AUTODESK UNIVERSITY

CS21553

# Construction Dynam(o)ite
# Explode Productivity with Dynamo

Dieter Vermeulen - Autodesk

## Learning Objectives

- Understand the wide range of possibilities Dynamo offers for the construction industry
- Learn how to create your own Dynamo scripts to automate and manage the creation of constructible BIM models
- Learn how to use computational design to apply your design rules on construction models in Revit
- Learn how to make custom model checks to increase the constructability of your Revit model

## Description

The step from detailed design to construction in a Building Information Modeling (BIM)-enabled workflow is a challenge for many design offices and construction companies. In many situations, designs need to be changed after contractors have reviewed the models on constructability. In this class, you will discover how you can involve computational design in this process. You will learn how to use the Dynamo extension to automate all kinds of custom tasks in Revit software to make your design constructible. You will get inspired by the wide range of possibilities the Dynamo extension offers to create your own model check and to automate the creation of construction elements. This session features Revit, Dynamo Studio and Navisworks Manage.

## Your AU Expert

Working as a Technical Sales Specialist AEC for the Northern European region at Autodesk, **Dieter Vermeulen** is specialized in the products of the structural solutions portfolio. Within that domain he supports the channel partners and customers with workflow solutions, especially - but not limited to - for design and engineering. With Revit, Robot Structural Analysis and Dynamo as his sidekicks, he is offering BIM workflow solutions covering the building process from design over analysis to fabrication for steel and concrete constructions. He has over 15 years of experience in the structural engineering business, starting his career in 2000 at Jacobs Engineering in Belgium.

Twitter www.twitter.com/BIM4Struc
Team Blog www.autodesk.typepad.com/bimtoolbox/
LinkedIn www.linkedin.com/in/dietervermeulen
YouTube www.youtube.com/user/RevitbeyondBIM
Blog www.revitbeyondbim.wordpress.com

# Table of Contents

# Computational Design

## What is computational design?

The last few years we all experienced the transition from traditional Computer Aided Design (CAD) to Building Information Modelling (BIM). As our projects are getting more complex, and need more design solutions, the current BIM focused workflow is challenged again. The answer on that is **Computational Design**. This new design approach represents a profound shift in design thinking and methods. Representation is being replaced by simulation, and the crafting of objects is moving towards the generation of integrated systems through designer-authored computational processes.

While there is a particular history of such an approach in the building industry, its relative newness requires the continued progression of new modes of design thinking for the designers and engineers of the 21st century.

Literally, “computational design” means “giving shape with the means of the power of a computer and scripting”.

### *Autodesk Dynamo*

Autodesk gives an answer to this new challenge in our design world. This solution is called **Autodesk® Dynamo** (open source) and **Autodesk® Dynamo Studio** (Subscription). Dynamo lets designers and engineers create visual logic to explore parametric designs and automate tasks. It helps you to solve challenges faster by designing workflows that drive the geometry and behavior of design models. With Dynamo you will extend your designs into interoperable workflows for documentation, fabrication, coordination, simulation, and analysis.

## Genetic Optimization

Genetic Optimization is an optimization technique that makes use of Genetic Algorithms (GA). GAs usually operate on solutions that are encoded as genotypic representations (called chromosomes) from their original phenotypic representations (i.e. actual data values). GAs start with a set of initial solutions (population) that are randomly generated in the search space. For each solution in the current population, objective functions defining the optimization problem are evaluated and a fitness value is assigned to reflect its (relative) merit standing in the population. Based on the fitness values, a GA performs a selection operation that reproduces a set of solutions with higher fitness values from the previous generation to fill a mating pool. A crossover operation is then pursued with which two parent solutions in the mating pool are randomly selected and interchange, with a prescribed crossover probability, their respective string components at randomly selected bit locations referred to as cross sites. The resulting new solutions are called children or offspring. This step is meant to hopefully combine better attributes from the parent solutions so that child solutions with improved merits could be created. The next operation in GA is mutation that changes the genotype value at one or more randomly selected bit locations in a child solution with a prescribed mutation probability. This operation serves to possibly recover useful information that may never be accessible through selection and crossover operations and therefore encourages search into a completely new solution space. After these three basic operations, a new generation is created. The search process continues until prescribed stopping criteria are met, for example, when the maximum generation number is reached or improvement of the generation-wise optimized solutions is negligible.

Unlike single-objective problems where the objective function itself may be used as the fitness measure (possibly further with scaling and constraint-handling treatment), a multi-objective optimization algorithm needs a single fitness measure that reflects the overall merit of multiple objectives.

### *Genetic Optimization workflow*

A typical workflow in a genetic optimization process is represented in the flowchart below:

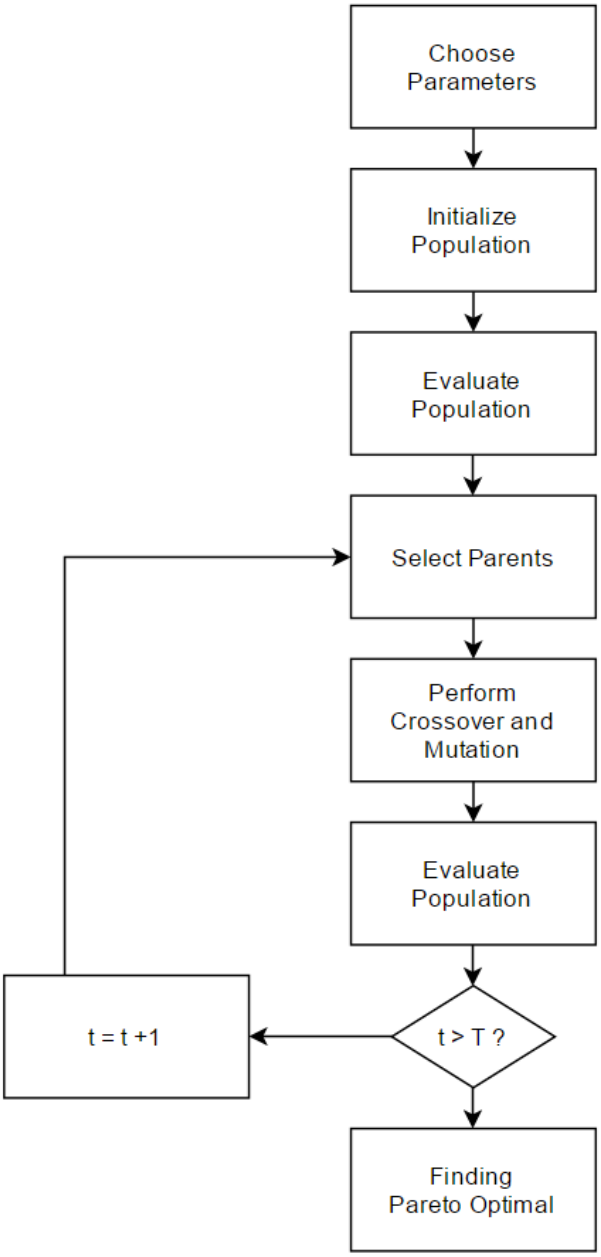



*FIG. 1 - GENETIC OPTIMIZATION WORKFLOW*

## Getting started

For better understanding of this handout, a basic knowledge of Dynamo and Revit is advised.

A comprehensive list of learning resources is listed at the end of this handout.

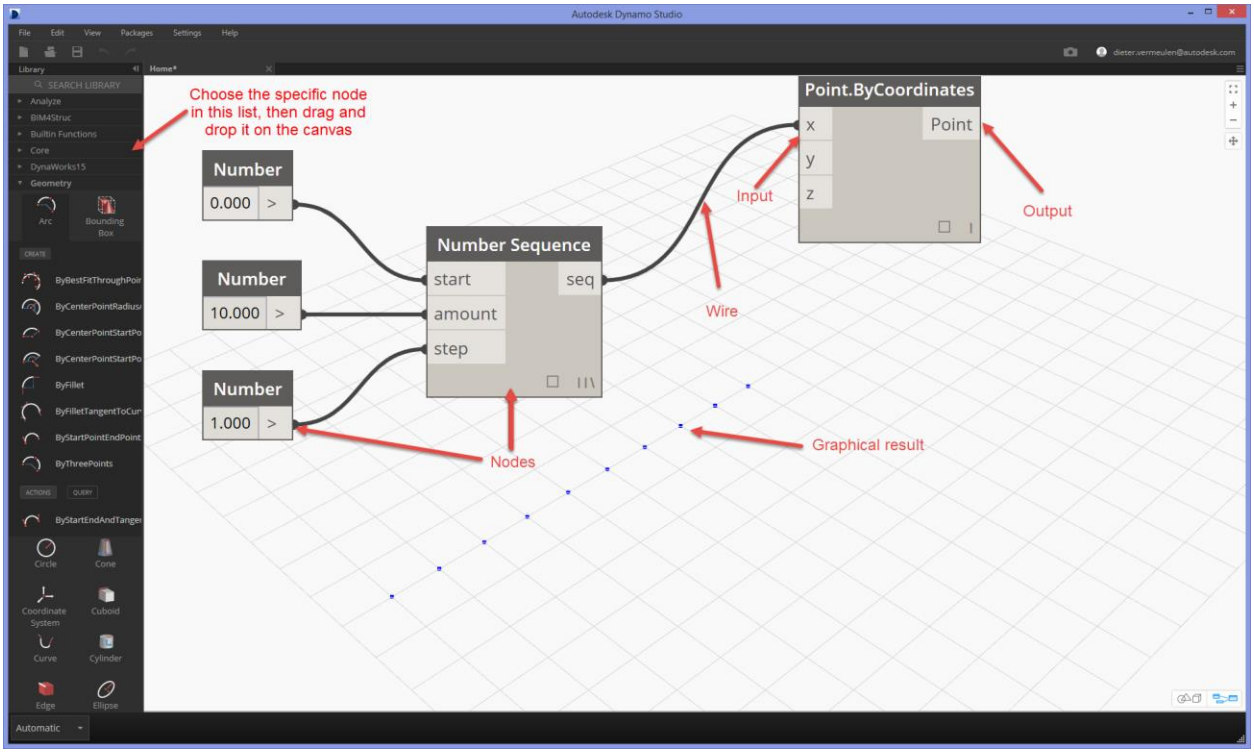



*FIG. 2 - DYNAMO GRAPHICAL USER INTERFACE*

# Used software

## Autodesk Software

The software that is used to work out the examples in the next chapters:

- **Autodesk Revit 2017.1**
  Used for setting up the base models
  More info on: http://www.autodesk.com/products/revit-family/overview

- **Autodesk Navisworks Manage 2017**
  Used in this class for performing clash detections.
  More info on: http://www.autodesk.com/products/navisworks/overview

- **Autodesk Dynamo 1.2.0**
  Free open-source tool used for computational design approach of the presented construction problems.
  More info on: http://www.dynamobim.org

- **Dynamo Studio 2017 v.1.2.0.2871**
  Dynamo Studio software is a stand-alone programming environment that lets designers create visual logic to explore parametric conceptual designs and automate tasks.
  More info on: http://www.autodesk.com/products/dynamo-studio/overview

- **Project Fractal Alpha**
  Cloud based platform to explore the parametric design space of models created in Dynamo Studio with the automatic generation of a wide sampling of options.
  More info on: https://home.fractal.live/

## Dynamo Packages

- **Optimo for Dynamo v. 0.1.2**
  Free package (plugin) for Dynamo with Genetic Algorithm nodes for the optimization of design and analysis.
  More info on: https://github.com/BPOpt/Optimo/wiki/0_-Home

- **DynaWorks17 v. 1.1.1**
  Free package for Dynamo which contains the binary files for using Navisworks in Dynamo. You will need Dynamo and Navisworks Simulate or Manage to use this library.
  More info on: https://github.com/Gytaco/DynaWorks

- **BIM4Struc.CraneAnalysis v 1.2.0**
  Free package containing custom nodes for performing crane analysis in a Revit model, in order to define the ideal position of one or more tower cranes. This is explained in this chapter of this handout.

The Dynamo packages can be found in Dynamo, by performing the next steps:

1. In Dynamo, go to Packages > Search for a Package.
2. In the next dialog find the appropriate package and install using the ‘arrow’ button.

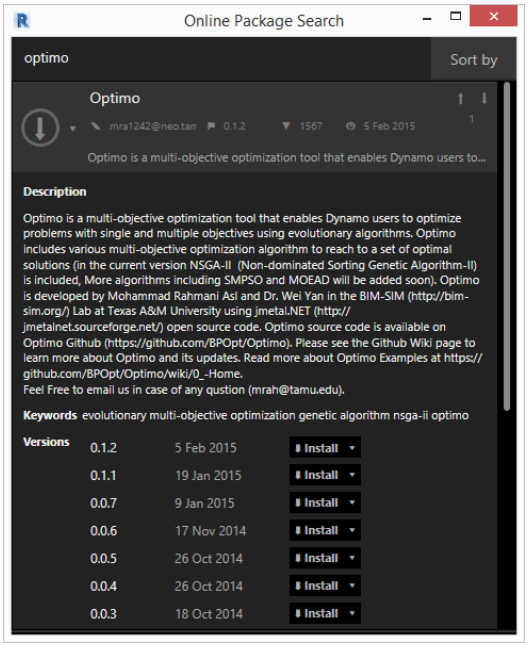



*FIG. 3 - DYNAMO PACKAGE MANAGER*

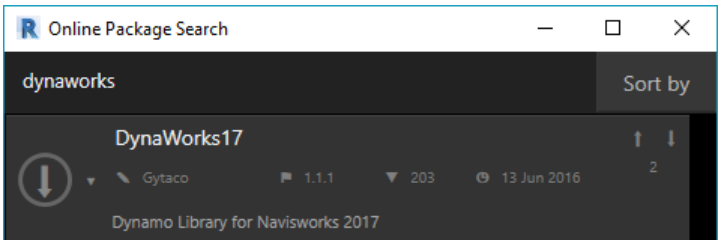
# LIVE CLASH DESIGN VERIFICATION

## Live Design Clash Verification

In this example you will learn how to stream clash results from Navisworks live into your Revit design, and see how design changes affect the clash tests with Dynamo and the DynaWorks package.

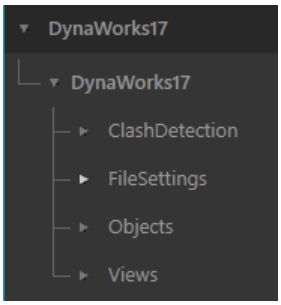
### DynaWorks

Dynamo provides a direct access to the Revit API and offers possibilities for advanced geometry creation, data manipulation and automation of Revit tasks. Dynamo comes with a set of default nodes, which can be extended by the downloadable packages provided by the Dynamo community. One of the packages especially interesting for the construction industry is the DynaWorks package by Adam Sheather, which provides access to the Navisworks API thus connecting Revit and Navisworks in a completely new way.




Please note that there are different versions of DynaWorks for Navisworks 2015 / 2016 / 2017.

Once you have installed DynaWorks, you can find following sections in your Dynamo Library:




As the sections above are already showing, there are four main areas where DynaWorks will help you:

**ClashDetection:** with these functions you can bring back your clashes from Navisworks to Revit, e.g. to color the elements which are clashing or to append comments from the Navisworks clash report to the appropriate Revit element.

**FileSettings:** provides nodes for opening, appending and saving Navisworks files through Dynamo. This can be very useful with big projects or with updates, as it can help you automatize your work.

**Objects:** offers access to Navisworks attributes, which can then by synced with the Revit model

**Views:** can retrieve saved views from Navisworks and bring them back to Revit.

DynaWorks will retrieve the results of your clash tests set up in Navisworks including all information like status, comments etc. This information can then be used to place “Clash Indicators” with these parameters in Revit and to make clash views around the point.

## Workflow

The general workflow that is used in this example is following the next steps:

1. Append the Revit models to a new Navisworks project and set up the Clash Tests
2. Modify/Add design in the Revit model(s).
3. Run the Dynamo script. This will change the “Clash” property of the analyzed elements, create separate sectioned 3D views showing each clash and place an indicator (by means of a 3D arrow with attributes) in the Revit model at the clash points.
4. Make new changes and run the script again. This will remove the clash views and indicators of resolved or approved clashes and generate new views and indicators for the new clashes.

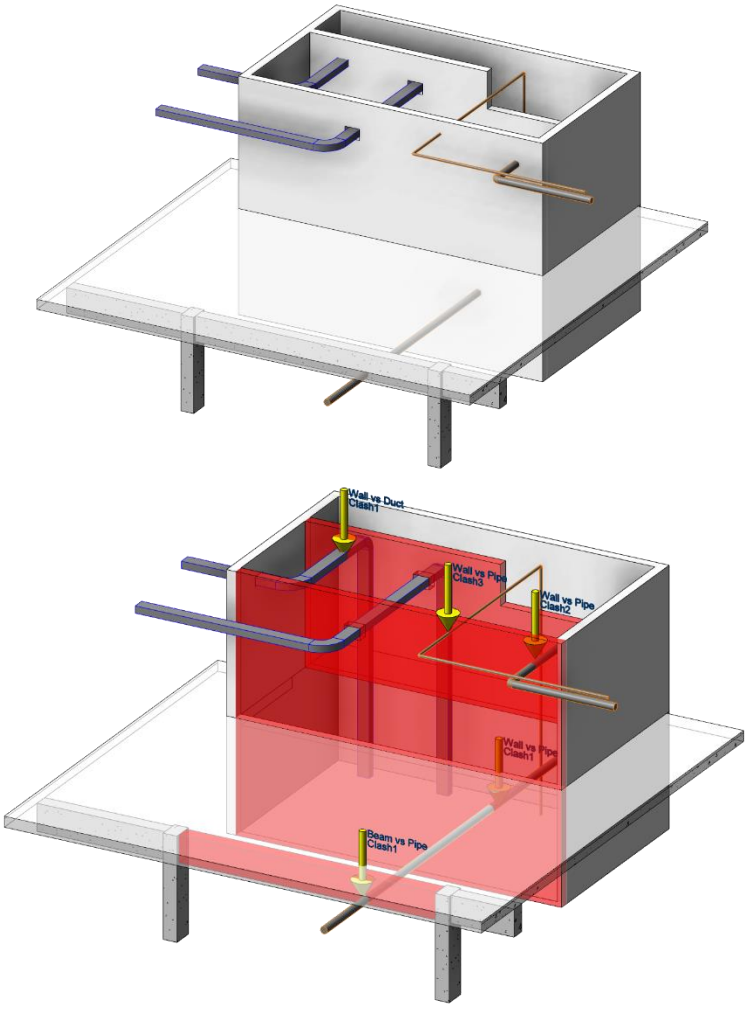
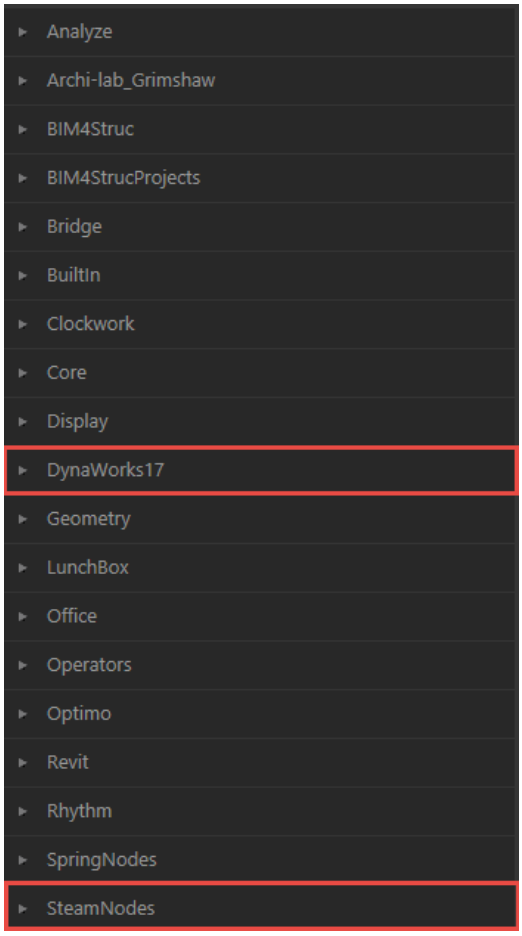



*FIG. 4 - RESULT IN REVIT BEFORE & AFTER CLASH REVIEW*

## Pre-requisites

Before you continue with this chapter, make sure the next two packages are installed in Dynamo (read more here).

1. DynaWorks17 – used for streaming the clash results from Navisworks to Dynamo.
2. Steamnodes 1.0.0 – used for the view creation in Revit.



## Datasets

The datasets for each product that are used for this script are listed below:

### DATASETS

**REVIT**

Models:
*Structural Model.rvt*
*MEP Model.rvt*

**NAVISWORKS**

Coordination model:
Coordination Model.nwf

**DYNAMO**

Workspace:
*Live Design Clash Check.dyn*

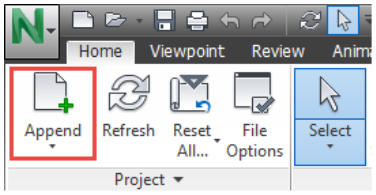
Custom nodes & Packages
*Document.Save.dyf*
*DynaWorks17*
*Steamnodes 1.0.0*

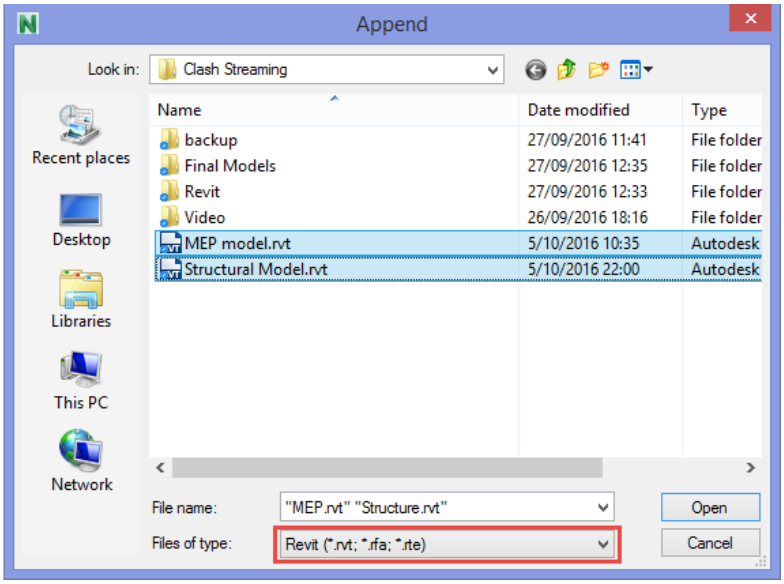
## Set up the Navisworks model

The coordination model needs to be set up in Navisworks Manage by following the next steps.

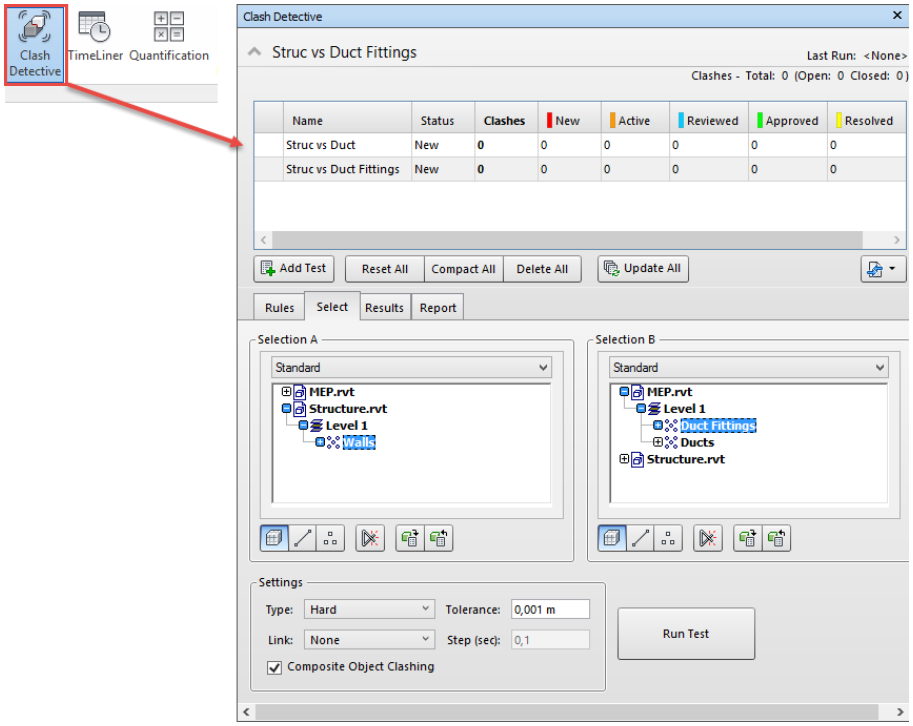
**(1)** Create a new project in Navisworks Manage 2017.

**(2)** Append the Structural Model.rvt and MEP Model.rvt as Revit file types to the project.



**(3)** Set up the appropriate clash detection tests in the Clash Detective panel

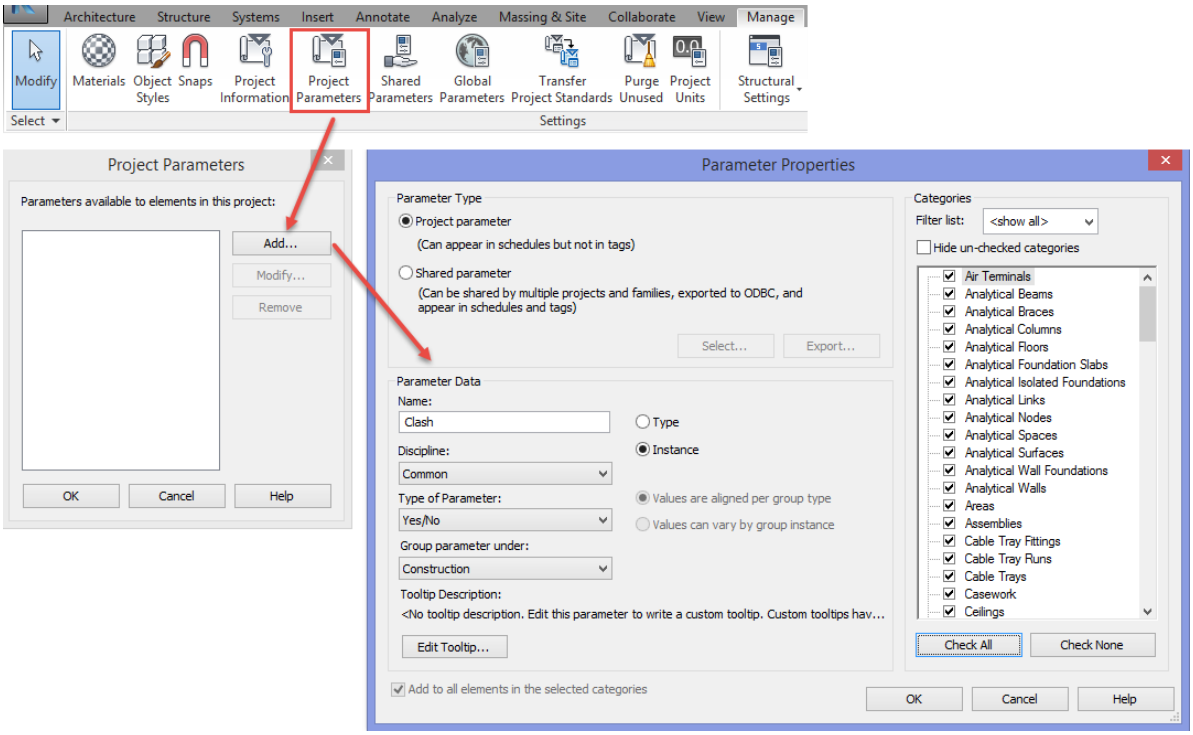



**(4)** Save the model to “Clash Detection.nwf” in the same folder as the Dynamo script.

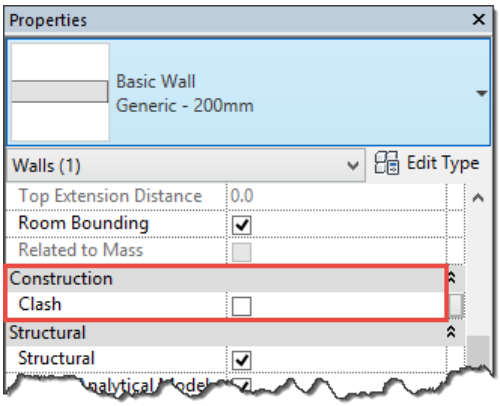
## Set up the Revit model

To ensure compatibility with the Dynamo script some things should be set up in the Revit model before running the script.

