til å løse problemstillingen i denne oppgaven, men det kan hende at andre programmer kan håndtere slike oppgaver bedre.

NSGA II (Non-dominated Sorting Genetic Algorithm) er på papiret en god teori og bygge dette konseptet på, men i denne oppgaven var det programmene som forhindret denne funksjonen fra å vise sitt fulle potensial. Om denne metoden (brukt av Optimo-pakken) kan implementeres i andre programmer med bedre minnehåndtering er det mulig å lage en god metode hvor parametrisk design kan bli brukt som beslutningsstøtte.

Andre kriterier enn dagslys bør også undersøkes, siden disse beregningene sto for mye av minnebruken, som ble sett på som oppgavens største problem.

Det er mulig å utføre denne typen simuleringer, som visst av Spacemaker (se kapittel 5.3).

Utfra oppgaven og Spacemaker sitt arbeid ser vi at parametrisk design som beslutningsstøtte er realistisk, slik at det fortsatt kan være metoder for å oppnå dette. Dessverre for denne oppgaven vil ikke Revit og Dynamo (hvert fall ikke dagens versjoner) være gode nok verktøy for å anvende parametrisk design som beslutningsstøtte.

7. Referanser

Blood Vessels [Bilde]. (2016). Hentet fra http://tellmewhyquestions.blogspot.com/2016/03/how-can-human-body-hold-so-many-miles.html

Dagslysfaktor – prinsipp [Bilde]. (2014). Hentet fra https://lyskultur.custompublish.com/getfile.php/3059152.2372.eqsydxupbw/Faktaark+03_Dagslysfaktor_3-2014+kopi.pdf

Danish Building Research Institute. (2013). *Daylight calculations in practice*. Hentet fra https://sbi.dk/Assets/Daylight-calculations-in-practice/daylight-calculations-in-practice.pdf

Feasible Desicion variable space and feasible objective space [Bilde]. (2017). Hentet fra http://oklahomaanalytics.com/data-science-techniques/nsga-ii-explained/

Leaf-veins [Bilde]. {2018}. Hentet fra https://fwallpapers.com/view/leaf-veins

Linge, Geir Nordal. (s.a.). *Hva er egentlig … BIM*. Hentet fra https://relasjon.skanska.no/hva-er-egentlig-bim/

Linge, Geir Nordal. (s.a.). *Hva er egentlig … VDC*. Hentet fra https://relasjon.skanska.no/hva-er-egentlig-vdc/

Lyskultur. (2014). Faktaark FO3. Hentet fra https://lyskultur.custompublish.com/getfile.php/3059152.2372.eqsydxupbw/Faktaark+03_Dagslysfaktor_3-2014+kopi.pdf

Pareto front of SCH Test Function [Bilde]. (2014). Hentet fra http://dynamobim.org/optimo/

Paul Calle. (2017). *NSGA-II explained!*. Hentet fra http://oklahomaanalytics.com/data-science-techniques/nsga-ii-explained/

Radiance-Online. (2016). *About Radiance*. Hentet fra https://www.radiance-online.org/about

SCH Test Function [Bilde]. (2014). Hentet fra http://dynamobim.org/optimo/

SINTEF. (2016). *Krav til dagslys i TEK10*. (20150471). Hentet fra https://dibk.no/globalassets/byggeregler/tek10-til-tek17/rapporter/krav-til-dagslys-i-tek10_sintef_januar-2016.pdf

Wikipedia. (2018). *C Sharp (Programming Language)*. Hentet fra https://en.wikipedia.org/wiki/C_Sharp_(programming_language)

Wikipedia. (2018). *Kruse Smith*. Hentet fra https://no.wikipedia.org/wiki/Kruse_Smith

Wikipedia. (2018). *Parametric Design.* Hentet fra https://en.wikipedia.org/wiki/Parametric_design

Wikipedia. (2018). *Python (Programming Language)*. Hentet fra https://en.wikipedia.org/wiki/Python_(programming_language)

Wikipedia. (2018). *Radiance (Software)*. Hentet fra https://en.wikipedia.org/wiki/Radiance_(software)

8. Figurliste

9. Tabell liste