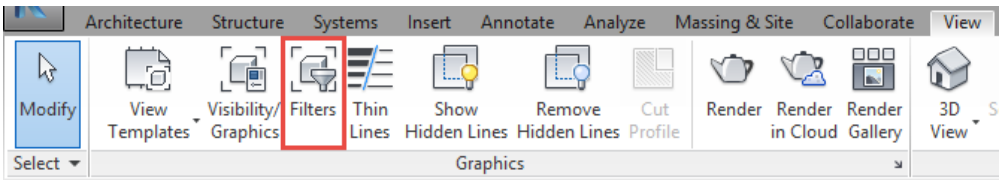
**(1)** Create a project parameter called “Clash” as a Yes/No parameter to hold the results from the Dynamo “clash stream” to Revit.

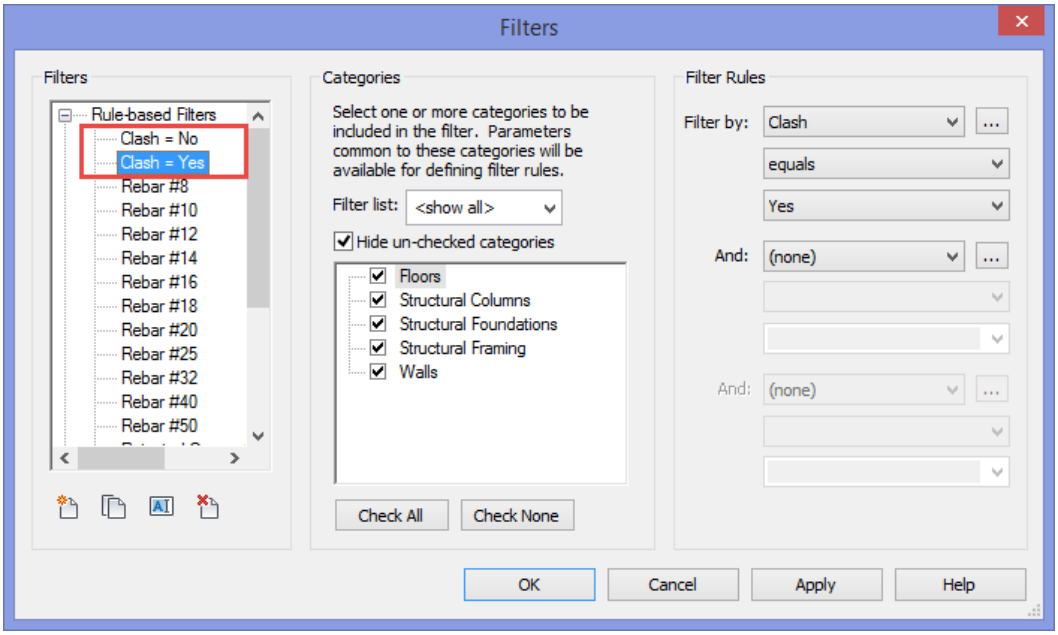



This will add this parameter to all indicated categories.

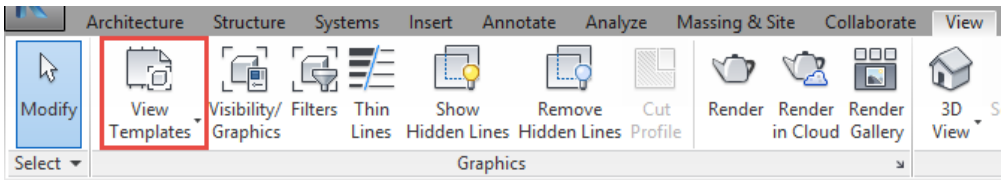


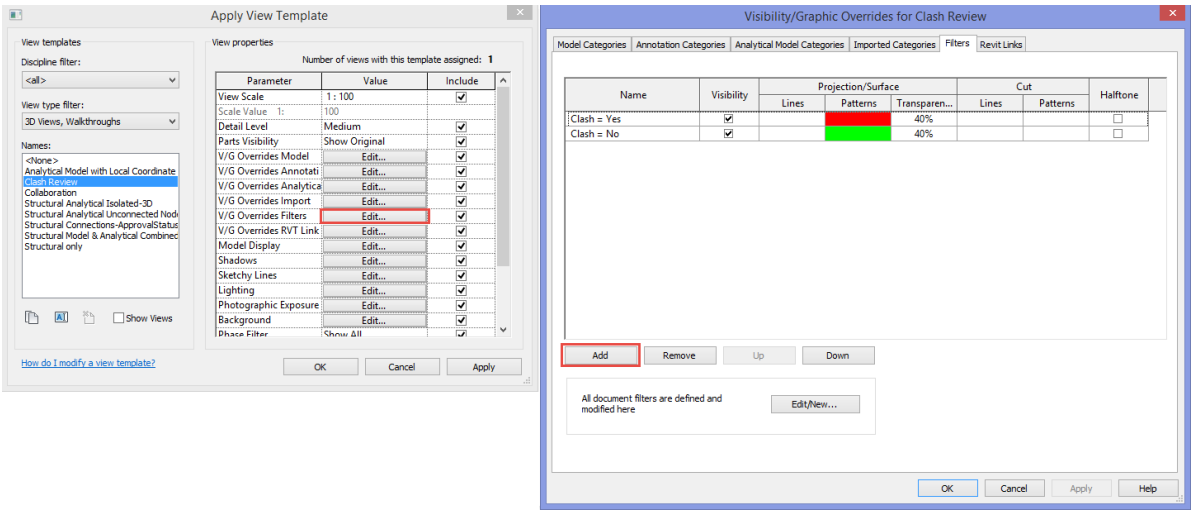
**(2)** Create *View Filters*, based on the “Clash” parameter to visualize the clashing elements based on the value of the parameter. This view filter will be used in *View Templates*.



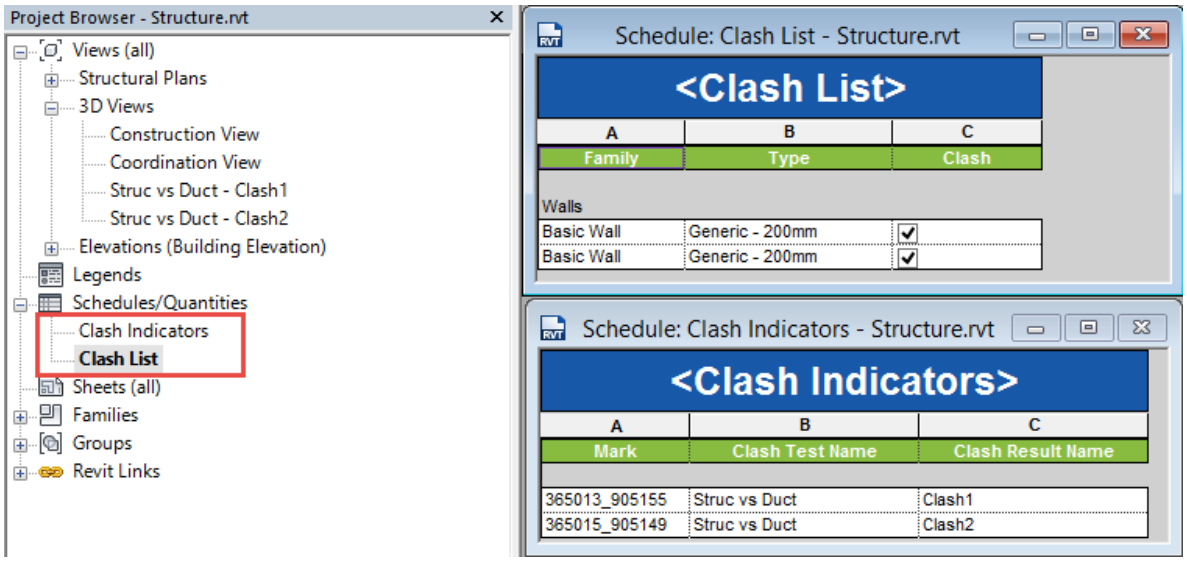




**(3)** Create a *View Template*, which holds the settings for the visualization of the *View Filter*.

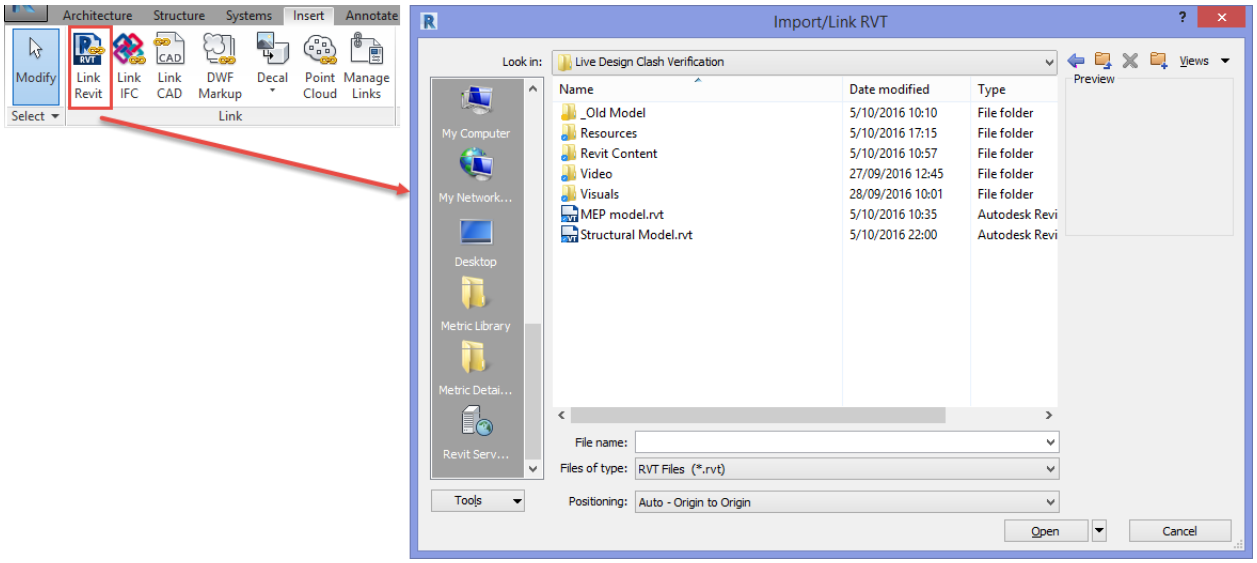


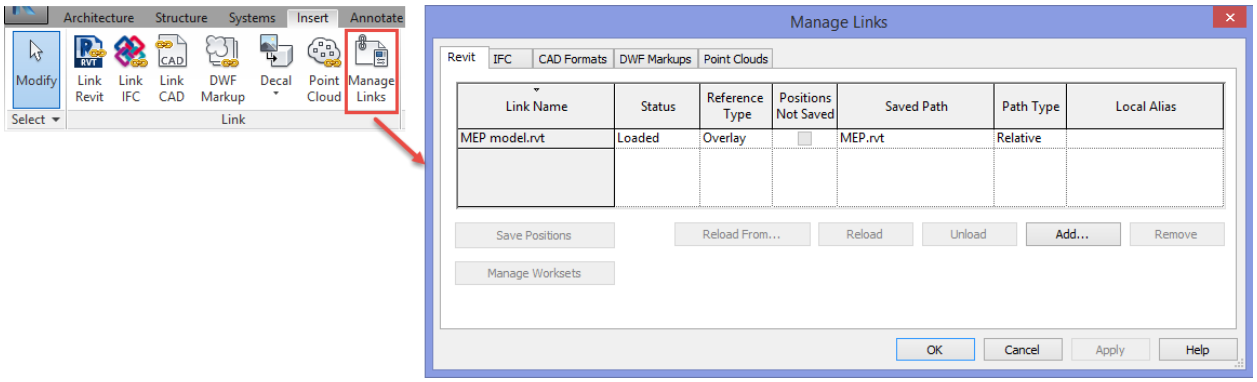



**(4)** Additionally some *Schedules* are created to show the created *Clash Indicators* and to give an overview of the elements that are *Clashing*. The sheets are updated automatically a change is done in the properties of the elements.



**(5)** Optionally you can link the MEP.rvt model as a reference in the Structure.rvt model. But this is not necessary to make it working, as the MEP model gets coordinated through the coordination model in Navisworks.



## Set up the Dynamo script

The workflow for reading the clashes in Dynamo and return feedback to Revit is explained in the flowchart on the figure below.

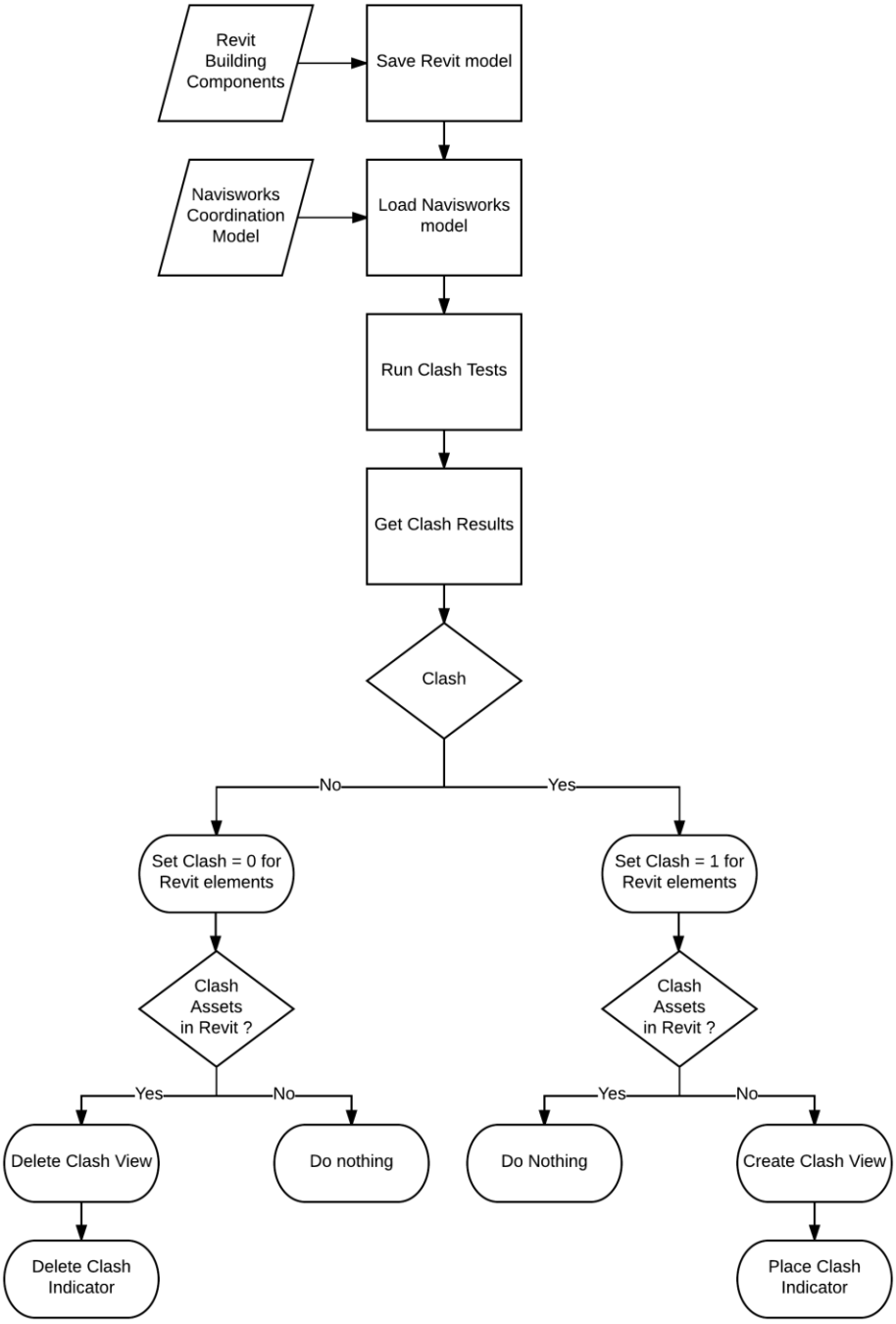
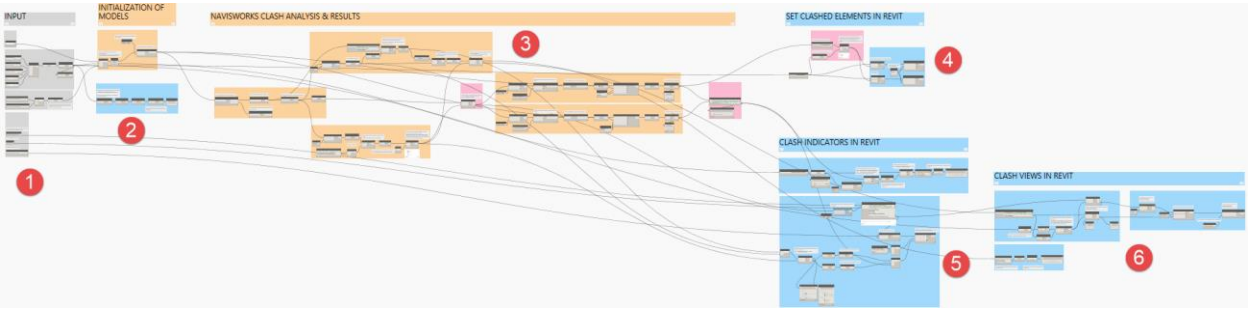



*FIG. 5 - LIVE DESIGN CLASH VERIFICATION FLOWCHART*

The script consists of the 6 big parts, which are translating the flowchart of FIG. 5 in the graph below.




**(1) Input**
User input such as reference to the Navisworks file, Revit elements to analyze, setting up family types.

**(2) Initialization of Models**
Reading the information from the Revit and Navisworks model and setting the base for the clash test run.

**(3) Navisworks Clash Analysis & Results**
Part where the clash test is run and all relevant information is read into Dynamo.

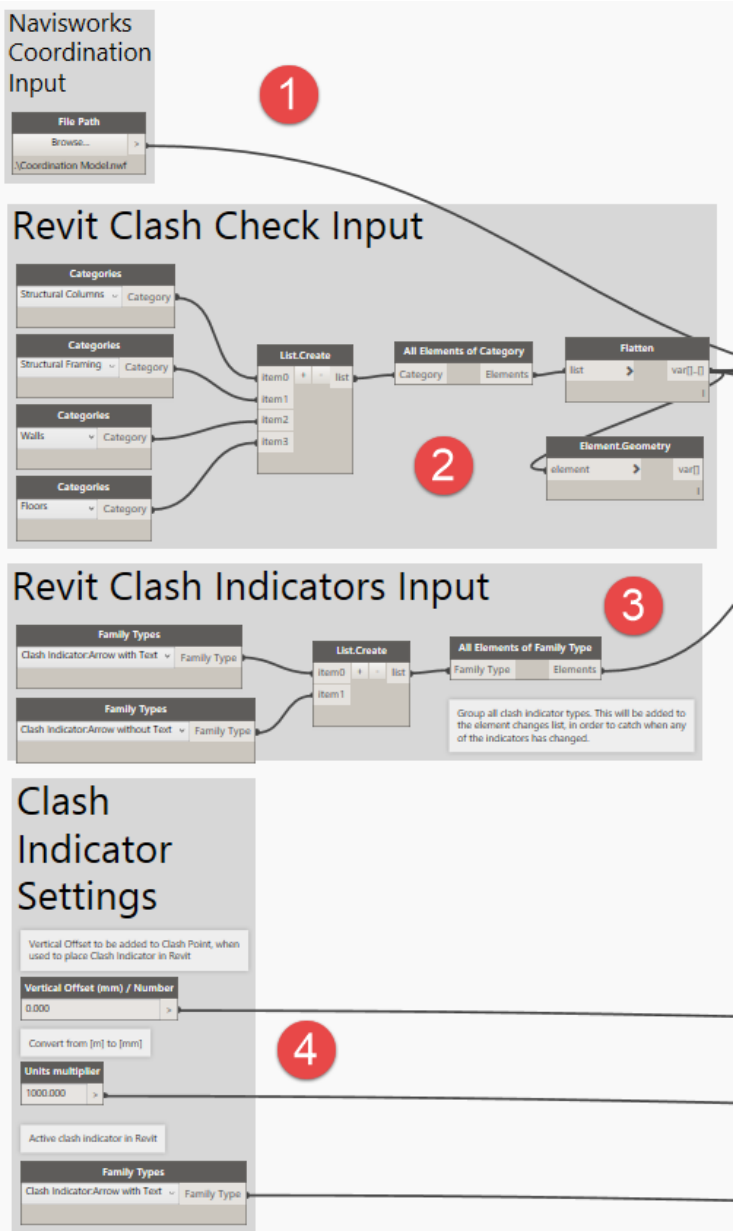
**(4) Set Clashed Elements in Revit**
Change the “Clash” parameter of the analyzed Revit elements to visualize results

**(5) Clash Indicators in Revit**
Create physical objects in the Revit model at the clash point.

**(6) Clash Views in Revit**
Create views in Revit around the clash using a Section Box.

Each of them is explained in the topics below.

*Input*




Select the Navisworks coordination model.

Select which elements should be synced with the coordination model, and will receive clash result feedback.

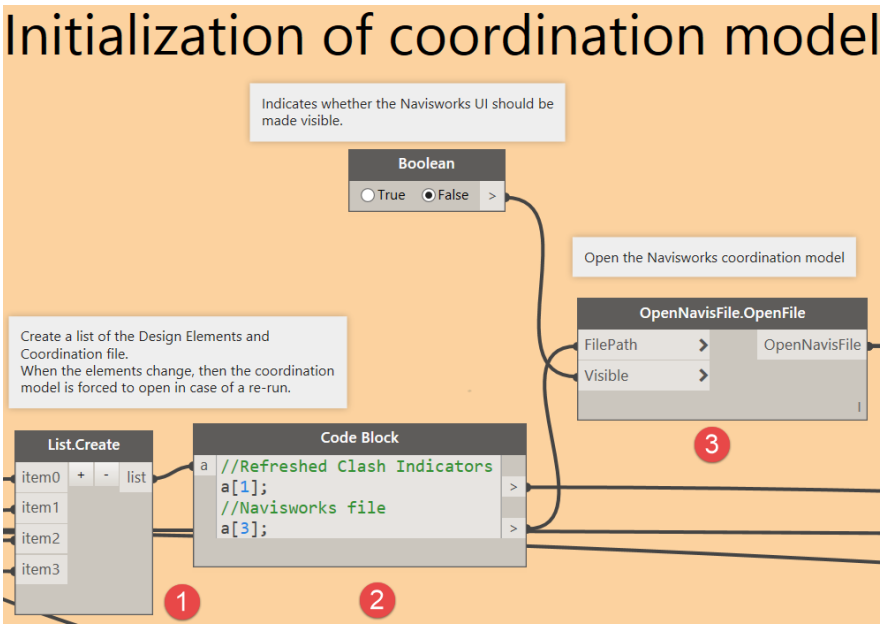
Choose between all possible clash indicator types.

Choose the appropriate settings for the clash indicators:

- The vertical offset between the detected clash point and the insertion point of the indicator.
- Units multiplier. The used DynaWorks nodes detect graphical information in metric [m], while the Revit project is in [mm].
- Finally choose the right clash indicator family and type.

### *Initialization of Models*

In this section of the script the Revit and Navisworks model get initialized for the clash detection.



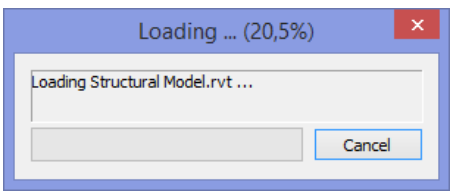

**(1)** Each time a change is done in the Revit model, the script should rerun from the start. This can be triggered by bringing all editable elements into a *List.Create* node. If one of these elements change (i.e. new indicators are placed or the geometry of wall changed), then this node gets "dirty" and needs a rerun: so the Navisworks model gets opened again and will rerun the clash, based on the updated (saved) Revit model. Everything that follows on this node will rerun then too.

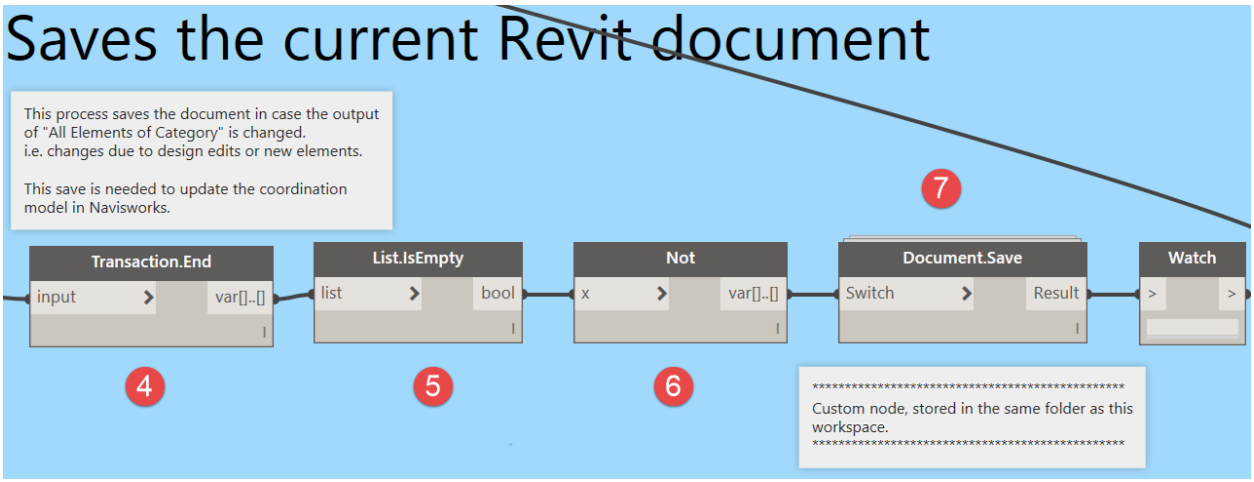
item0 = The Revit building components
item1 = All existing 'Clash Indicators'
item2 = The 'Save' status of the Revit model
item3 = The Navisworks coordination model

***Note: The order the inputs get into the List.Create are important as they will be executed in that order.*** *(i.e. Save the Revit model before the Navisworks model gets opened).*

**(2)** Break up the list again in the right order.

**(3)** Open the Navisworks coordination file. In this case the file gets opened in the background.






**(4)** With *Transaction.End* every Dynamo operation is stopped before you continue. This is needed to access the Revit API in order to save the file (step 7). The input of this node are the collection of Revit building components from the Input group.

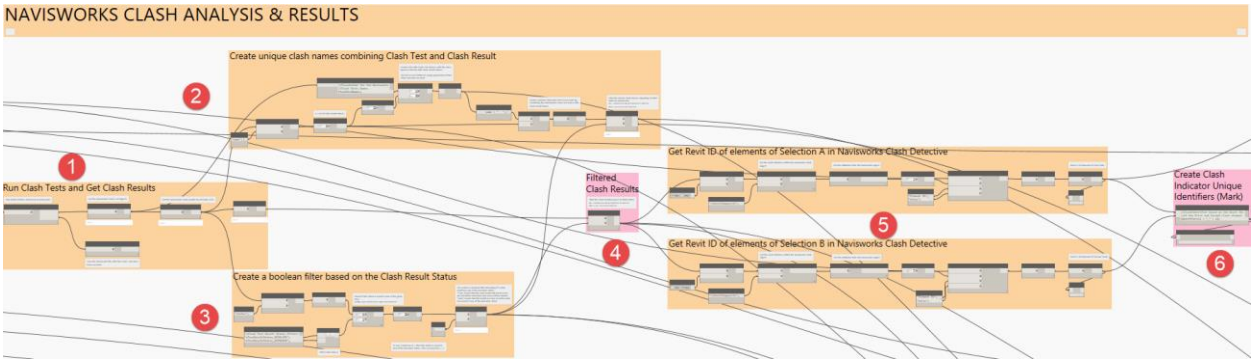
**(5)** If the list of elements is not empty then a *False* statement is returned.

**(6)** This *False* statement is reversed in order to trigger the *Document.Save* node with a *True* value.

**(7)** The *Document.Save* node is a custom node, which is delivered with the datasets. This node saves the current Revit project, when the *Switch* input equals *True* and returns if the model was saved or not.

Generally these nodes are triggered (get “dirty”) once the building components are edited and the model thus needs to be saved (and the Navisworks coordination model needs to be refreshed).

### *Navisworks Clash Analysis & Results*




This is the core part of the script, where the clash tests in Navisworks are started, and the results are filtered out.

This phase consists of 6 main groups:

**(1)** Run the clash test in Navisworks, and get the results

**(2)** Extract the Clash Test Name and Clash Result Name to create unique names (for view creation)

**(3)** Create a boolean filter that makes a difference between clashes, depending on their results

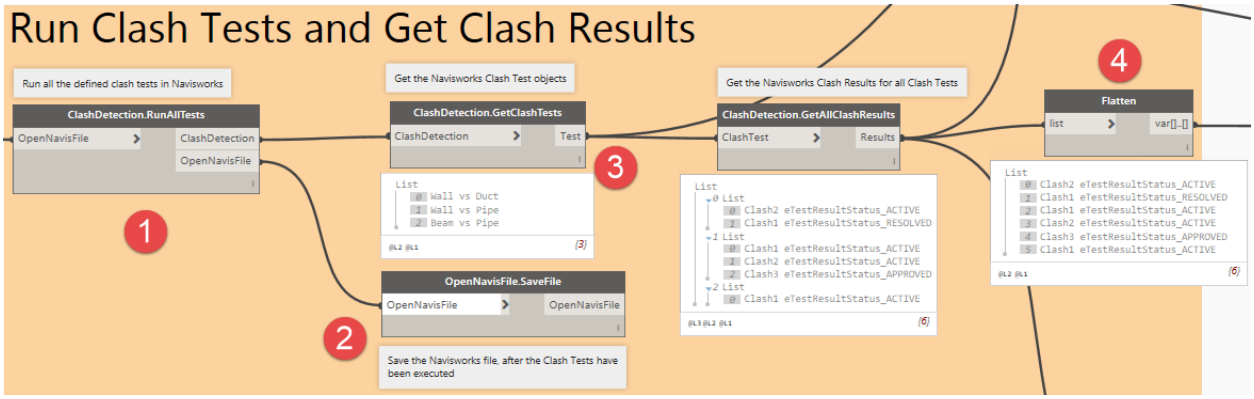
**(4)** Filter the clash results

**(5)** Get the Revit IDs of the clashing elements.

**(6)** Create a unique ID (“clash indicator”).

In the next topics these are described in depth.

### STEP 1 - CLASH TEST RUN

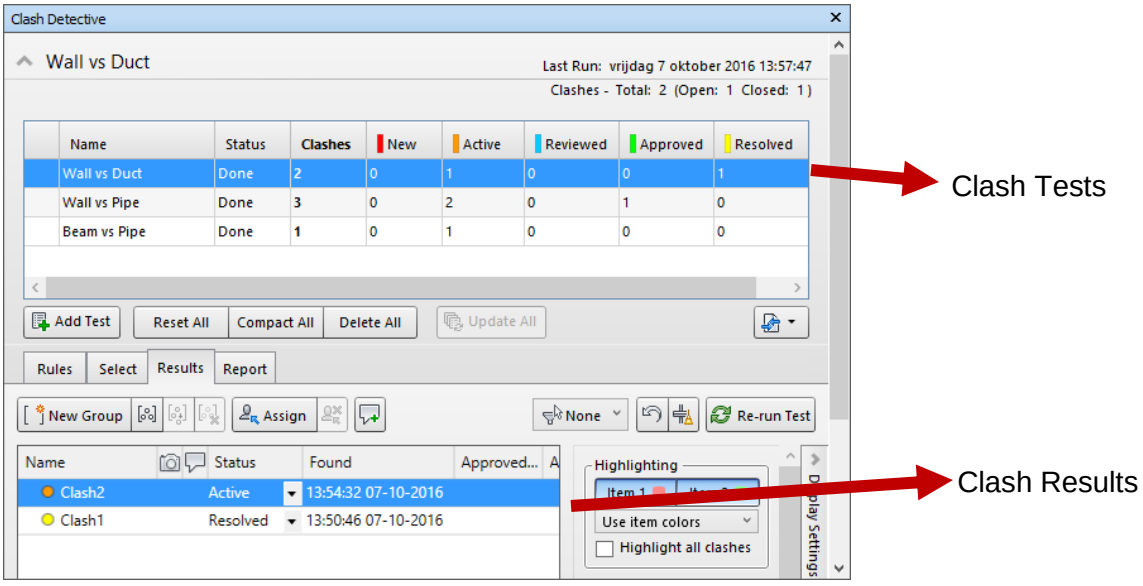



**(1)** Run the clash tests in the Navisworks model opened in the background.

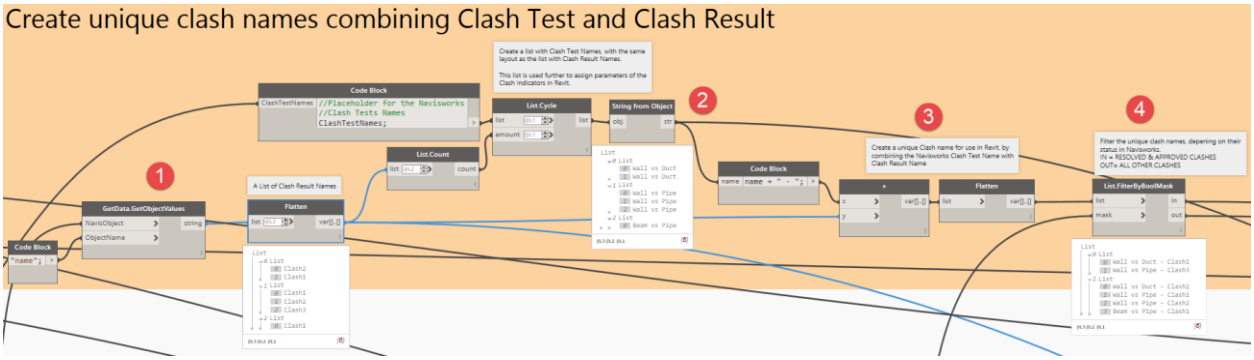
**(2)** Save the coordination model, for later review

**(3)** Get the Clash Test objects

**(4)** Get the Clash Result objects for all clash tests.



## STEP 2 – CREATE UNIQUE NAMES BASED ON THE RESULTS



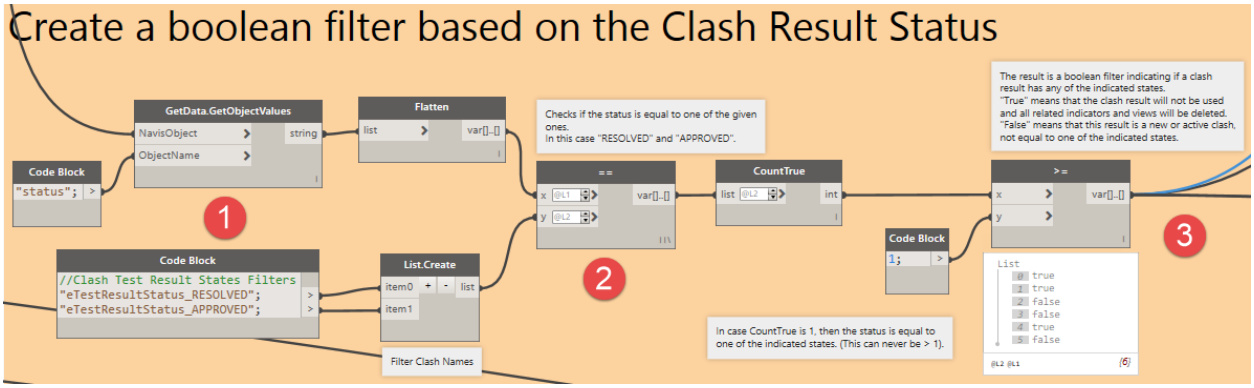

**(1)** Get the name of the Clash Result in Navisworks (Clash1, Clash2, …)

**(2)** Get the name of the Clash Test in Navisworks from (3) in Step 1.

**(3)** Combine the names into one unique text string, which will be used as view name when creating clash views. See later in this chapter.

**(4)** Filter the unique view names based on the Boolean filter, defined in Step 3.

## STEP 3 - CREATE A BOOLEAN FILTER

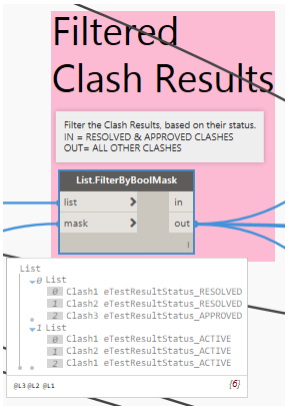



**(1)** Get the *Status* property of a Clash Result and indicate in the Code Block which clash states you want to filter out, combined in a list.

**(2)** Compare the resulted status names with any of your indicated ones.

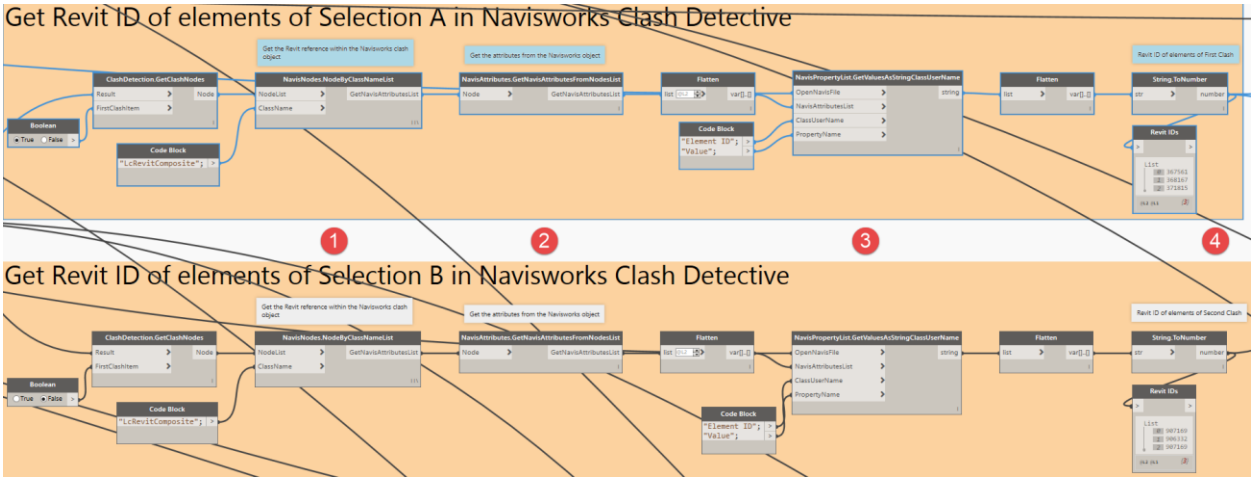
**(3)** The result is a boolean list, telling which clash result should be filtered out. True means the clash meets one of state filters (resolved or approved). False means the clash result is a new or active one.

## STEP 4 - FILTER CLASH RESULTS




Finally the Boolean filter from Step 3, is used to filter the clash results from Step 1.

## STEP 5 - CLASH ELEMENT IDS



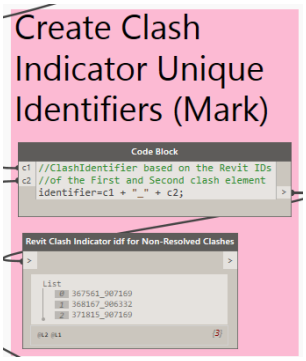

**(1)** Get the Revit reference with the Navisworks clash object from the clash results (from DynaWorks17)

**(2)** Get the attributes from the Navisworks object (from DynaWorks17)

**(3)** Get the "Element ID" of the clash object

**(4)** The result are the Revit IDs in a number format of the clashing elements.
A = Structural Model
B = MEP Model

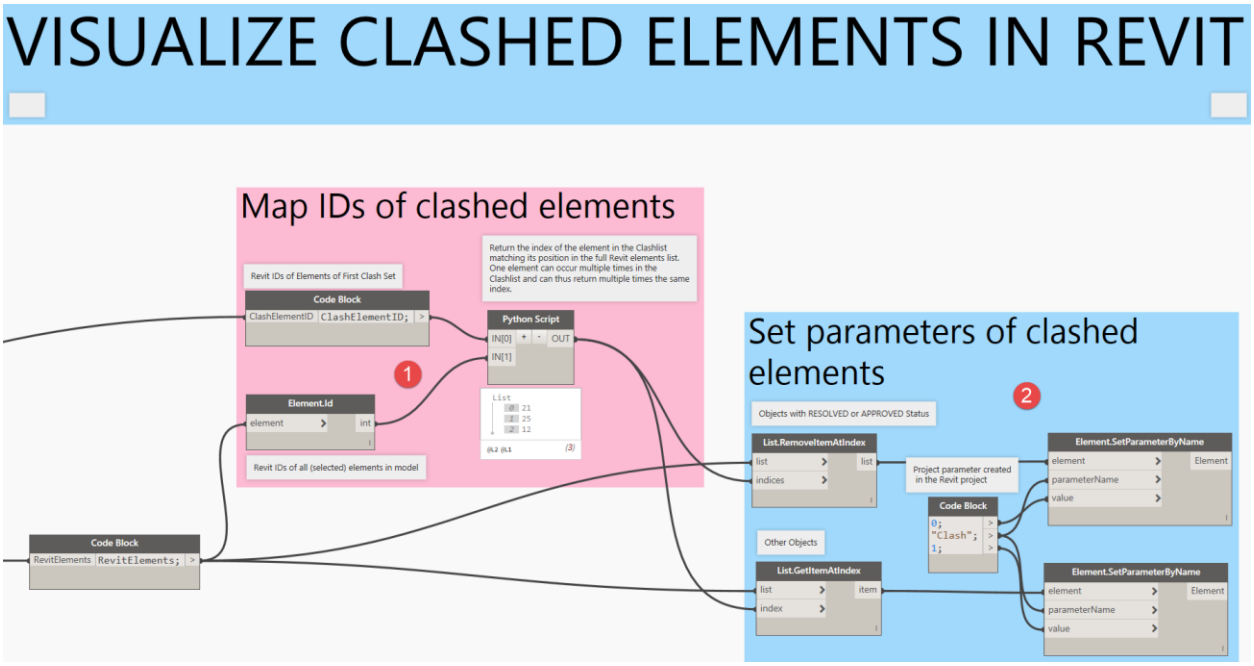
### STEP 6 - CREATE UNIQUE ID




The resulting IDs from Step 5 are combined to create a unique ID. This ID is used as value for the *Mark* parameter of the "clash indicators".

As the *Mark* parameter needs unique input this is the best way to get a unique ID as 2 objects can only be detected as 1 clash.

*Visualize Clashed Elements in Revit*

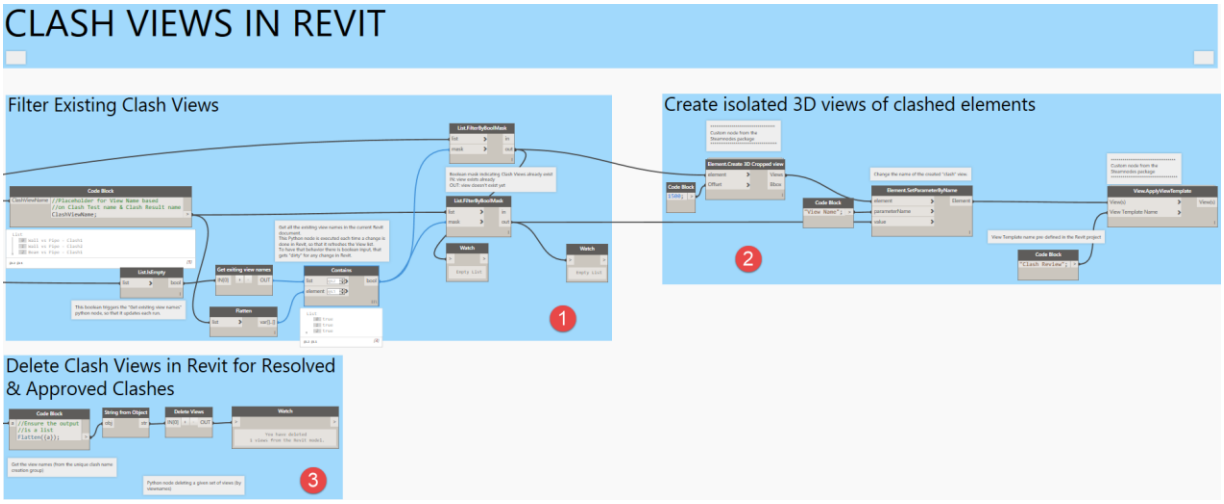



**(1)** In the first part the resulting Revit IDs from the clash tests, defined in Step 5 of previous chapter, are compared with the Revit IDs of all verified elements (defined in the Input phase). The *Python Script* node returns the index of the element in the Clash list matching its position in the full Revit elements list. A clash can occur multiple times on one element can occur multiple times in the Clash list and can thus return multiple times the same index.

**(2)** The resulting indexes are then used to filter the Revit elements. On top the indexes are used to only remain the elements without a clash result or an approved or resolved clash. On the bottom only the clashing elements are taken.

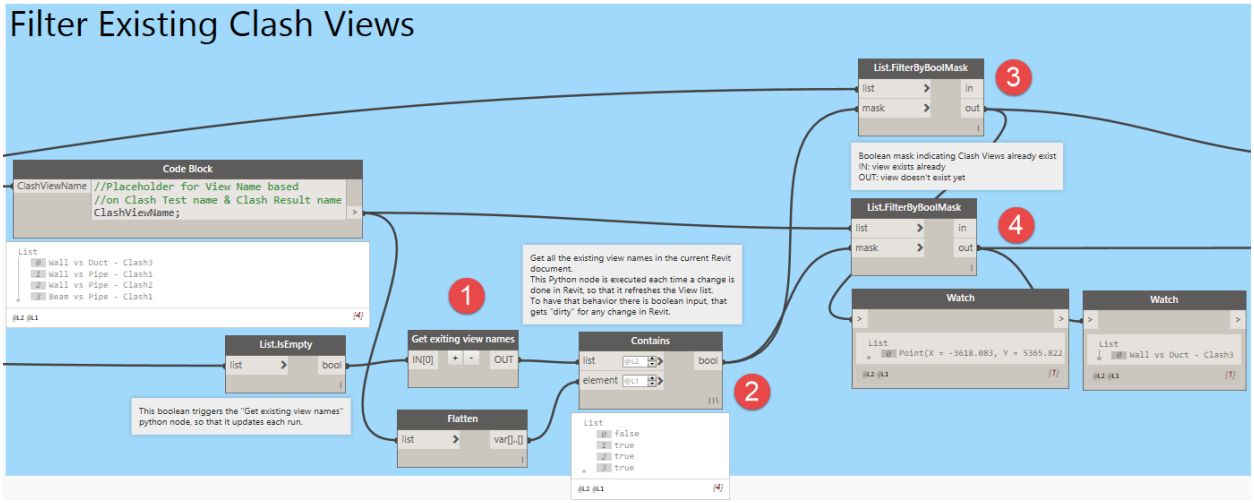
In the Revit project there is a project parameter called “Clash” assigned to different categories (read here). Depending on the elements clash status the *Clash* parameter gets the value 0 (false) or 1 (true).

*Clash Views in Revit*




In this phase a 3D view with section box around the clash point is created, showing the clashing elements and the clash indicator. This phase needs the Steamnodes package to create the 3D views. The views may only be generated if they don't exist yet for that clash. And when a clash is filtered as "Resolved" or "Approved" then the clash view needs to be deleted, if exists.

## STEP 1 – FILTER THE EXISTING CLASH VIEWS



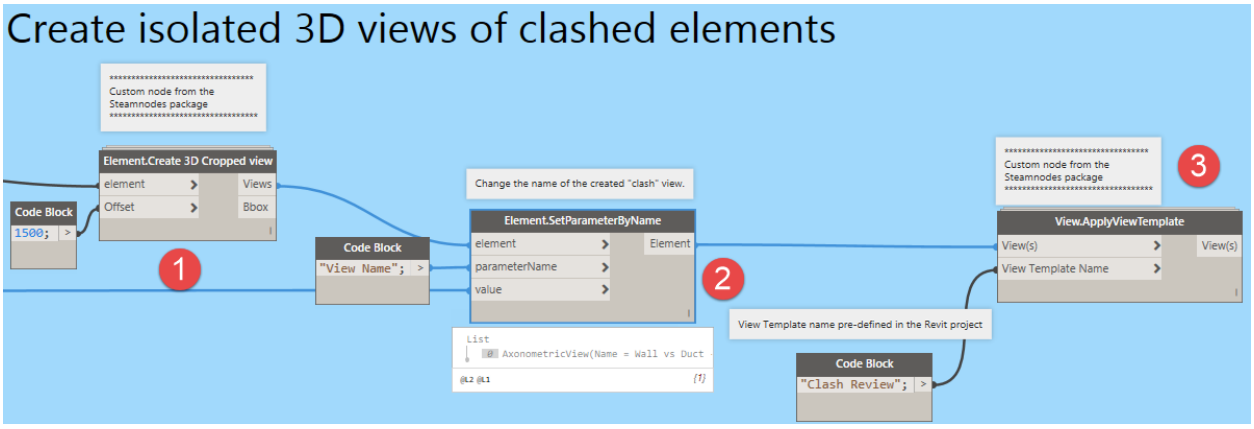

**(1)** A python script node, *Get existing view names*, is used to get all the existing view names in the Revit project.

**(2)** This list is compared with the created unique view names from the clash results. When the result is *True* then the clash view doesn't exist yet.

**(3)** The Boolean filter from (2) is now used to filter the clash points (read here)

**(4)** Also the views names are filtered, to keep only the new to-create ones.

### STEP 2 – CREATE CLASH VIEWS

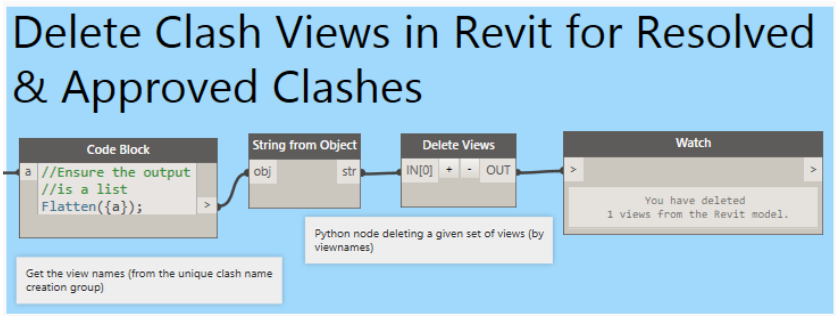



**(1)** Create a 3D cropped view around the filtered clash points, defined in Step 1 (3). This node is part of the Steamnodes package, which you need to install prior to running this script.

**(2)** Set the view name of the 3D clash view to the unique view name that was created previously.

**(3)** Optionally apply a view template to set the view representation. This view template needs to be available in the Revit project (read here).

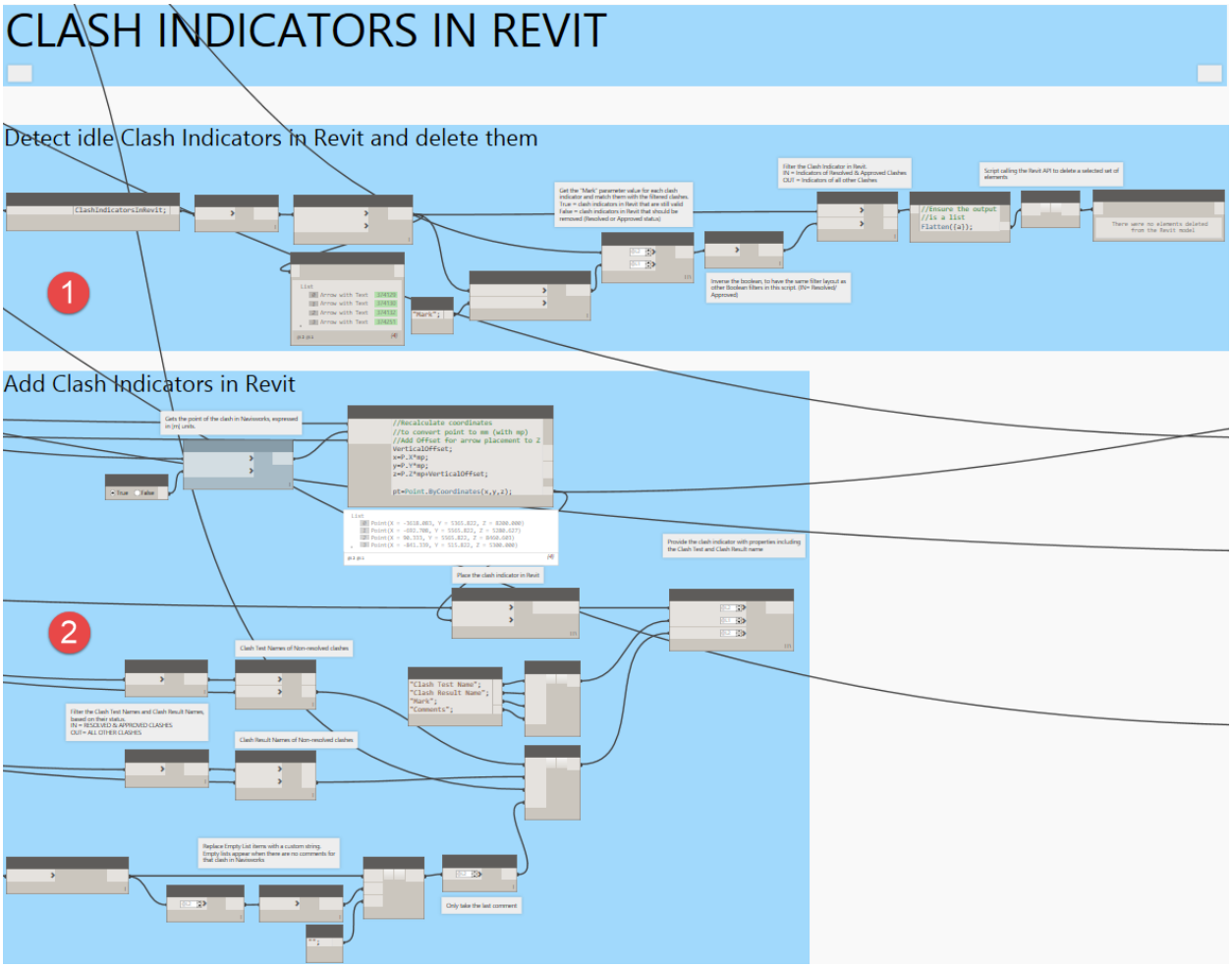
### STEP 3 – DELETE IDLE CLASH VIEWS



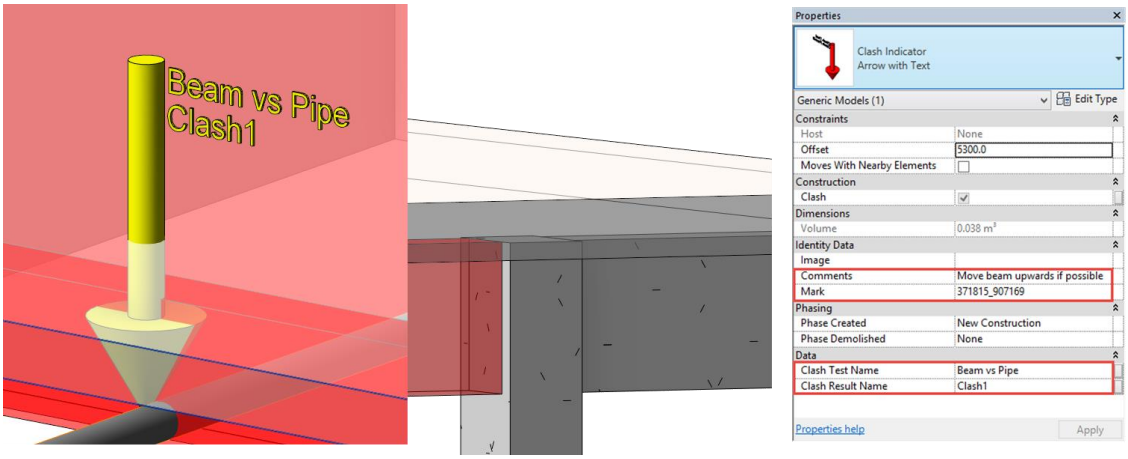

This step uses a *Python node* called “Delete Views”. The code calls the Revit API to delete a given set of views. In this case the views for “resolved” and “approved” clashes are removed.

These view names are defined and filtered previously in this step.

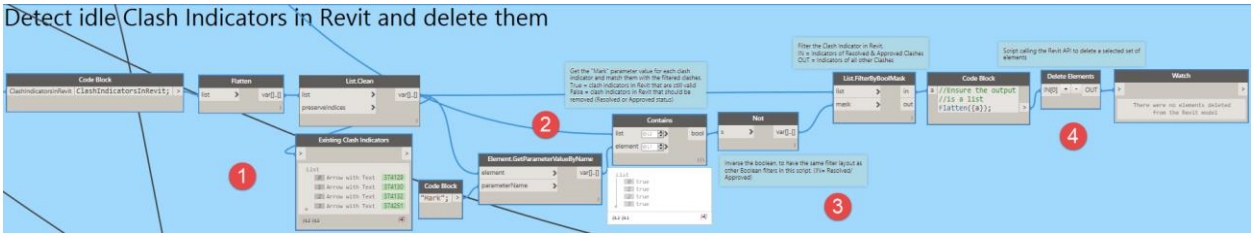
### *Place Clash Indicators in Revit*




To visualize the clashes better in the Revit project, a “Clash Indicator” is placed on the resulting clash point from Navisworks. These indicators are also provided with parameters, which can be scheduled.



## STEP 1 – DELETE IDLE CLASH INDICATORS



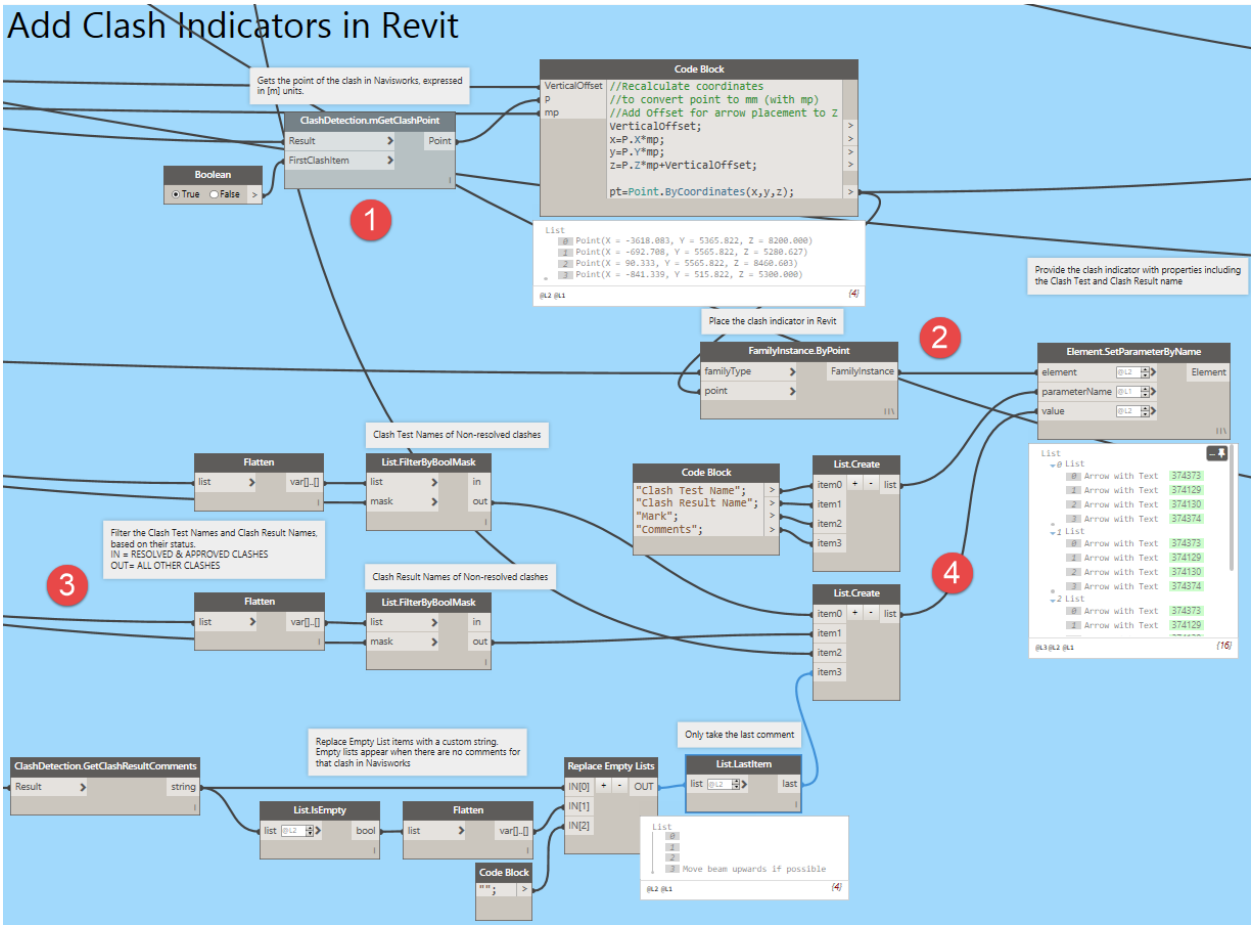

**(1)** Get all the clash indicators

**(2)** Get the value of their *Mark* parameter and compare them with the output of the unique IDs that where generated previously (read here). Remark the list levels that are set in the *Contains* node.

**(3)** The resulting boolean values are inversed, to have the same filter layout as other boolean filters in this script. (IN= Resolved/Approved, OUT= New/Active)

**(4)** The python node *Delete Elements* calls the Revit API to delete all idle clash indicators, corresponding with clashes that are set to the “Resolved” or “Approved” status.

## STEP 2 – CREATE CLASH INDICATORS IN REVIT




**(1)** Get the clash point of the elements clashing in Navisworks. Remark that in this script the metric version of the node is used. The resulting points are expressed in [m], so the values are converted to the project unit, [mm], in the Code Block.

**(2)** The points are used to place the Clash Indicators in the Revit model

**(3)** Collect the properties of the clash to add to the clash indicator parameters.

a. The first property is the name of the Clash Test, previously defined.

b. The second item is the name of the Clash Result.

c. The third item needs to be stored in the *Mark* parameter, and used the unique ID that is generated for the clash.

d. The last item is the comment that is optionally added to the clash in the Navisworks coordination model. This is always taking the most recent comment.

**(4)** Combine the parameter names and their values with the *Element.SetParameterByName* to provide the clash indicators of custom properties.

# CRANE POSITION OPTIMIZATION

# Crane Position Optimization

## Introduction

Tower cranes are most likely the key elements of production equipment on today's building construction sites. They need to lift and transport a huge variety of elements (and thus loads). Tower cranes are submitted to lots of constraints like overlapping work zones, time, load capacity, ranges …

*Why optimizing the position of tower cranes?*

It is important for a contractor to identify the optimal number and location of tower cranes. This helps avoiding conflicts between several cranes. Possible hoisting problems can be detected before the construction gets built. And finally the contractor saves time and cost by not having an overload of cranes on site.

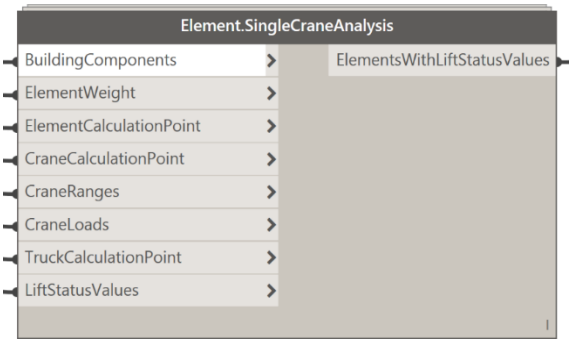
## Crane Analysis Method Description

The analysis of the range and capacity of a tower crane can be analyzed in many ways. In this particular case the analysis is based on a simplified method described in the flowchart presented in **FIG. 6**. This method calculates the *Lift Status* of each element depending on its location relative to the tower crane and truck and its weight.

The main inputs are:

- **Building Components**: A selected set of Revit elements (floors, walls, structural framing, parts …), which need to be lifted on the construction site.
- **Lift Capacity Table**: a list of values representing the loading capacity of a tower crane depending on the hoisting range.
- **Truck**: this is also called the *Supply* zone. In this example the supply zone is represented by a truck in the Revit model.

The core of the analysis in Dynamo is contained in a custom node called *Element.SingleCraneAnalysis.dyf*, which is part of the **BIM4Struc.CraneAnalysis** package for Dynamo. This chapter explains the content of the package in detail.




This custom node calculates the *Lift Status* of each Revit element in the analysis set (*Building Components*), based on the flowchart, and returns a sorted list of elements with an assigned Lift Status Value.

Additionally the lift status values for the whole construction can be evaluated into a Lift Score, representing the effectiveness of the crane position(s). Read more here how this is calculated.

When reusing the outcome as parameter values in Revit it is possible to even visualize the results in the Revit views, by means of View Filters. Read more about this in this part.

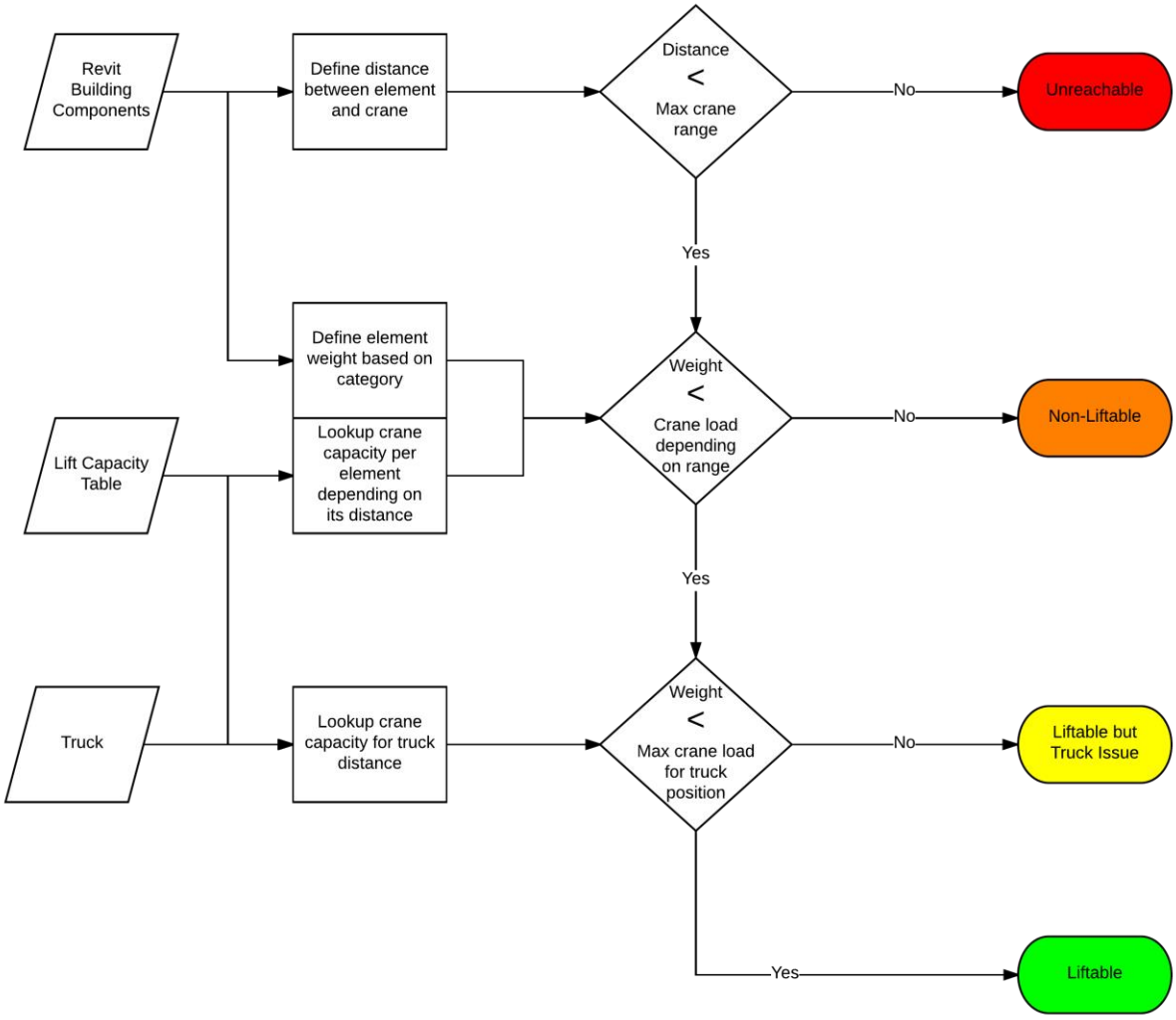



*FIG. 6 - Element Lift Analysis Flowchart*

### *Result assignment with Lift Status Values*

The Lift Status can be interpreted as follows:

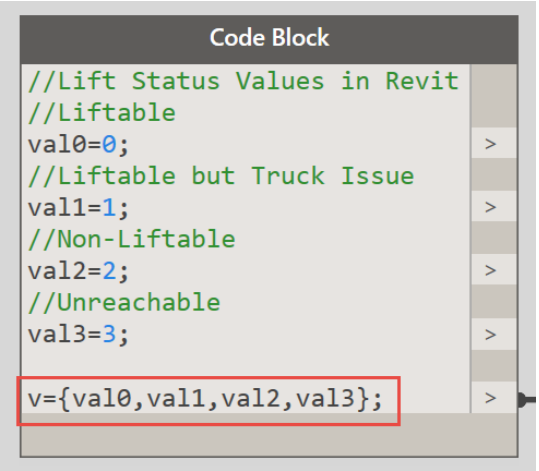
- **Liftable**:
  The element is fully within the capacity of the crane either when lifting from the truck or when putting on its place in the building.

- **Liftable but Truck Issue**:
  The element is within the capacity of the crane to be put on its place in the building, but the truck is too far away from the crane, to lift the element. So the delivery points is ok, but the supply point is out of range for the crane.

- **Non-Liftable**:
  The element is not within the capacity of the crane, and cannot be lifted due to its weight, combined with its distance to the crane.

- **Unreachable**:
  The element is too far away from the crane to be lifted.

In this project an integer is used to identify each Lift Status. These integers are then grouped into a list “v” for later use in the scripts as “LiftStatusValues”. The integers are further also used for representing the results in Revit, using view filters.

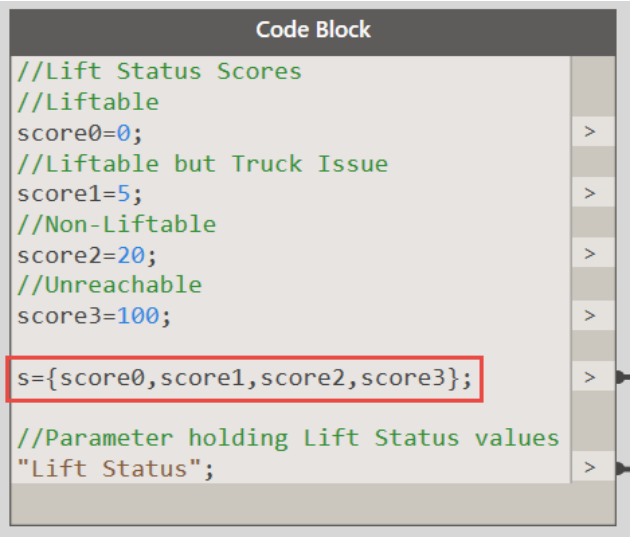



These values can also be used in the Dynamo environment to group elements according their lift status.

### *Evaluation of results with Lift Score*

In each of the methods described further in this chapter, the goal is to find a solution with a minimal *Lift Score*. This score is an indication of the optimal position of a crane in order to be able to lift all the elements in the analysis set, from the truck (supply) to its final destination in the building (delivery). The smaller the number, the better the crane is positioned. A *Lift Score* equal to “0” is the optimal one.

The *Lift Score* is defined based on the *Lift Status* of each element in the analysis set. For each *Lift Status* you assign a score and the total of scores returns the *Lift Score*.




To make it easier to find an optimal solution, it’s advised to well differentiate the *Lift Status Scores* per *Lift Status*. This means that “Liftable” elements should have the minimum score, preferable “0”, and the “Unreachable” elements should have an excessive score (i.e. “100”), as this is an unacceptable situation. The “Liftable but Truck Issue” can be solved by moving the truck, thus its influence is not that big. The “Non-Liftable” status has a bigger influence, but can be solved by splitting up the elements or by increasing the crane capacity.

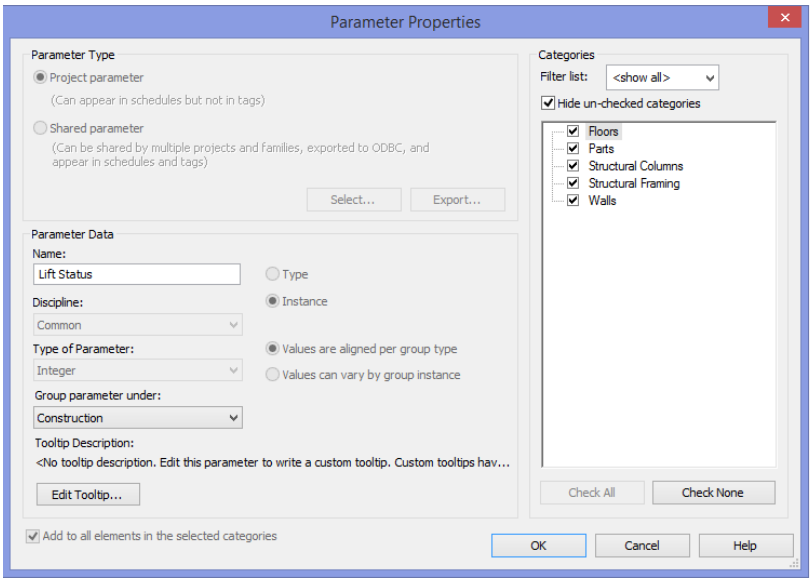
In the examples of this class the following Lift Scores are used:

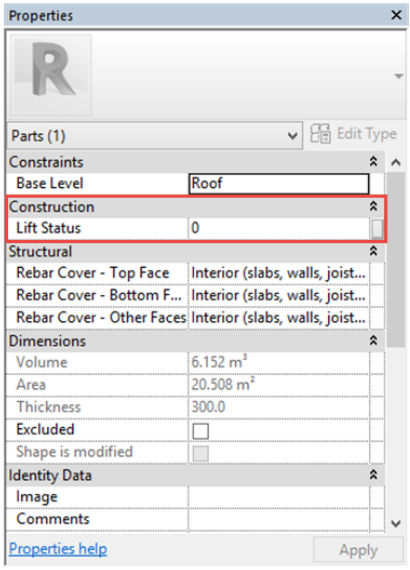
| Lift Status | Lift Status Value in Revit | Lift Score for Analysis |
|---|---|---|
| Liftable | 0 | 0 |
| Liftable but Truck Issue | 1 | 5 |
| Non-Liftable | 2 | 20 |
| Unreachable | 3 | 100 |

This is explained more in detail on the specific examples further in this handout.

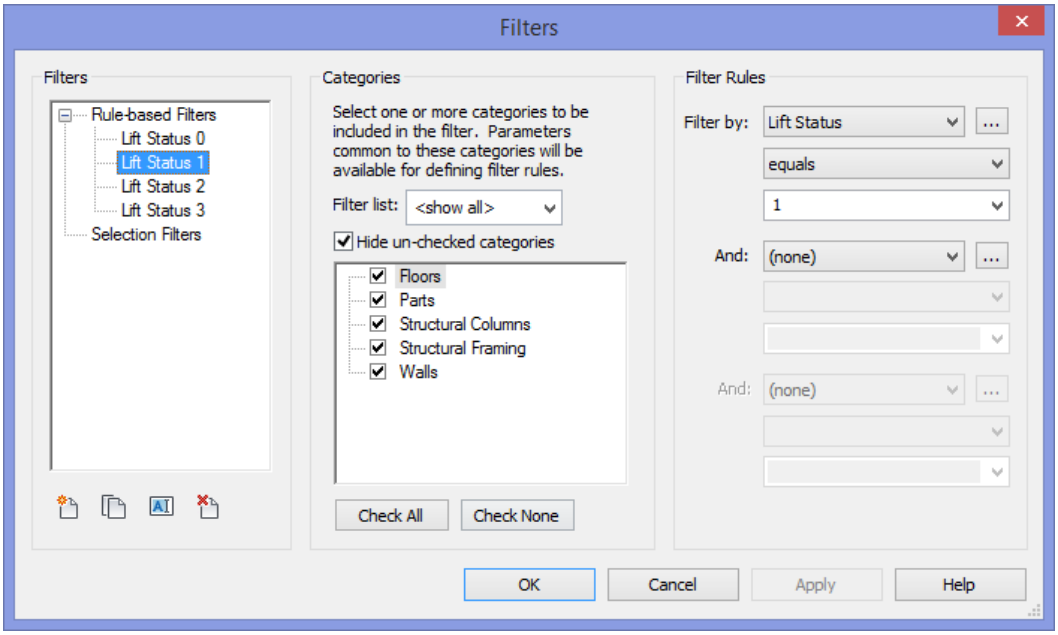
### *Representing Results in Revit*

For each of the lift states a parameter value is assigned to the element (Lift Status Value), in order to visualize the analysis results. In this project, the parameter “Lift Status” is created as a *Project Parameter* in Revit, to hold this value.

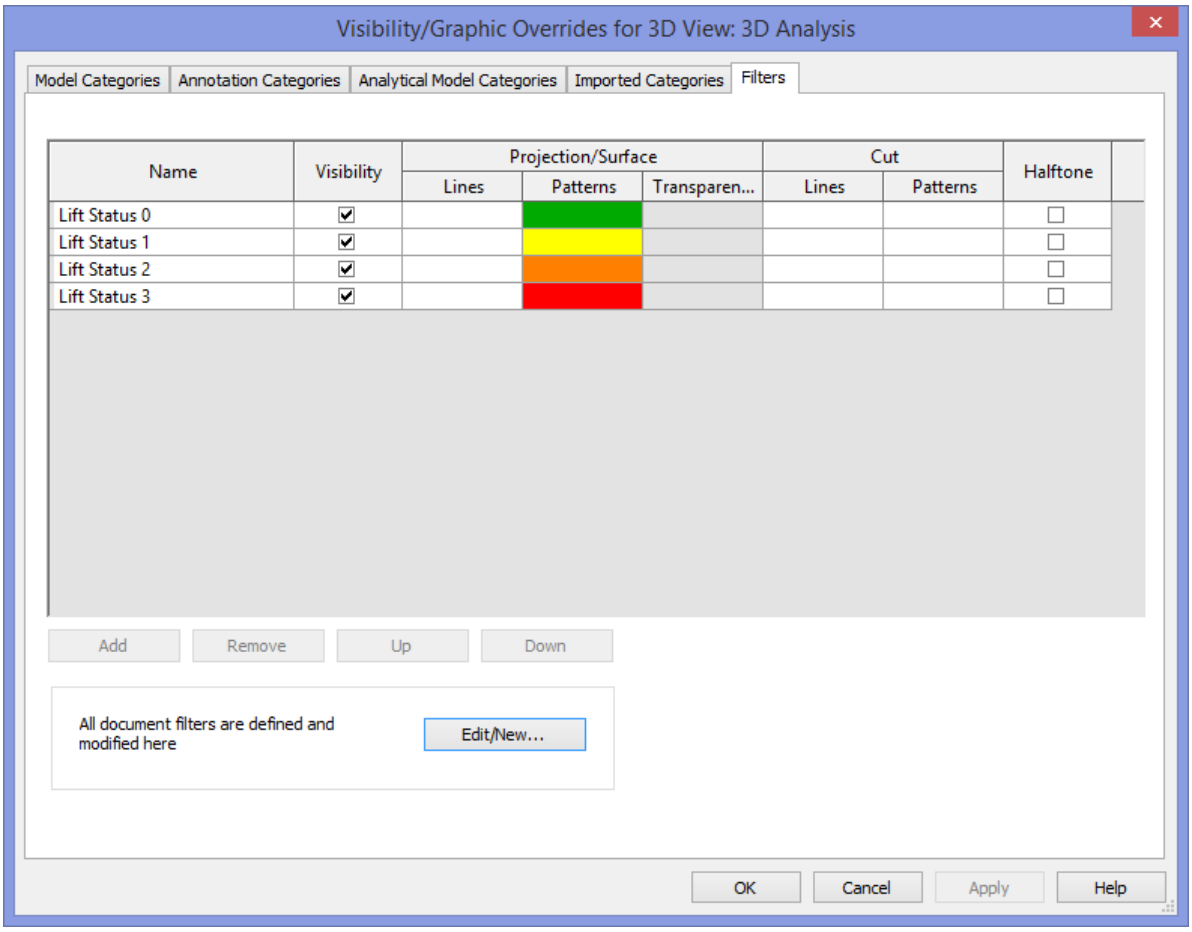



With the use of Filters in Revit, we can group the elements based on the value of the Lift Status parameter.



Per view you can create a representation configuration in Revit with the *Visibility/Graphics Overrides* and assign an override for each *View Filter*.



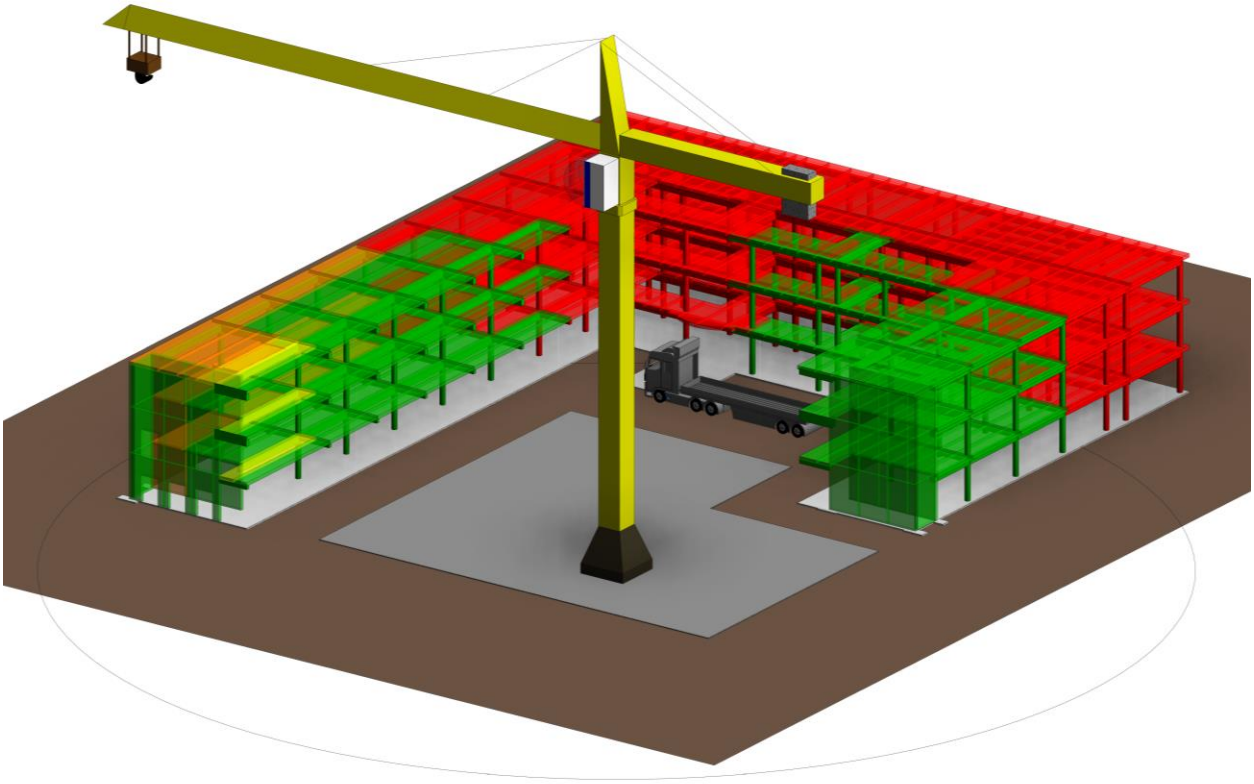


*FIG. 7 - EXAMPLE OF LIFT STATUS RESULTS REPRESENTATION*

## Crane Analysis Types in Dynamo

In this class multiple possibilities to analyze and evaluate the crane position, are considered. The examples are created for a single crane and a double crane setup.

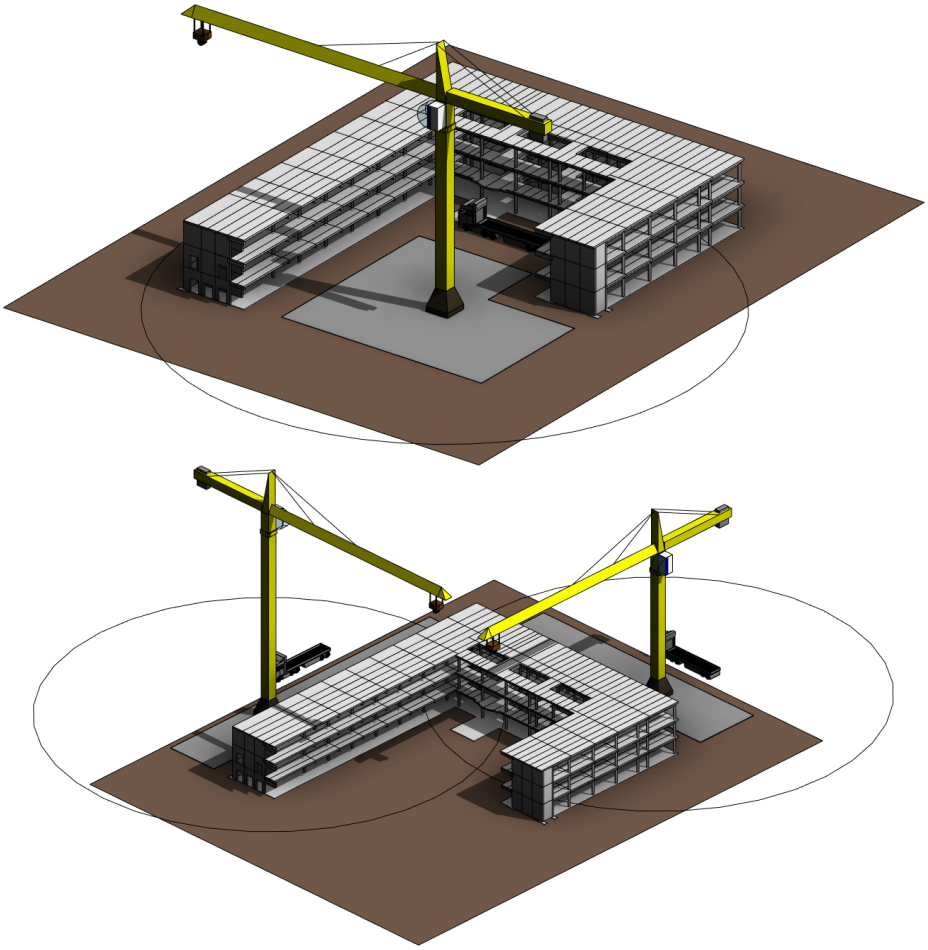


*FIG. 8 - CRANE SETUPS*

The next analyses are explained on both of the setups:

- **Situation Analysis**: evaluates the current setup in Revit and returns the Lift Status Values in Revit and the Lift Score in Dynamo.
- **Parametric Run**: evaluates a given list of possible positions of the crane(s) and returns the Lift Score for each situation in Dynamo. The results for the minimal Lift Score can then be used to reposition the cranes in Revit.
- **Genetic Optimization**: generates a random list of possible crane(s) positions and uses genetic algorithms to find the optimal position(s) of the crane(s), returning the smallest possible Lift Score.
- **Cloud based design exploration**: Explore the parametric design space of models created in Dynamo Studio with the automatic generation of a wide sampling of options.

## Input parameters for Crane Analysis

Each of the Crane Analysis is based on the same layout for the input of elements, external data and user input parameters. The *Input* phase consists of 4 parts, explained below.


[Part 1 – Element Collection]
Collects the elements from the Revit model according their category.

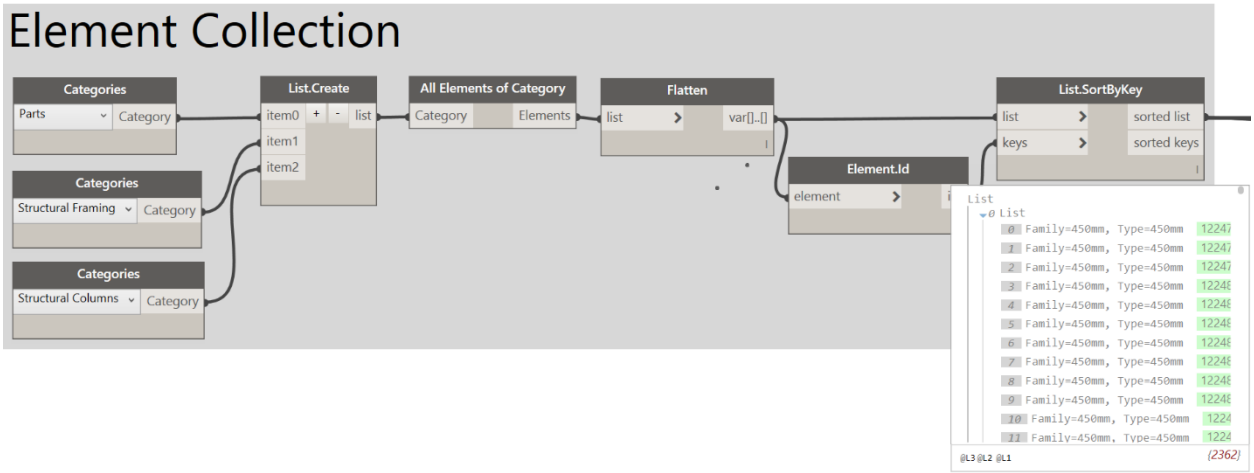
[Part 2 – User Input]
Parameters to be changed by the user, such as wall weight, lift status values, …

[Part 3 – Lift Capacity from Excel]
Reads the Excel table containing the crane specifications.

[Part 4 – Fixed Sample Creation]
Definition of UV divisions of a surface boundary. This is only used with the Parametric Run methods.

### *Part 1 - Element Collection*

Collects the elements from indicated categories and sorts them according the Revit ID. You can add other categories to the *List.Create* node.



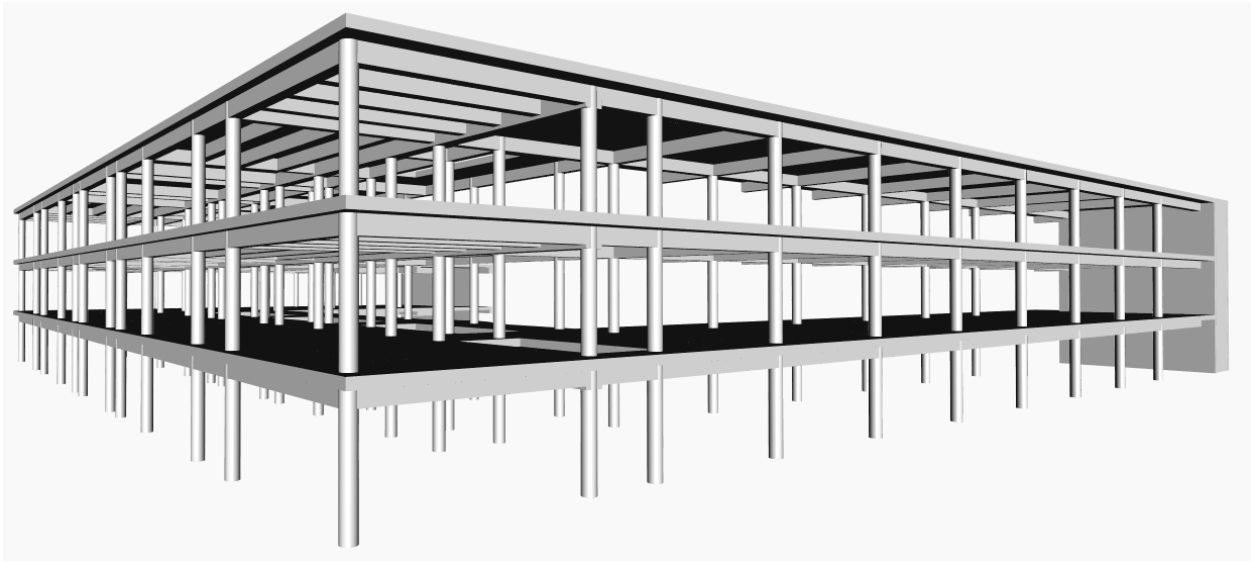
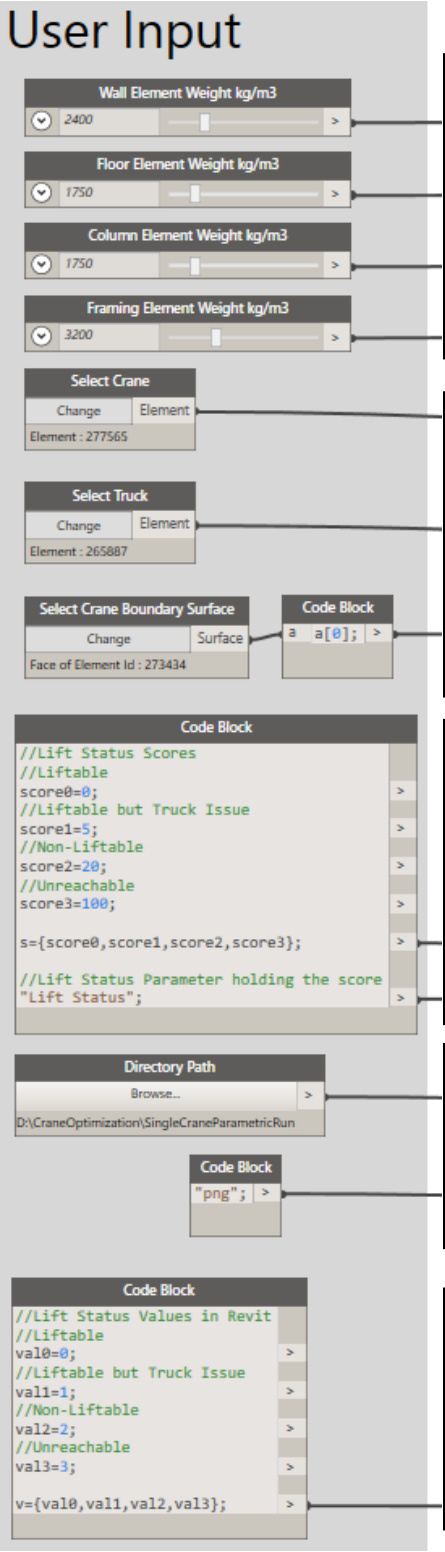



*FIG. 9 - ELEMENT SOLIDS IN DYNAMO*

*Part 2 – User Input*




Volumetric weight of the elements taking into account concrete, reinforcement steel and cavities.

Selection of the crane and truck elements in Revit.

In case of the Parametric Run or Genetic Optimization method, you also need to select the surface of the floor or building pad which limits the crane position.

This code block is used to define the base set of Lift Status Scores.

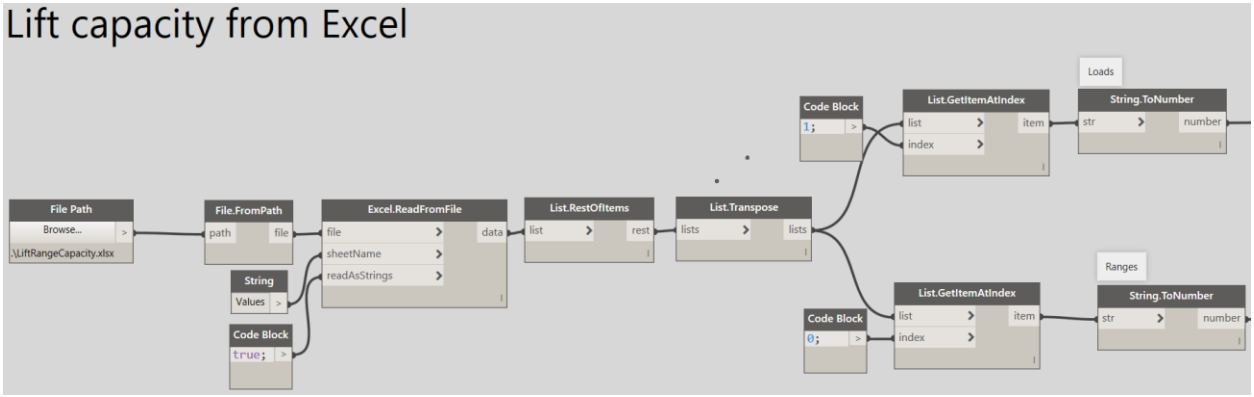
At the bottom the parameter used in Revit to store the Lift Status Values is given as a string.

This optional part is only used in the method *Parametric Run with Capture* and is needed to indicate a folder where the screen captures of the current Revit view need to be stored. The code block gives the file extension, which is “png” by default.

Use this code block to define the base set of Lift Status Values.

### *Part 3 – Lift Capacity from Excel*

This classical node group reads the crane specifications from an indicated Excel sheet, as shown in FIG.  10 and returns the Crane Range (column A) and Loads (column B) as numbers.

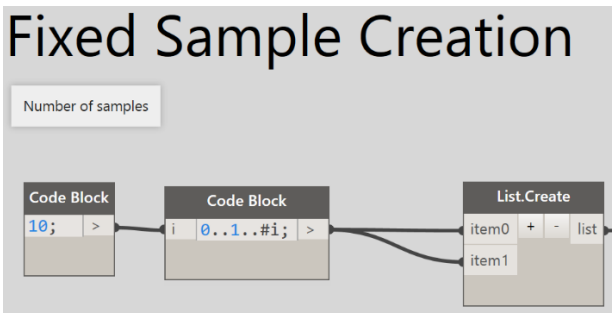



| | A | B |
|---|---|---|
| 1 | **Range** | **Load** |
| 2 | 23 | 16000 |
| 3 | 25 | 14800 |
| 4 | 27 | 13500 |
| 5 | 30 | 11900 |
| 6 | 32 | 11000 |
| 7 | 35 | 9900 |
| 8 | 37 | 9300 |
| 9 | 40 | 8400 |
| 10 | 43 | 7300 |
| 11 | 45 | 5900 |

*FIG.  10 - CRANE SPECS IN EXCEL*

### *Part 4 – Fixed Sample Creation*

This node group is only used in the Parametric Run methods. The result from this operation is a list of values between 0 and 1 that are further used to define points on the boundary surface, using a UV division method.



## Single Crane - Method 1 Situation Calculation

A *Situation Calculation* is a method where the current position of a tower crane is analyzed and evaluated. The result is a Lift Score and Lift Status Values assigned to the Revit elements, which can be used as base for View Filters then.

### DATASETS

**REVIT**

Start model:
*Single Crane Optimization – Case 1 – 00 Start.rvt*

Finalized model:
*Single Crane Optimization – Case 1 – 01 Situation Calculation.rvt*

**DYNAMO**

Workspace:
*SingleCraneAnalysis - 01 Situation Calculation.dyn*

Custom nodes:
*From the BIM4Struc.CraneAnalysis package*

External input:
LiftRangeCapacity.xlsx

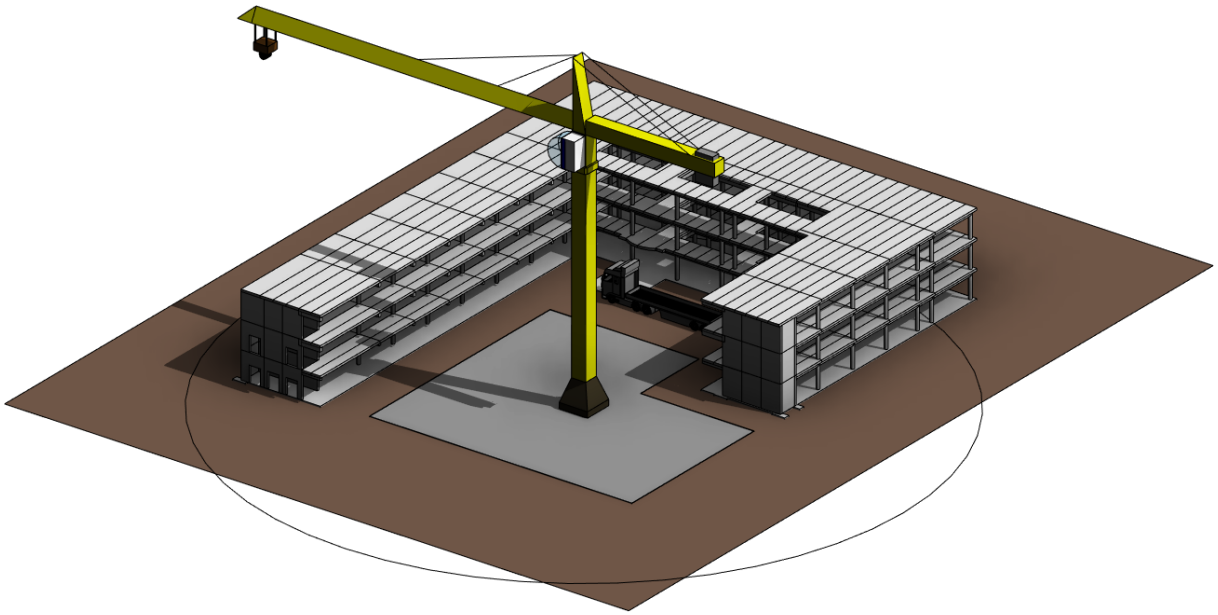
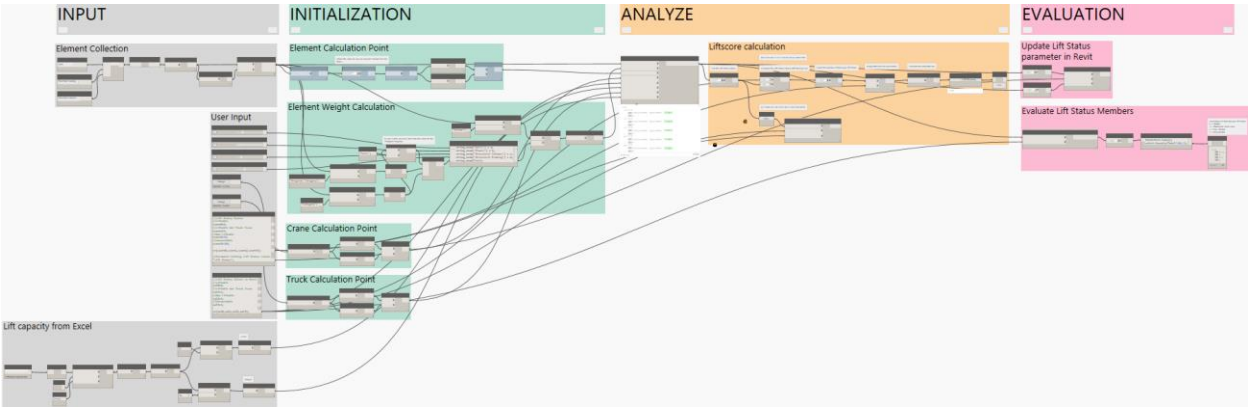


*FIG. 11 - SINGLE CRANE OPTIMIZATION - BEFORE ANALYSIS*

### *General overview*

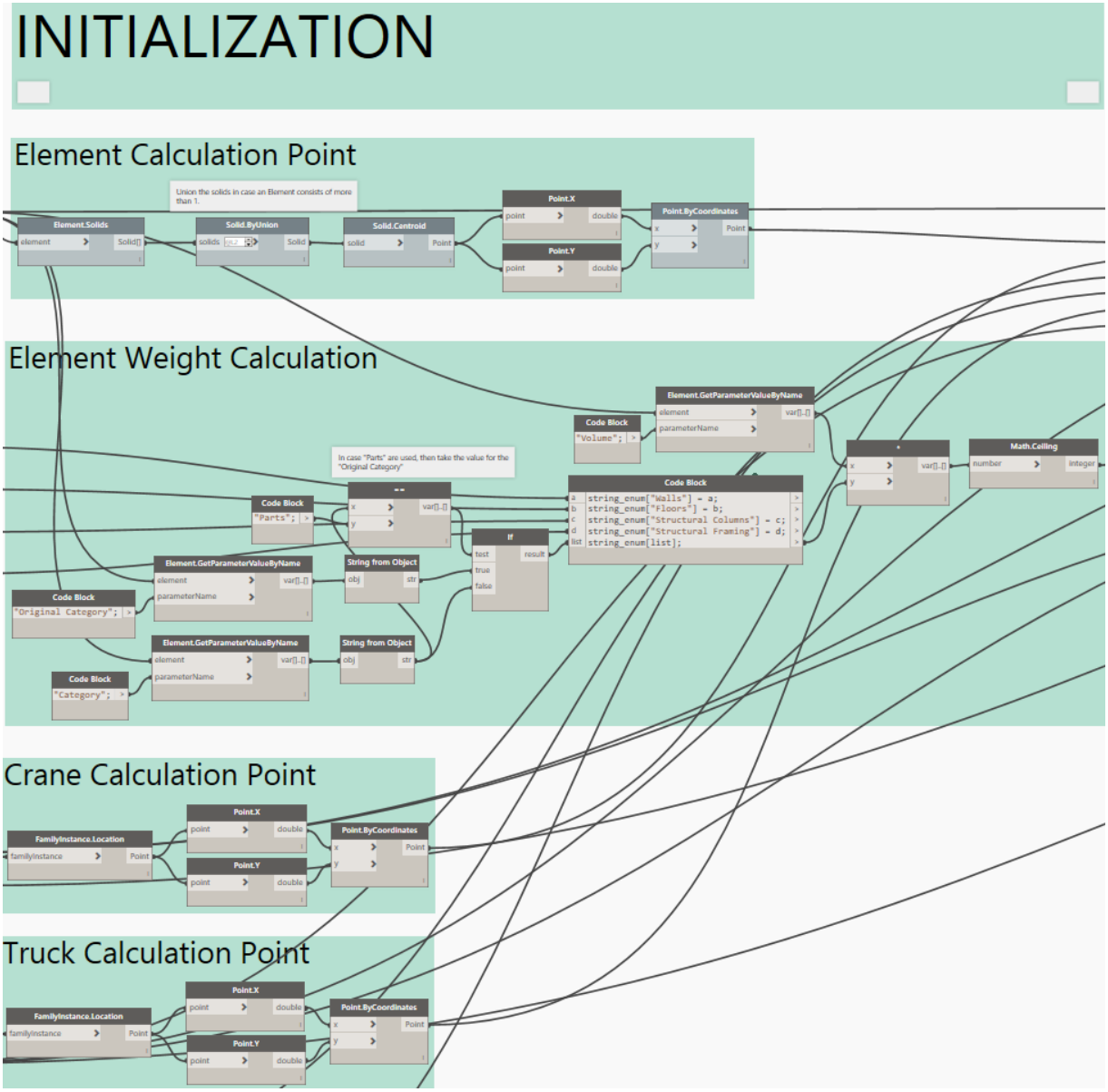



The method consist of 4 phases:

- **Input:** Collecting elements from Revit, reading external data and custom user input. Read this section.
- **Initialization:** Section used for calculating and consolidating the information from the user input.
- **Analyze:** Performs the actual crane analysis and returns the Lift Score
- **Evaluation:** Evaluates the Lift Score and returns feedback in Revit and Dynamo.

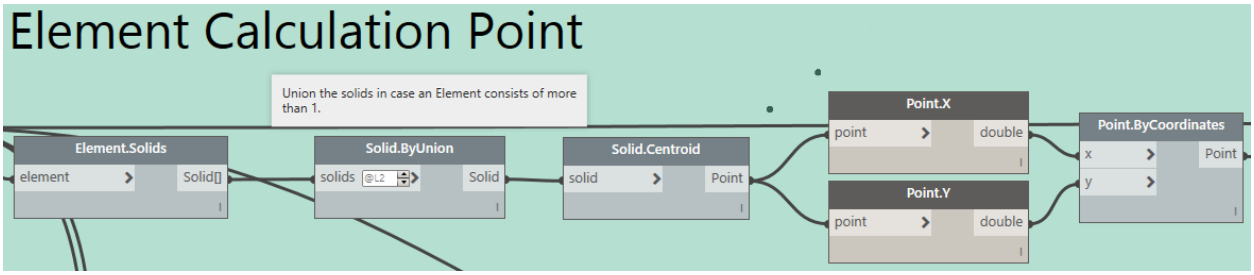
### *Initialization*

In this second phase, the information that has been collected in the “Input” phase, is evaluated, calculated and consolidated to be used further in the *Analyze* phase.



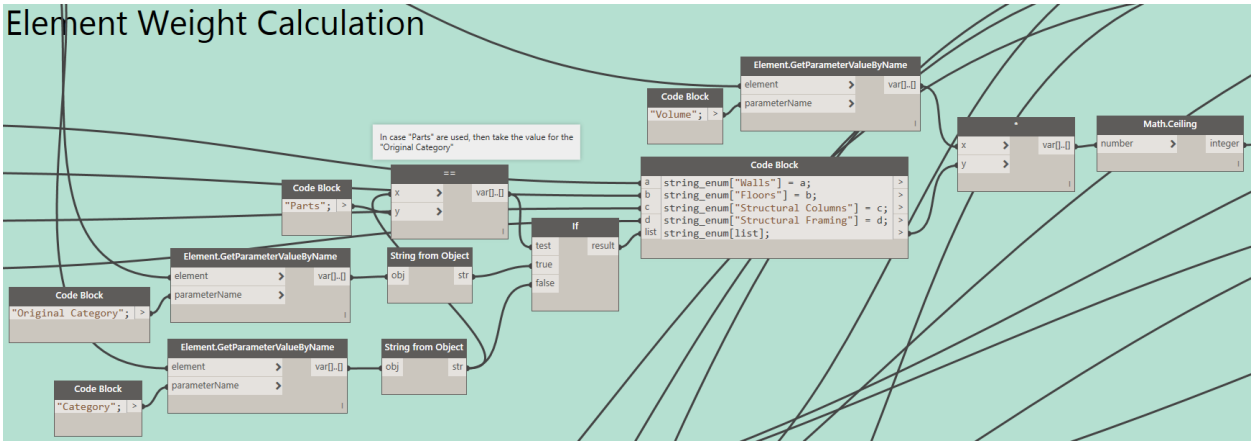
### ELEMENT CALCULATION POINT

This group of nodes reads the resulting elements from the Element Collection and defines the centroid of the solids shaping the element in Revit. In case the element consists of multiple solids, they are merged together with the *Solid.ByUnion* node. The resulting centroid point is then used to create a new point on Z=0 reusing the X- and Y-coordinate of the centroid. The results is then the Element Calculation Point (or the delivery position)



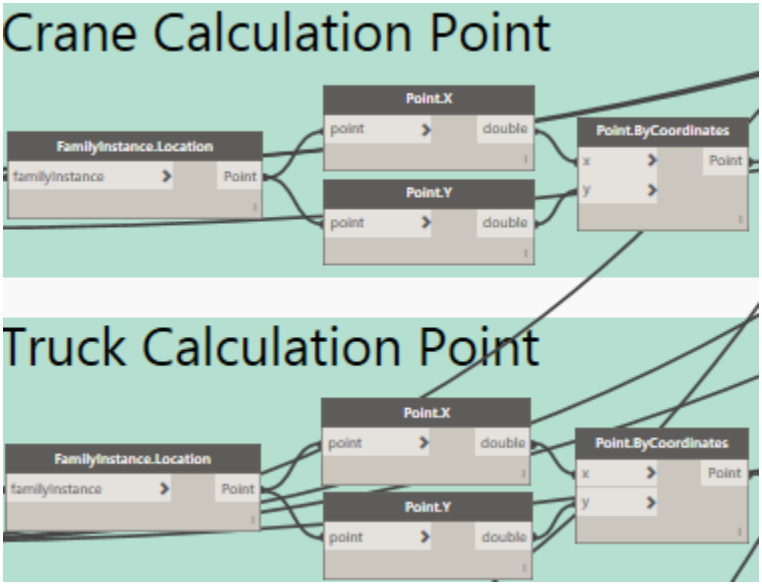

### ELEMENT WEIGHT CALCULATION

A second item that is needed for the analysis is the weight of each element. The weight is calculated based on the *Volume* (from the elements parameters) and *Volumetric Weight* (defined in the User Input) of the elements. In case an element is a Part (which means it can be a floor, wall, beam, column …) the original category is detected and that value is used to define the volumetric weight using the enumerators of the Code Block.



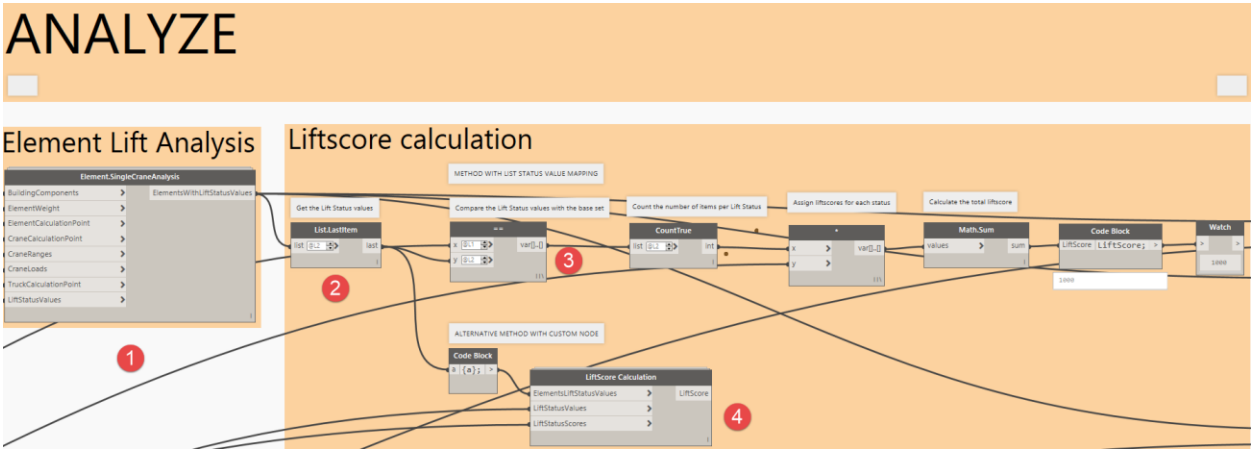
### CRANE & TRUCK CALCULATION POINT

The calculation points for the crane and truck, which are needed to get the distance between supply – crane – delivery points, are based on the insertion point of the family instance. Again, the points are recalculated to a new point with Z = 0.



### *Analyze*

The “Analyze” phase is the part where the input and initialized data are put together and analyzed. The result is a Lift Score.




## 1 – ELEMENT LIFT ANALYSIS

Evaluates the distance between the elements delivery point, the crane and the supply point (truck) and returns a grouped list of elements with their Lift Status value. This is done with a custom node *Element.SingleCraneAnalysis* delivered by the **BIM4Struc.CraneAnalysis** package. The contents of this node are explained in this chapter.

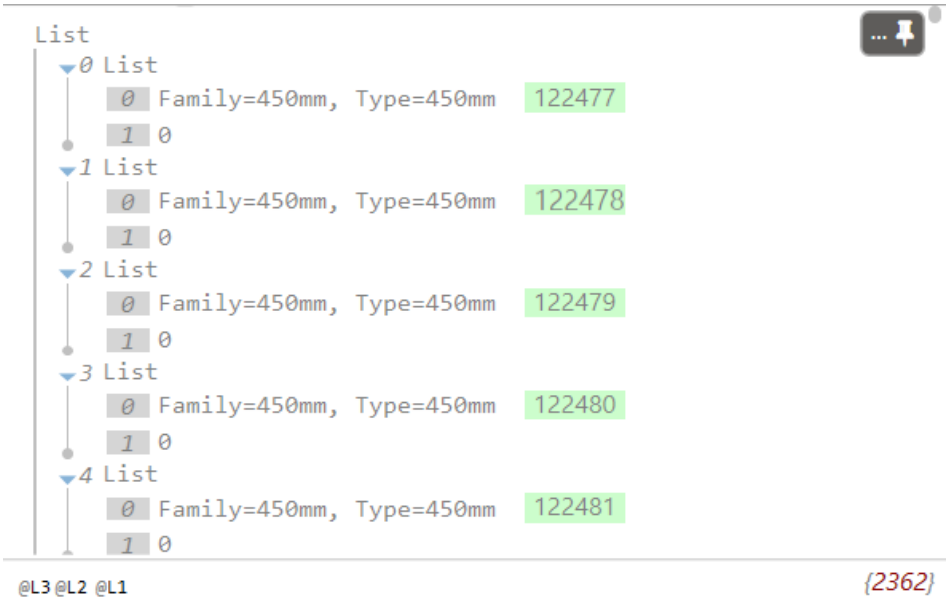
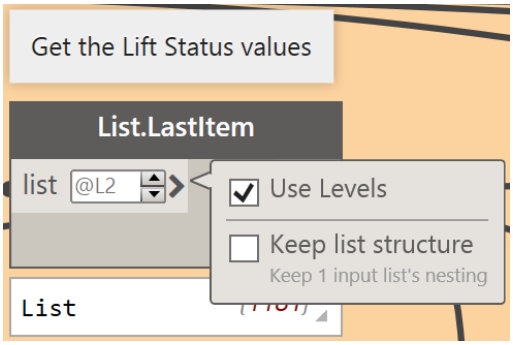



*FIG. 12 - RESULTS FROM ELEMENT.SINGLECRANEANALYSIS NODE*

### 2 – ISOLATE LIFT STATUS VALUES

To calculate the Lift Score we only need the Lift Status Values. These values are stored as the second (last) item of the sub lists, as shown in FIG. 12.



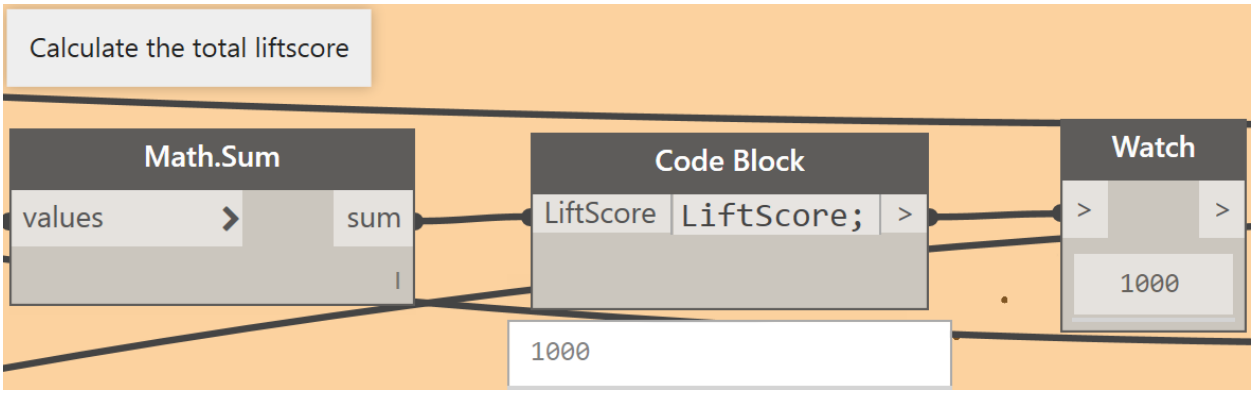

Activate the *Use Levels* option to map the *List.LastItem* node on the sub lists.

### 3 – CALCULATE LIFT SCORE

This part of the *Analyze* phase calculates the number of items for each *Lift Status*, by mapping the resulting *Lift Status Values* with the *Lift Status* base set from the Input phase. For each Lift Status the total amount of items is then multiplied with the corresponding defined *Lift Score Value* from the base set. The sum of these values result into the final *Lift Score.*

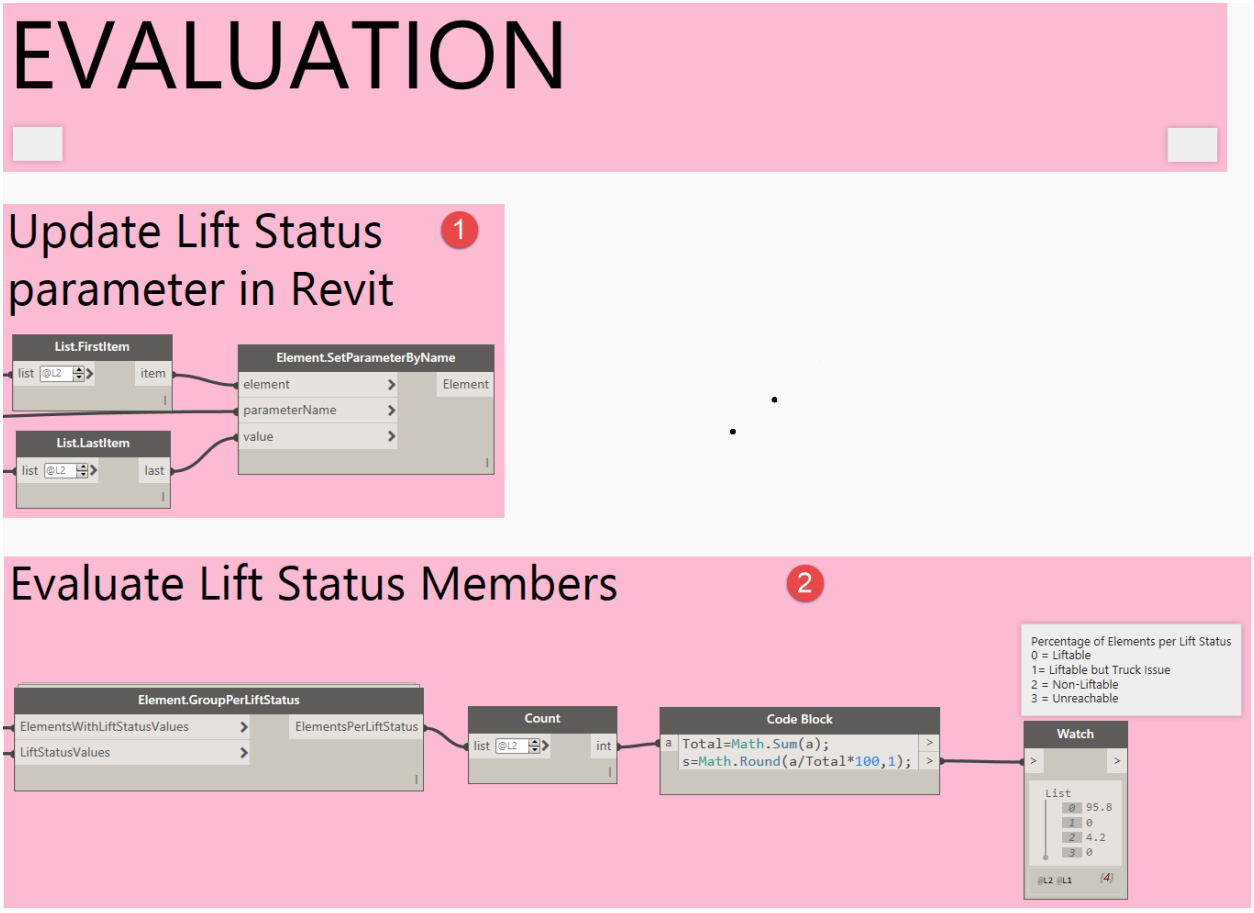
*In this example there are 1181 elements. A Lift Score of 0 would mean that each element is "Liftable". A Lift Score of 118100 would mean that all of them are "Unreachable".*

For the current crane position the Lift Score is 28460 which indicates a quite high score, hence a bad position of the crane.



### *Evaluation*

In this final phase, the *Lift Status values* and the total *Lift Score* are evaluated in Revit and Dynamo. This evaluation will help a user to make the right decisions on the crane position.

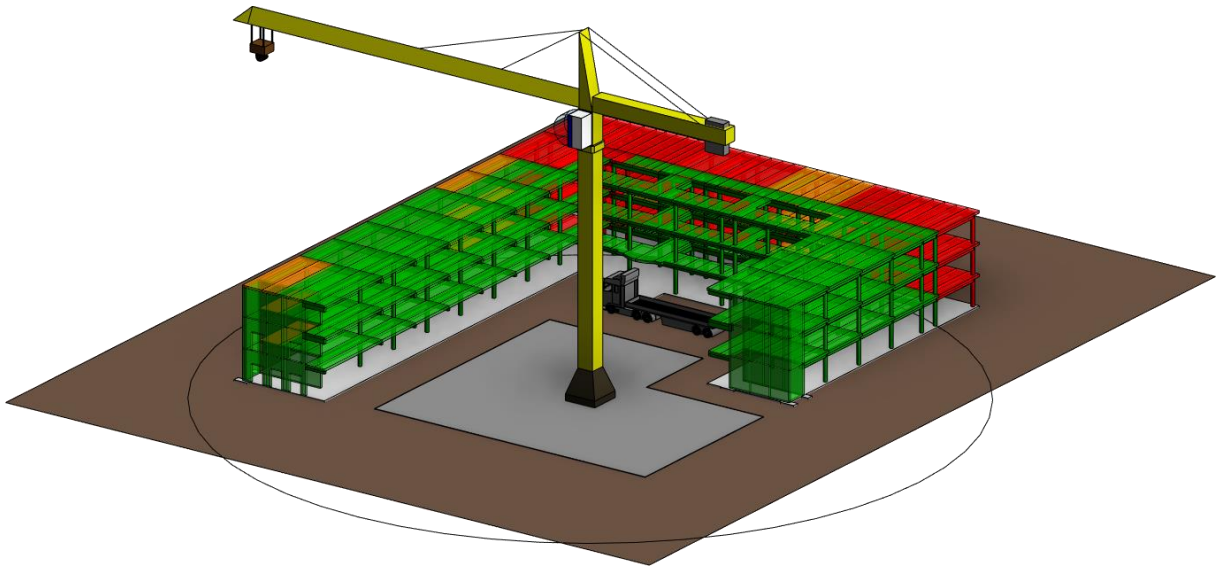


### 1 - UPDATE LIFT STATUS PARAMETER IN REVIT

In a first evaluation the results for the Lift Status Values are stored in the assigned instance parameter of each element in the Revit model. These results are based on the output of the *Element.SingleCraneAnalysis* node. The first sub-item of each main item in the list represents the Revit family instance, the second sub-item (last) of each main item represents the corresponding *Lift Status* value.

With the *Element.SetParameterByName* node, the family instance is provided with this *Lift Status* value in the assigned parameter (*Lift Status* in this case).

By doing this, it is possible now to evaluate the model by using View Filters, based on the parameter value. Read more in this section.

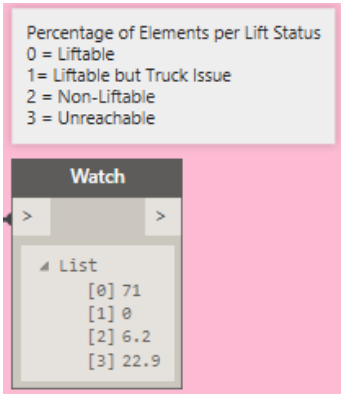


**Green** = Liftable
**Yellow** = Liftable but with truck issue
**Orange** = Non-Liftable
**Red** = Unreachable.

This corresponds with the percentages calculated below.

## 2 - EVALUATE LIFT STATUS MEMBERS

This optional evaluation splits the elements in groups again, according to their assigned lift status, and calculates their relative share to the whole construction (percentage).




In this example the construction consists of:

71% Liftable elements

0 % Liftable elements to far away from the truck

6.2% Non-Liftable elements

22.9% Unreachable elements

## Single Crane - Method 2a Parametric Run

In the previous method, only one single situation is considered. In that method you need a manual interaction with the Revit model for the relocation of the crane and recalculation of the Lift Score. When you need to find an optimal solution, then that method might take a lot of your time.

With a Parametric Run method, we will automate this process by feeding the analysis nodes with a (combined) list of variables defining the possible locations of a tower crane. To avoid infinite running, we need to constraint these variables. Practically this could be a surface on which the tower crane can be located (boundary surface). Or in case you want to simplify you could constraint the location of the crane along a model line.

This script will result into a list of Lift Scores, one for each situation of the crane. The combination with the minimal Lift Score will then be the best solution out of the fixed sample list.

### DATASETS

**REVIT**

Start model:
*Single Crane Optimization – Case 1 – 00 Start.rvt*

Finalized model:
*Single Crane Optimization – Case 1 – 02 Parametric Run result.rvt*

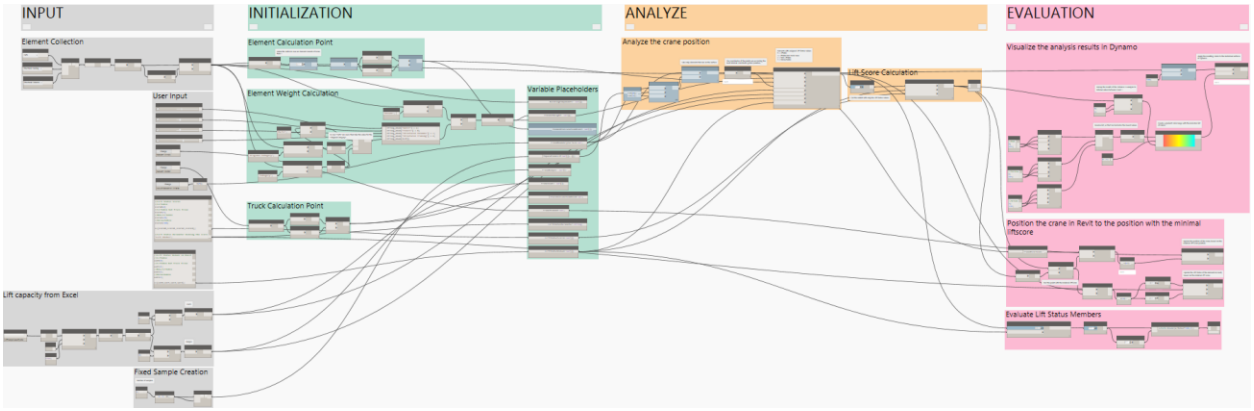
**DYNAMO**

Workspace:
*SingleCraneAnalysis – 02a Parametric Run.dyn*

Custom nodes:
*From the BIM4Struc.CraneAnalysis package*

External input:
LiftRangeCapacity.xlsx

### *General Overview*




The method consist of 4 phases:

- **Input**
  Collecting elements from Revit, reading external data and custom user input. Also a fixed set of samples is defined on which the analysis needs to run. This phase is already explained in this section.

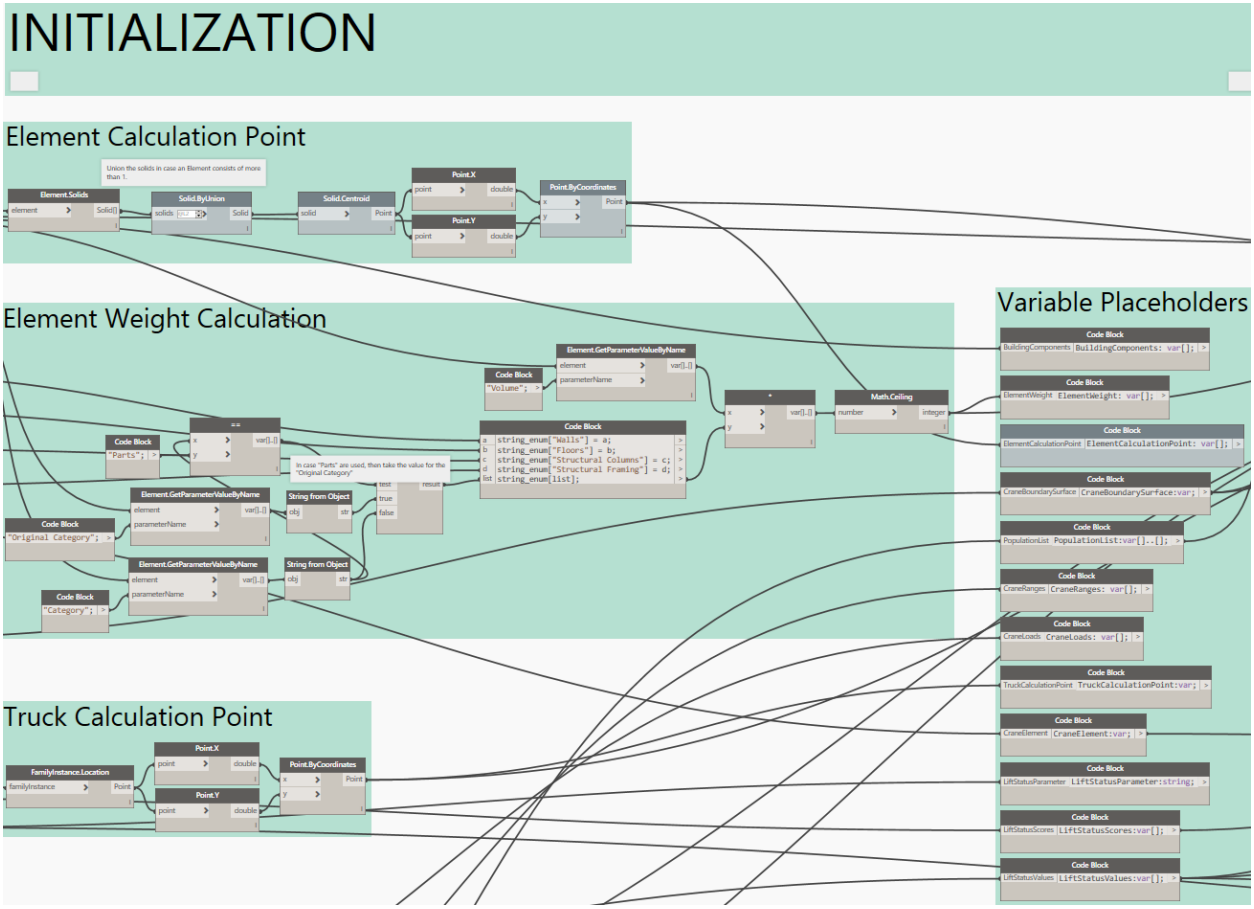
- **Initialization**
  Section used for calculating and consolidating the information from the user input.

- **Analyze**
  Performs the actual crane analysis on the sample list and returns the Lift Scores for each situation.

- **Evaluation**
  Evaluates the Lift Score by finding the situation with the minimal score and returns feedback in Revit and Dynamo.

*Initialization*

In this second phase, the information that has been collected in the “Input” phase, is evaluated, calculated and consolidated to be used further in the *Analyze* phase.




In this second phase, the information that has been collected in the “Input” phase, is evaluated, calculated and consolidated to be used further in the *Analyze* phase.

### ELEMENT CALCULATION POINT

This group of nodes reads the resulting elements from the Element Collection and defines the centroid of the solids shaping the element in Revit. Read more in this section.

### ELEMENT WEIGHT CALCULATION

A second item that is needed for the analysis is the weight of each element. Read more in this section.

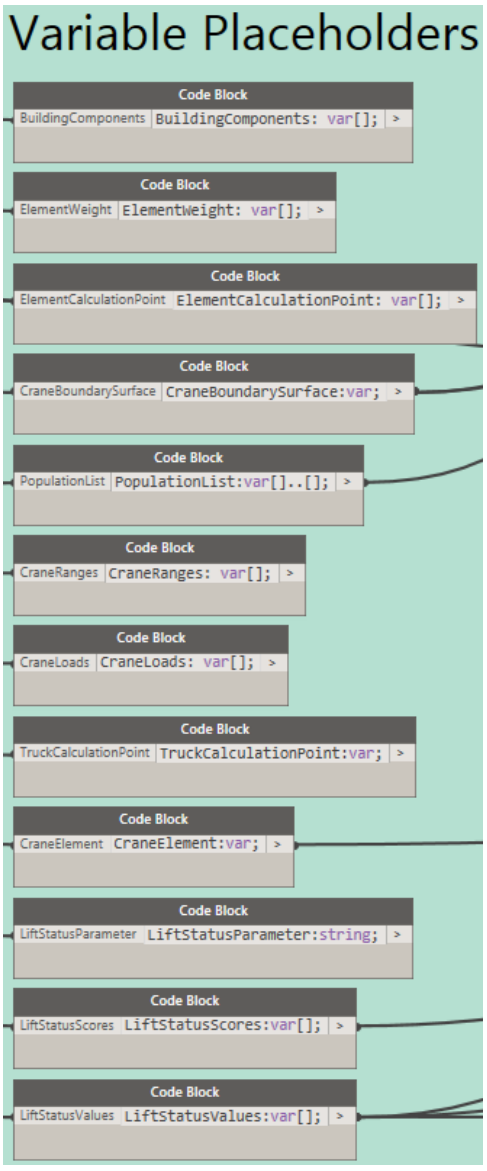
### TRUCK CALCULATION POINT

The calculation points for the truck (supply point) is based on the insertion point of the family instance. Again, the points are recalculated to a new point with Z = 0.

The crane calculation point is defined in the *Analyze* phase, using the UV parameters from the fixed sample list in the *Input* phase.

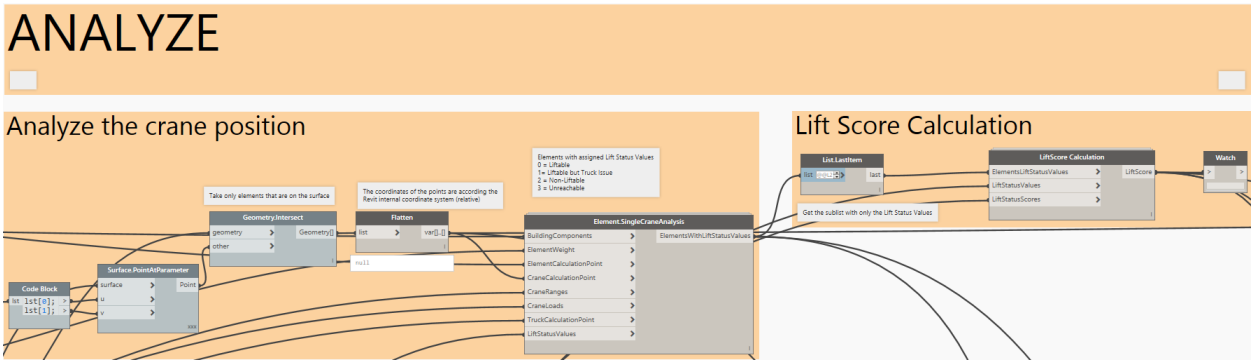
### VARIABLE PLACEHOLDERS

This part reorganizes all the variables into one readable group of parameters. This makes it easier to connect with the nodes from the *Analyze* and *Evaluation* phase.

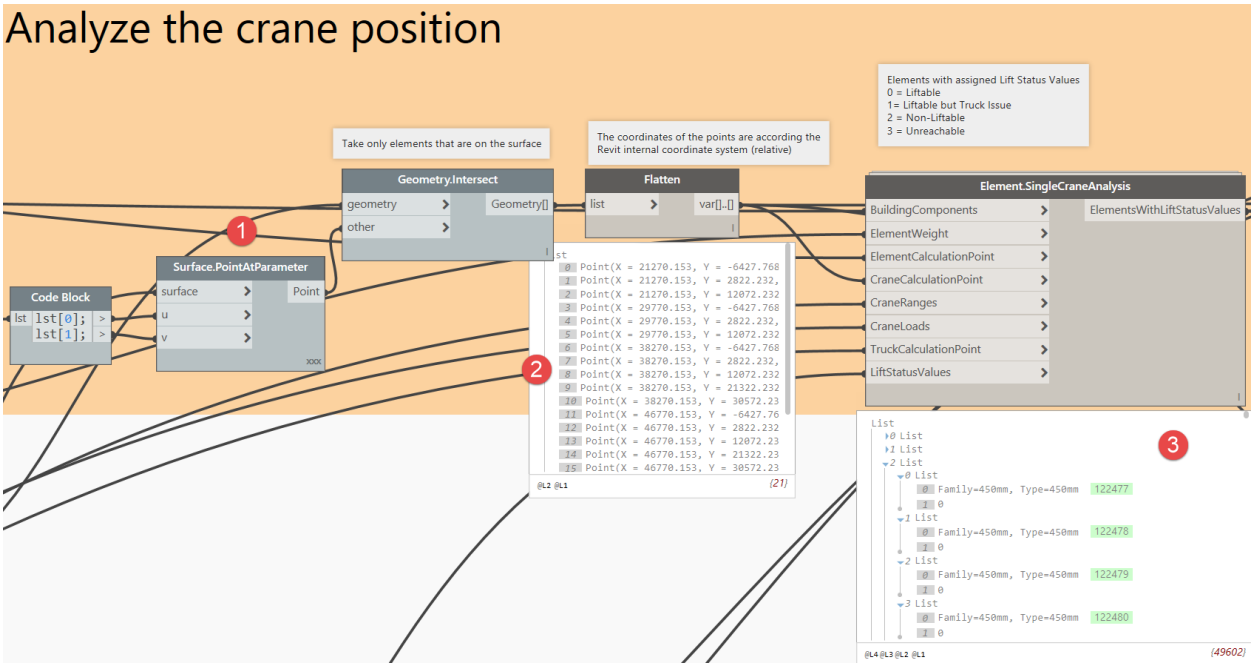


### *Analyze*

The “Analyze” phase is the part where the input and initialized data are put together and analyzed. The result is a list of Lift Scores, one for each situation.




## ANALYZE THE CRANE POSITION




**(5)** This part creates possible points for the crane position with the *Surface.PointAtParameter* node The UV parameters are defined by the Fixed Sample list (between 0 and 1) and the surface is defined by the face of a *Building Pad* element in the Revit model. This surface defines the limits of the crane position.

**(6)** The *Surface.PointAtParamater* node creates points according the global coordinate system and only considers the bounding box of the surface contour. In case the contour of the “boundary surface” is not rectangular, then some points are created outside this contour (but still within the bounding box). With the *Geometry.Intersect* node you can collect the points that are only on the surface.

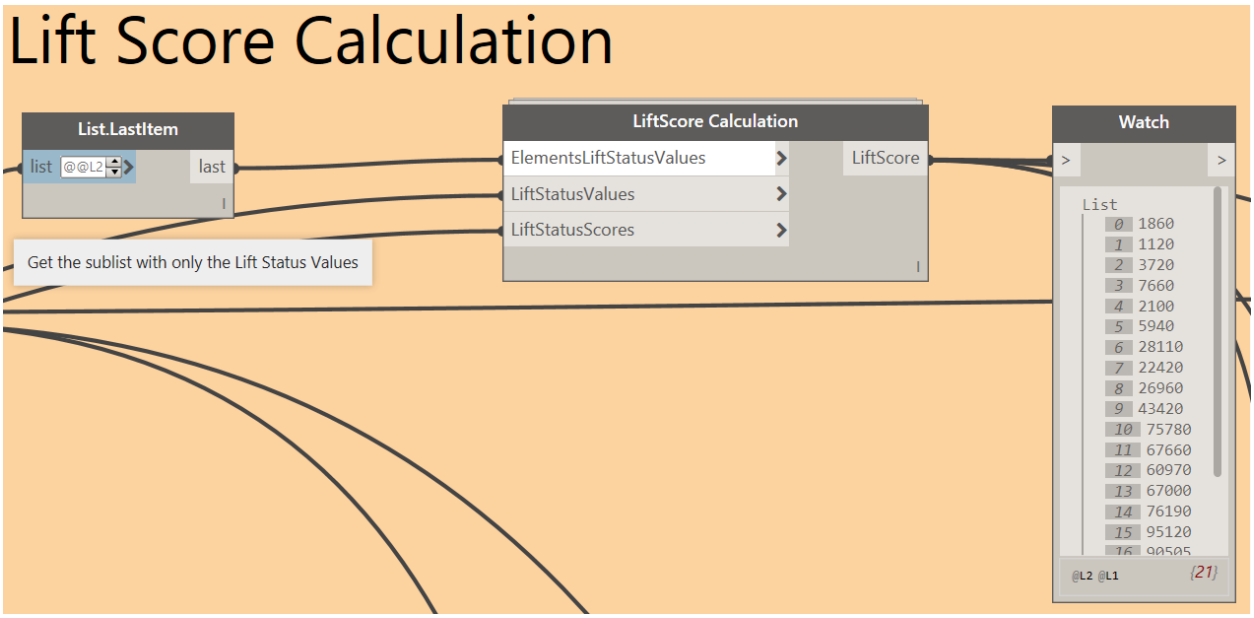
Finally you get a list of possible positions of the crane in Revit. The coordinates of the points are according the Revit internal coordinate system (relative).

***Note: Lacing Settings***
*The lacing for the Surface.PointAtParameter needs to be set to 'Cross Product'. This is needed if you want each U-division point to be combined with each V-division parameter to create the points.*

**(7)** The *Element.SingleCraneAnalysis* runs on each of the points separately. The result is an array of Family Instances with their Lift Status value, for each possible location of the crane.

***Note:***
*You only get this behavior in Dynamo 1.2. In earlier versions you need to connect the points with the analysis node by using a List.Map* node.

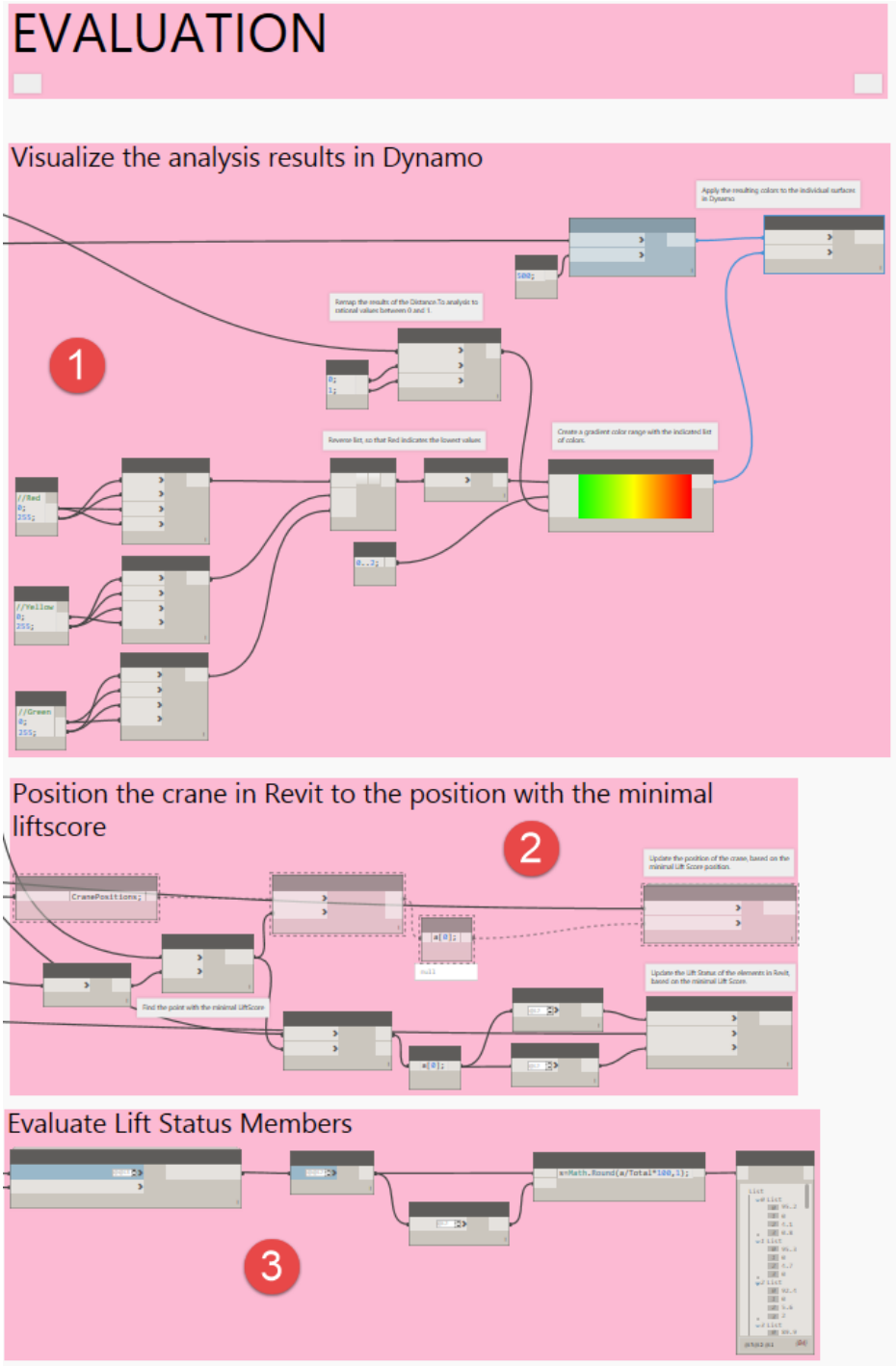
### LIFT SCORE CALCULATION




In this part the *Lift Status Values* are taken separately for each crane position. The values are recalculated to a total *Lift Score* for each situation with the *LiftScore Calculation* node.

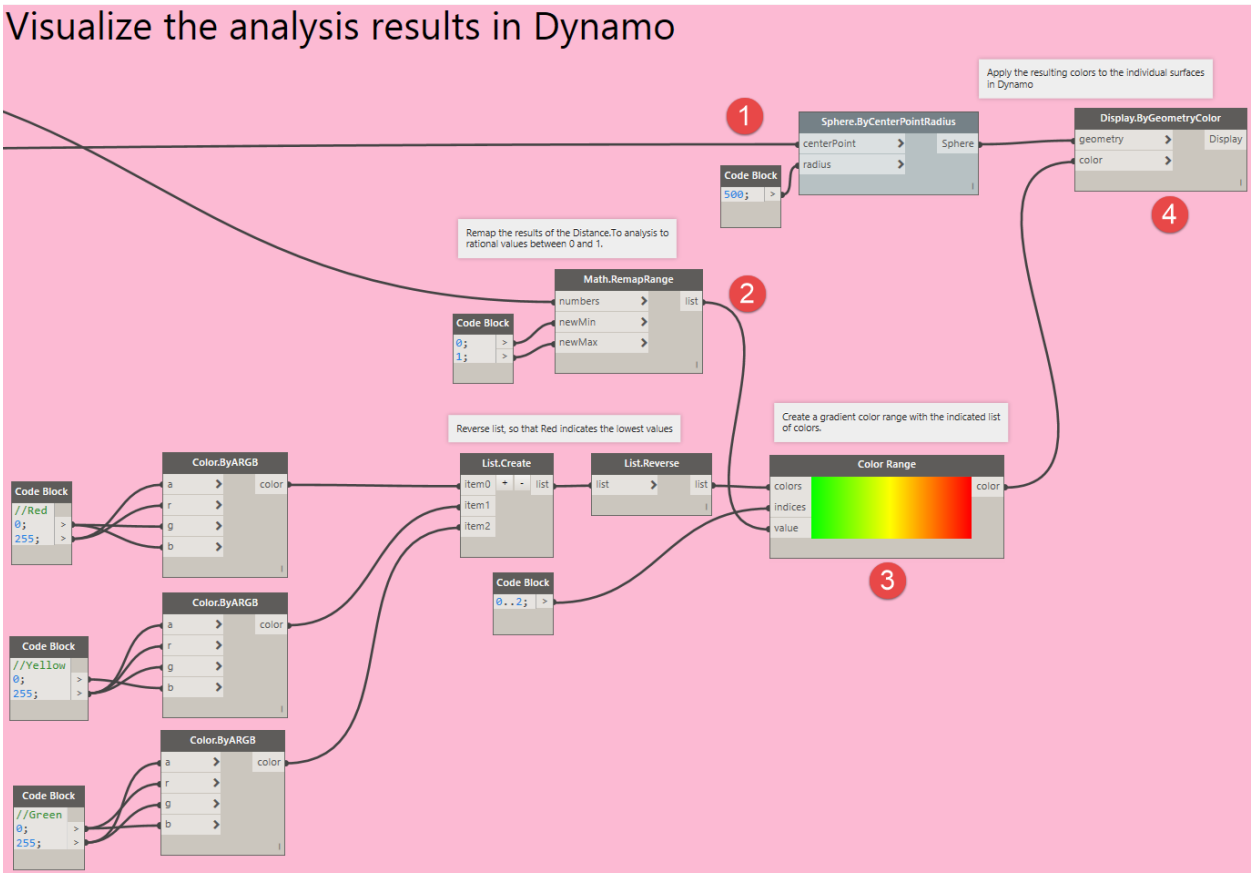
### *Evaluation*

In this final phase, the *Lift Scores* are evaluated in Dynamo and the Crane position point corresponding with the minimal *Lift Score* is detected.



### 1 - VISUALIZE THE ANALYSIS RESULTS IN DYNAMO

This group of nodes makes it possible to visualize the results of the *Lift Scores* as colored balls in the Dynamo GUI.




**(1)** Use the points from the possible crane positions to create a *Sphere.ByCenterPointRadius*

**(2)** Remap the results of the *Lift Scores* from 0 to 1.

**(3)** Use the new values between 0 and 1 to create a color range.

**(4)** Apply the color range to the spheres. Each sphere has a corresponding crane position point, which has a corresponding lift score, which has a corresponding color (after remapping).

The results are shown in FIG. 13.

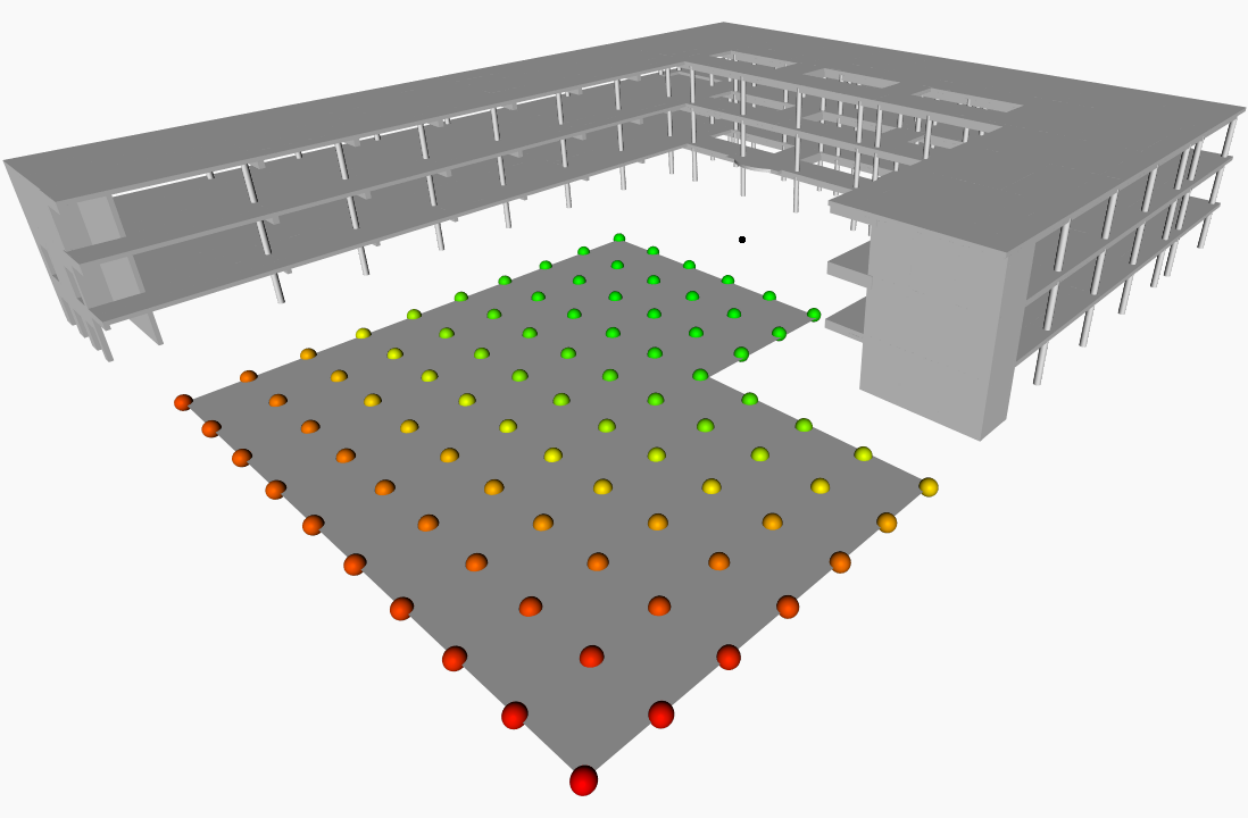
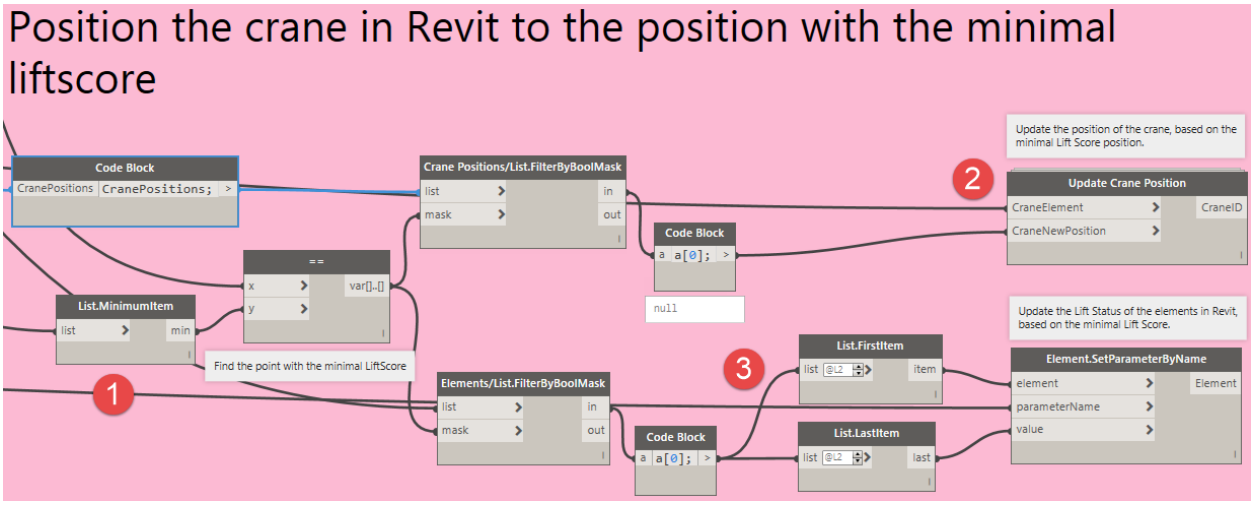


*FIG. 13 - SINGLE CRANE PARAMETRIC RUN ANALYSIS RESULTS IN DYNAMO*

### 2 - VISUALIZE THE BEST RESULT IN REVIT

The results in Revit are shown for the best situation only.




**(1)** First find the minimal lift score (*List.MinimumItem)* and use it to create a Boolean mask with the == node. This mask will be a list with one *True* and the rest *False*.

**(2)** The index of the *True* item, is used to get the Crane Position point corresponding with the Minimal Lift Score. That point is used as input for moving the selected crane to its new location, using the custom node *Update Crane Position*.

**(3)** The same Boolean mask is used to filter the list of elements with their *Lift Status* values. The result is set to the appropriate Revit parameter of the instances.

The results are shown in FIG.  14.

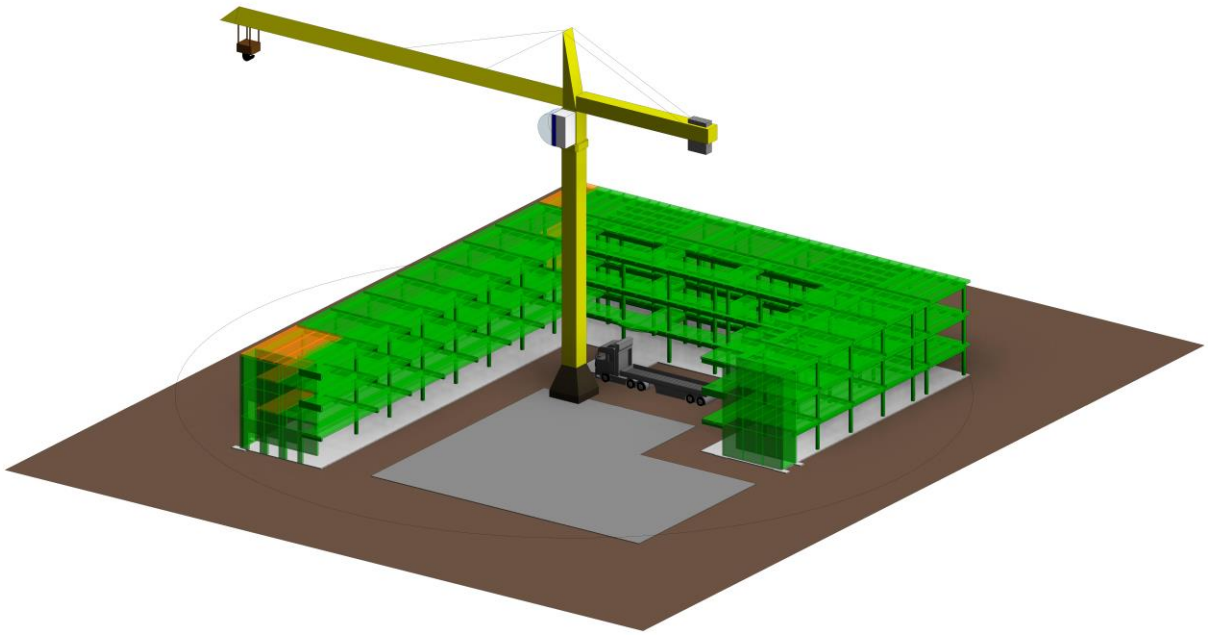
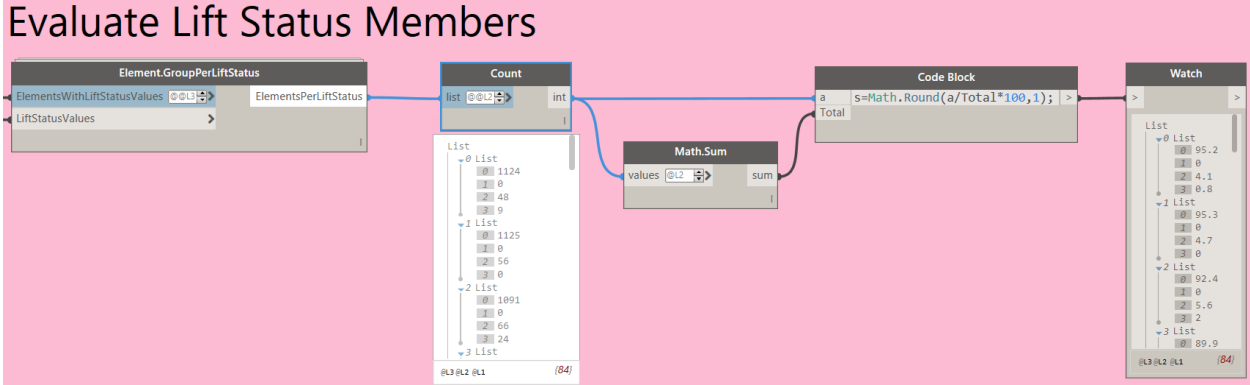
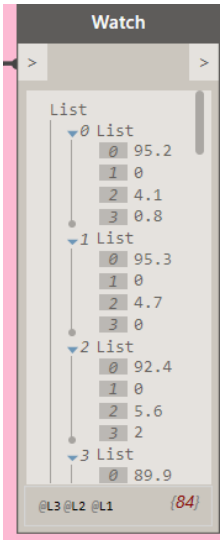


*FIG. 14 - RESULTS IN REVIT FOR A SINGLE CRANE PARAMETRIC RUN*

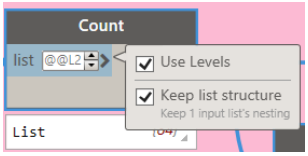
### 3 - EVALUATE LIFT STATUS MEMBERS




This optional evaluation splits the elements in groups again with the *Element.GroupPerLiftStatus* node (**BIM4Struc.CraneAnalysis** package), according to their assigned lift status, and calculates their relative share to the whole construction (percentage). This is done for each situation of the crane calculation point.

The list level of the *Count* node need to be set as shown below, so that every sub list is counted and the structure of the input is maintained.




The output is then an array of the number of elements, per lift status (4 in this case) per situation of the crane position (which is the main list).

## Single Crane - Method 2b Parametric Run with Capture

This method is an alternative Parametric Run method, where the results of each iteration are exported to images capturing the current Revit view. This method makes it possible to review the intermediate results of the calculations in a visual way.

### DATASETS

**REVIT**

Start model:
*Single Crane Optimization – Case 1 – 00 Start.rvt*

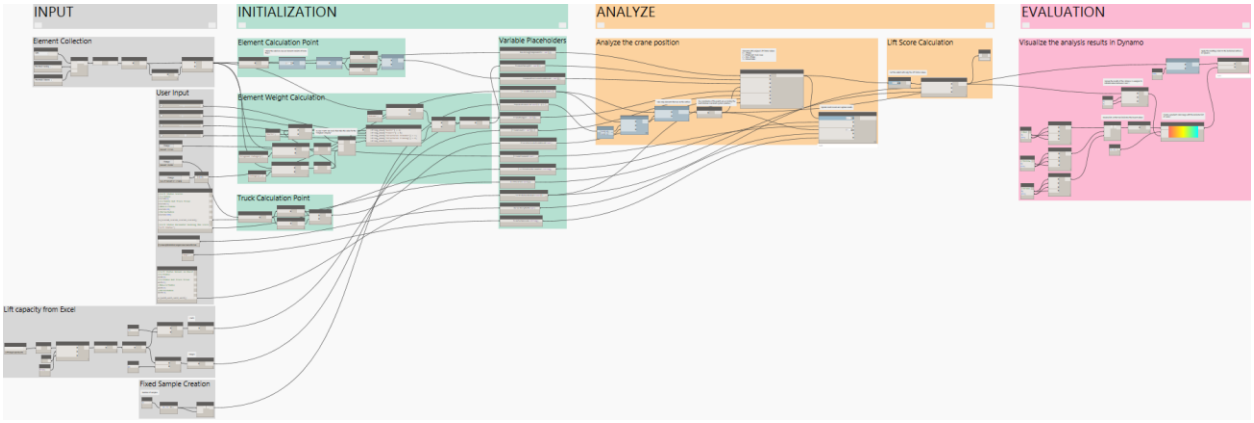
Finalized model:
*Single Crane Optimization – Case 1 – 02 Parametric Run result.rvt*

**DYNAMO**

Workspace:
*SingleCraneAnalysis – 02b Parametric Run with Capture.dyn*

Custom nodes:
*From the BIM4Struc.CraneAnalysis package*

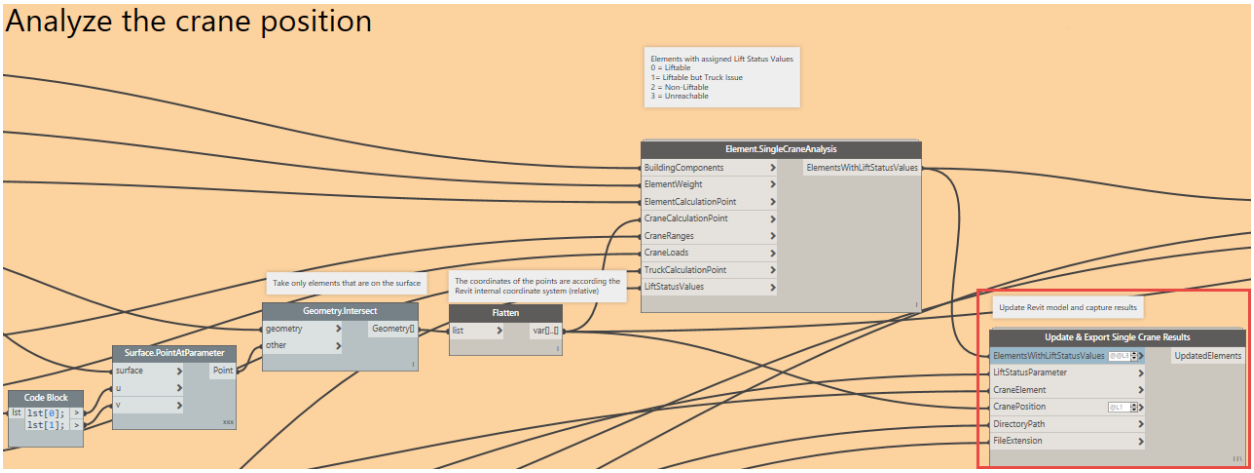
External input:
LiftRangeCapacity.xlsx



### *Update & Export Analysis Results*

The layout of this script is basically identical to the one described in Method 2a.

The only exception is the part where the crane position gets analyzed.



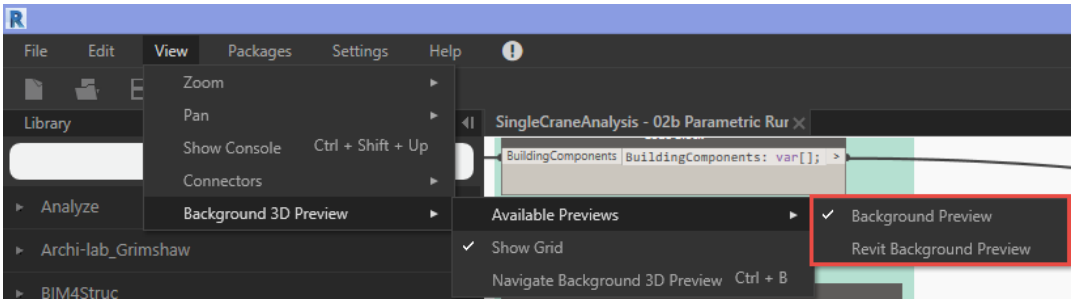

First, the crane position is analyzed for the suggested sample set of points with the *Element.SingleCraneAnalysis* node.

The results (the Revit family instances and the *Lift Status* values) are then combined with the crane position points into the custom node *Update & Export Single Crane Results*.

This custom node performs these actions, as explained in this chapter.

***Note:***

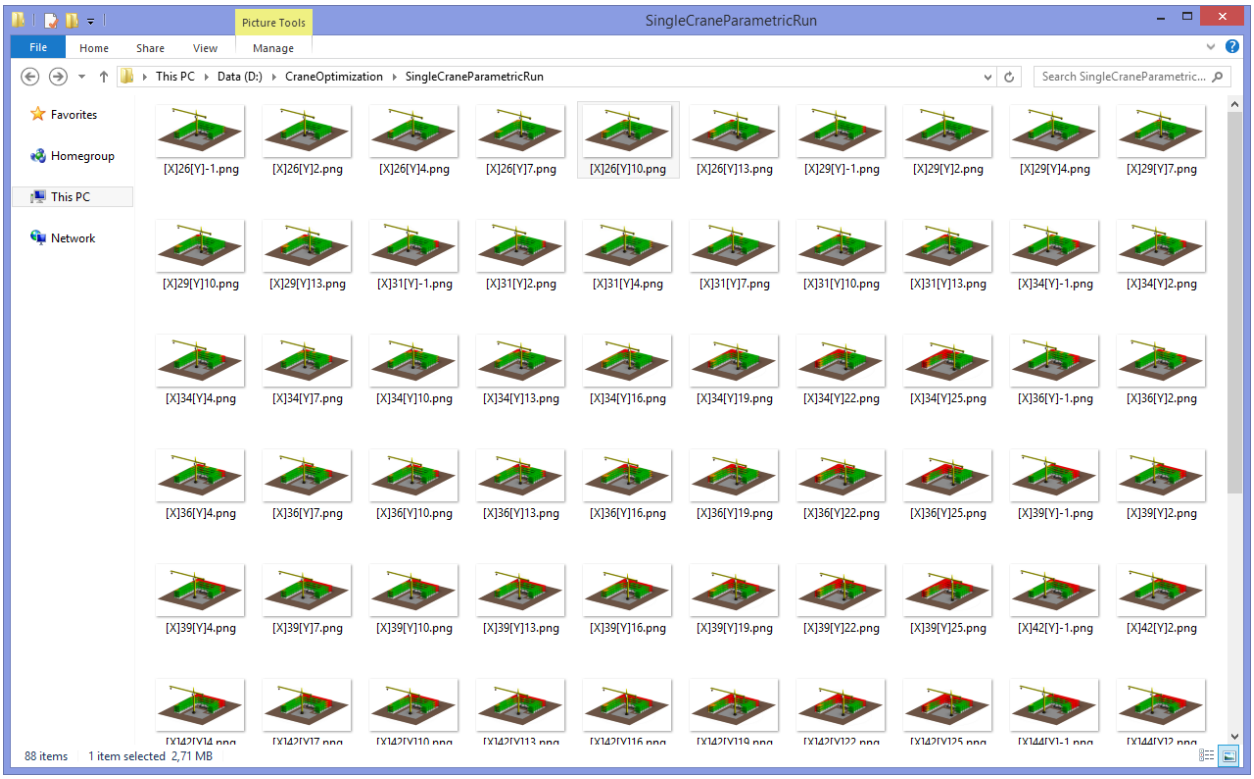
*Before you run the script it is advised to switch off the Revit Background view in Dynamo. This will avoid lighten the colors in the Revit screen captures.*



### *Resulting Files*

The result is a number of image files. Putting these images together in an animated video, you can see the evolution of the parametric run.

Tools like Windows Movie Maker are very handy in use for this exercise.



## Single Crane - Method 3 Genetic Optimization

The “parametric run” method can be time consuming, as it considers every design “combination” option. Besides that, the parametric run will only calculate the given set of combinations, and won’t find alternatives that might be better.

To speed this up, and to find more accurate results we need to introduce Genetic Algorithms in the analysis process. The GA Optimization will use specific methods making a natural selection of the initial solution list and by using cross-over and mutation algorithms in order to find the optimal solution.

A bit straight forward but you can see this as “introducing the natural evolution theory of Darwin” into computational design.

The main parts of the script that has been made in the Parametric Run method, can be reused by adding specific nodes from the ***Optimo for Dynamo*** package. Read more about the use of this package via this link.

### DATASETS

**REVIT**

Start model:
*Single Crane Optimization – Case 1 – 00 Start.rvt*

Finalized model:
*Single Crane Optimization - Case 1 - 03 Optimization result.rvt*

**DYNAMO**

Workspace:
*SingleCraneAnalysis – 03 Genetic Optimization.dyn*

Custom nodes:
*BIM4Struc.CraneAnalysis package*
*Optimo package*

External input:
LiftRangeCapacity.xlsx

### *Representation of the Optimization Problem*

Before you start building up the script, it is important to first represent the optimization problem which consists of an “objective”, “design variables” or “population” and the “design variables domain”.

#### OBJECTIVE

With the optimization we want to find a crane configuration that has a minimal *Lift Score.* The *Lift Score* is calculated depending on the lift status of each calculated element in the construction. Read more in this section.

#### DESIGN VARIABLES

The design variables for this optimization are expressed with two parameters:

- U-division of the boundary surface
- V-division of the boundary surface

These divisions are used to define a point on the surface, which could be a potential crane position, until an optimal solution is found.

#### DESIGN VARIABLES DOMAIN

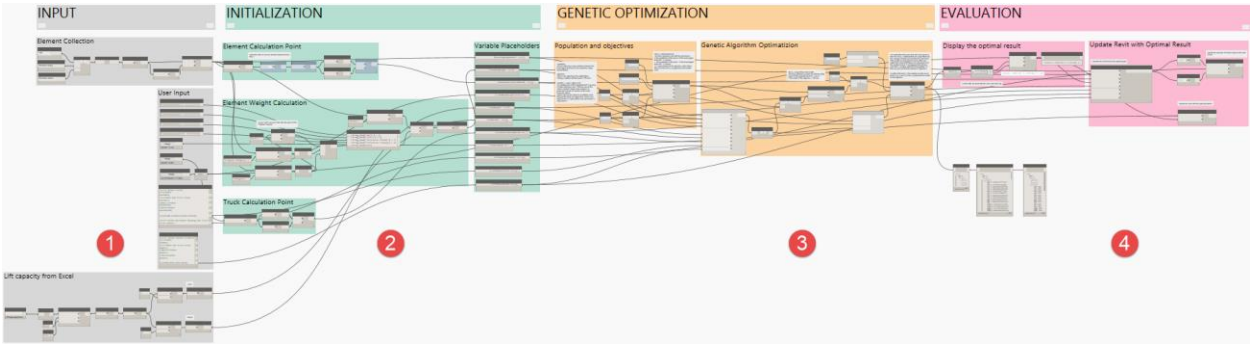
We need to give some restrictions for the design variables and give the optimization routine a boundary within to search an optimal solution for. In this case the domains are set as follows:

- $0 \leq U \leq 1$
- $0 \leq V \leq 1$

The UV-divisions need to be numbers between 0 and 1 to work with the Surface.PointAtParameter node in the analysis part. But this could be constrained between 0.2 and 0.8 as well if needed.

### *General Overview*

In this script constant values and an initial solution list generated by Optimo will be connected to a so called “fitness function” which is processed by the Optimo NSGA algorithm nodes.




The total script consists of 4 phases:

- **Input**
  Collecting elements from Revit, reading external data and custom user input. Also a fixed set of samples is defined on which the analysis needs to run. This phase is already explained in this section.

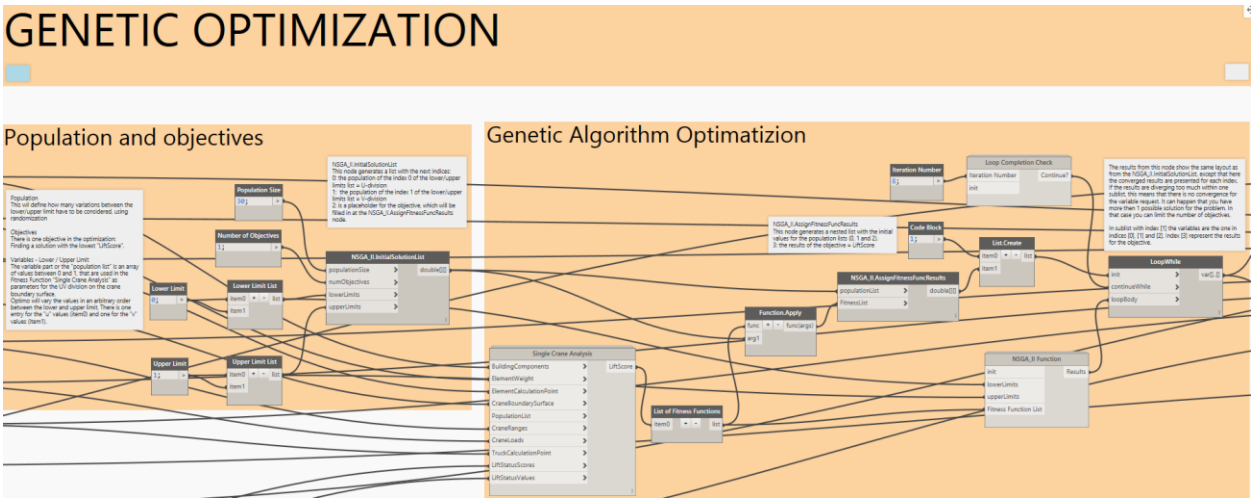
- **Initialization**
  Section used for calculating and consolidating the information from the user input. This phase is already explained in this section.

- **Genetic Optimization**
  The initial population list is generated and the design variables are optimized using Genetic Algorithms until the objective of a minimal Lift Score is reached.

- **Evaluation**
  Evaluates the Lift Score and returns feedback in Revit and Dynamo.

### *Genetic Optimization*

This phase generates the initial solution list, based on design variables within a defined domain, and calculates the *Lift Score* with a fitness function. In the next iterations genetic algorithms (GA) are used to define a better population until an optimal solution is found or the number of iterations is reached.



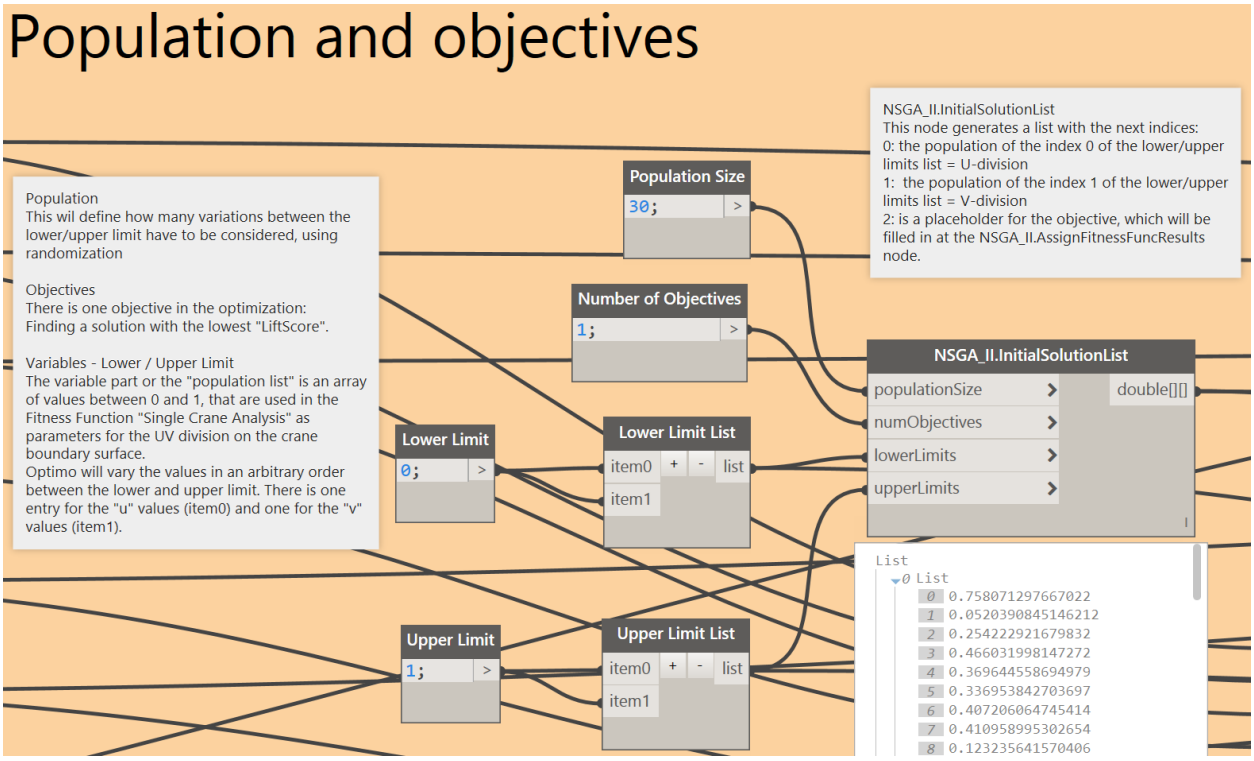

Two important parts can be considered here:

- **Population and objectives**: Generation of initial solution list
- **Genetic Algorithm Optimization**: Optimization of the solutions, using a custom fitness function.

### POPULATION AND OBJECTIVES

The goal of this step is to generate the initial set of random variables within the provided range and with the size of population defined by user. Therefore the Design Variables need to be defined and set up within a Design Variables Domain.

The *NSGA_II.InitialSolutionList* node creates a list of randomized numbers (initial population) within the range specified and with the number of defined variables.




This node has 4 input ports:

- **Population Size**: this is the amount of numbers that need to be generated for each variable. In this script we use a population of 30, which means that 30 random values are being generated. This is minimal needed for the size of the surface that is holding the crane.

- **numObjectives*:*** this is the amount of fitness functions that need to be optimized. This needs to be equal to the number of list input ports of the *List of Fitness Functions* node (which is a List.Create node).
  There is only 1 objective here, which is finding the *Lift Score.*

- **lowerLimits**: this represents a list of the minimal values of the variables that need randomization.

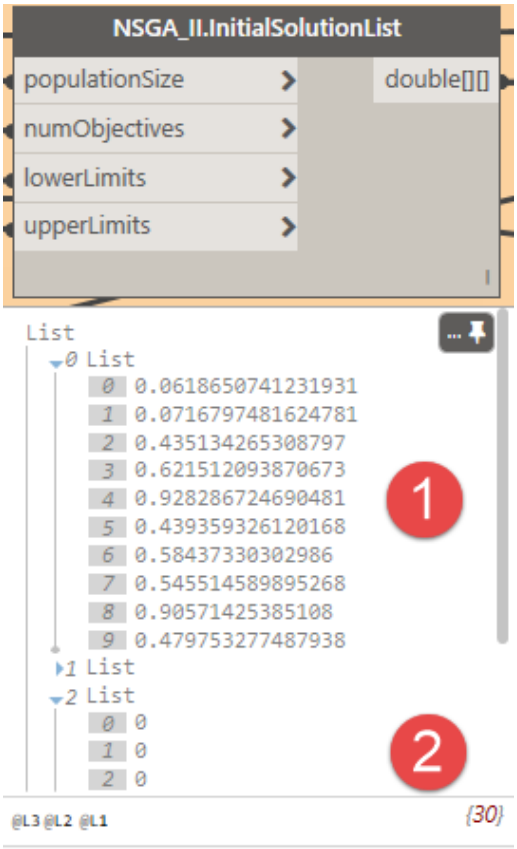
- **upperLimits** : this represents a list of the minimal values of the variables that need randomization

The size of the list for lowerLimits and upperLimits needs to be the same. So we have 2 lower limits (1 for the U division and 1 for the V division), and similar for the 2 upper limits.

The output of the *NSGA_II.InitialSolutionList* node consists of a list containing:

**(1)** An amount of sub lists, one for each variable with an amount (Population Size) of randomized numbers (between lowerLimit and upperLimit). This is called the "parents solution list".
*In the example below: 3 sub lists with 10 random numbers each (Population Size = 10)*

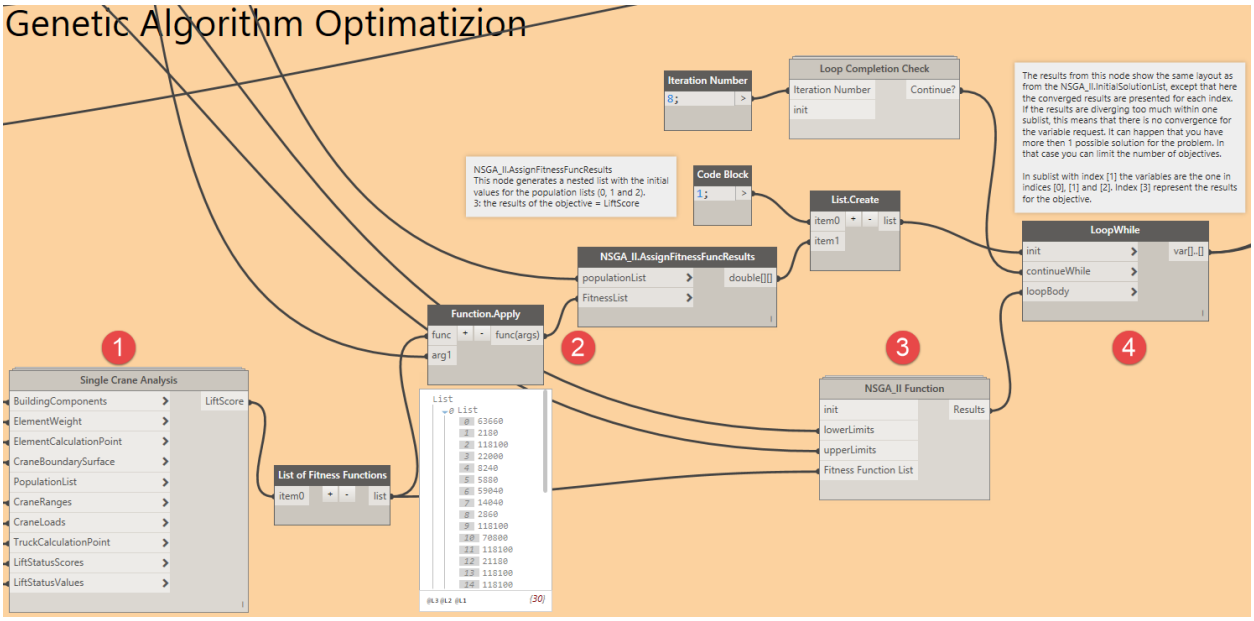
**(2)** The sub list with the last indices represent the objectives. The number of sub lists depends on the numObjectives value. The objectives are all zero when they come out of InitialSolutionList and will be overwritten in the *NSGA_II.AssignFitnessFuncResults* node.
*In the example below: 1 sub list (for 1 objective) with 10 zero values.*




The number of variables can be changed by adding or removing an index to both the *Lower Limit List* and *Upper Limit List* nodes (they must have the same number of indexes).

## GENETIC ALGORITHM OPTIMIZATION

This is the core of the optimization script. The nodes in this part are implemented with the NSGA algorithms. NSGA is a non-domination based genetic algorithm for multi-objective optimization. It works in the following way: An initial population is initialized and sorted. Each individual in the population is sorted based on their fitness (determined by the fitness functions). The parents for the next generation of population are selected by using binary tournament selection. These parents create the next generation with crossover and mutation operators. This continues until a goal is reached or until a set number of generations has been created.




### *1 - Fitness Functions*

The *Single Crane Analysis* is a custom node that is designed that way to read the population list variables (in this case a list of two variables: U-division and V-division). Each of these lists contains a given amount of randomized values (amount is equal to the population size). The custom node contains the *Element.SingleCraneAnalysis* custom node, which is the backbone of this calculation.

The *List of Fitness Functions* node represents the number of functions to be optimized.

### *2 - Assign Fitness Function Results*

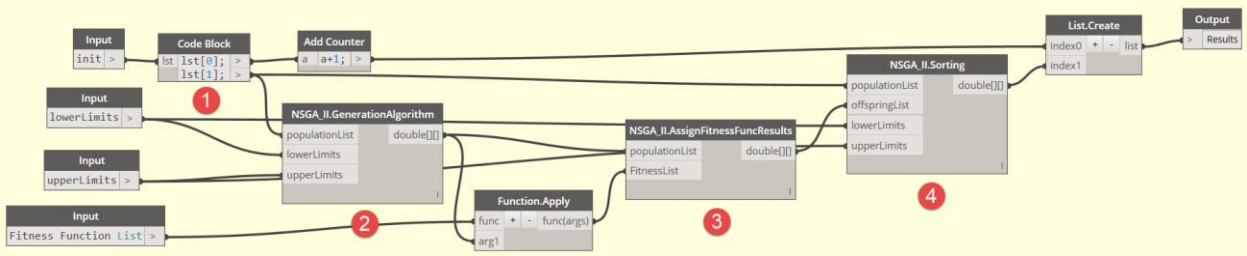
The NSGA_II.AssignFitnessFuncResults node gets the list of objectives as input parameters and assigns them to the populationList input port. By using the Function.Apply node, this workflow enables Optimo to accept objective functions as nodes or packages of nodes.

The population list is also assigned to the *Armadillo Spatial Fitness Function* through the

*Function.Apply* node, but in the custom node, only the variables lists are considered and not the objectives lists. This node evaluates the fitness of each member of the population and sends that data to the AssignFitnessFuncResults node, where the zeroes that were created in the initial population node get overwritten.

### *3 - NSGA Genetic Algorithm Function*

The content of the *NSGA_II Function* node is displayed below. The function of this node is to run the NSGA II algorithm recursively to generate the specified number of generations.




**(1)** The first input to this custom node, "init" is a list containing two elements:
The first element in the list (index [0]) is a counter which keeps track of how many times the function has run. This is needed in the *Loop Completion Check* node (discussed later).
The second element (index [1]) in the list is the population from the previous generation.

**(2)** This second element from the *init* input, gets fed into the *NSGA_II.GenerationAlgorithm* node as an input along with the *lowerLimits* and *upperLimits* (which serves the same purpose as in the main workspace). The Generation Algorithm creates a new population based on the previous generation, called “children solution set” or “offspring”

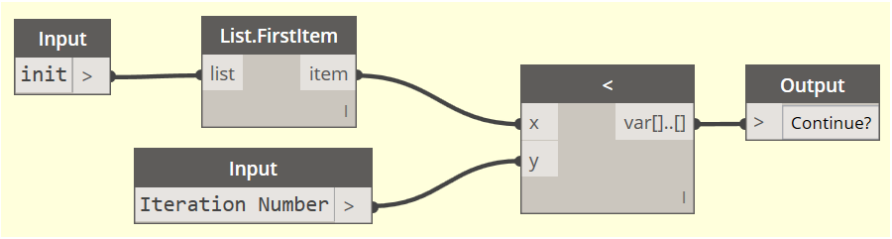
**(3)** The fitness of this new generation is then tested with the same fitness functions as the main Dynamo file (*Single Crane Analysis*), and the fitness function results override the objectives in this new generation.

**(4)** The *NSGA_II.Sorting* node takes the previous generation as well as the current generation, and re-arranges them in order based on their fitness (best results first), using Pareto Front sorting.

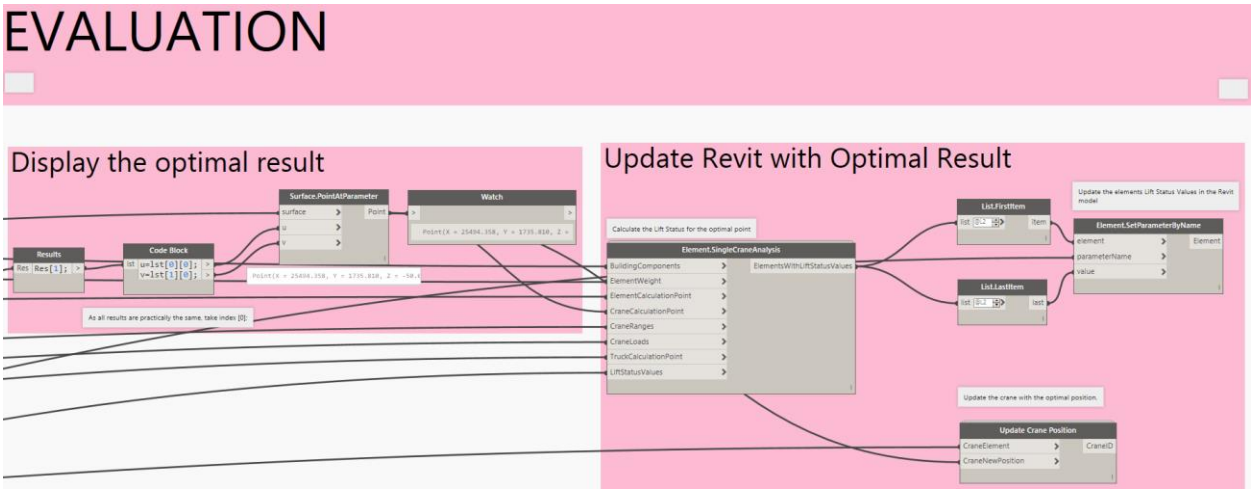
The result gets added to a list which contains all of the previous generations' data.

### *4 – Loop completion check*

The *Loop Completion Check* node simply compares the counter from the *NSGA_II Function* node and compares it to the *Iteration Number* (the max number of iterations that the algorithm should run). When the counter reaches the *Iteration Number*, the algorithm stops looping and displays the result at the output of the *LoopWhile* node.



*Evaluation of Results*




## DISPLAY THE OPTIMAL RESULT

When you add some watch nodes to the output of the *LoopWhile* node (at the end of the optimization phase), then you get results in this form:

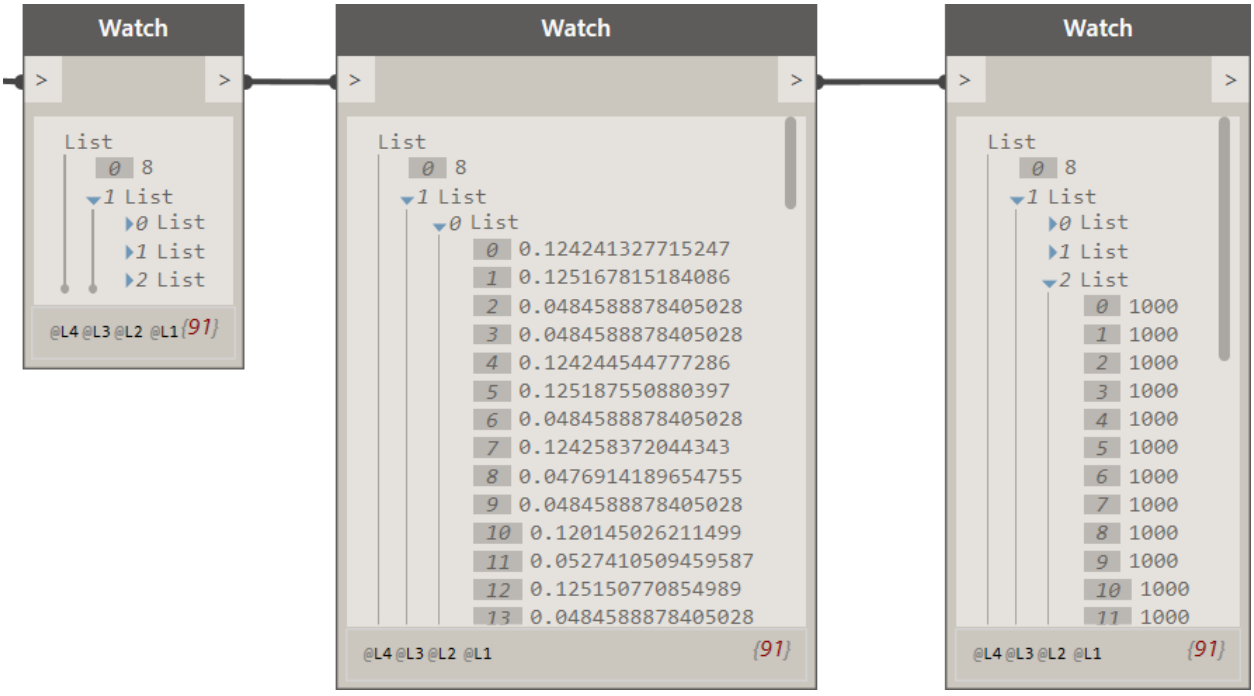



*FIG. 1 - RESULTS AFTER OPTIMIZATION*

The results list consists of two main lists:

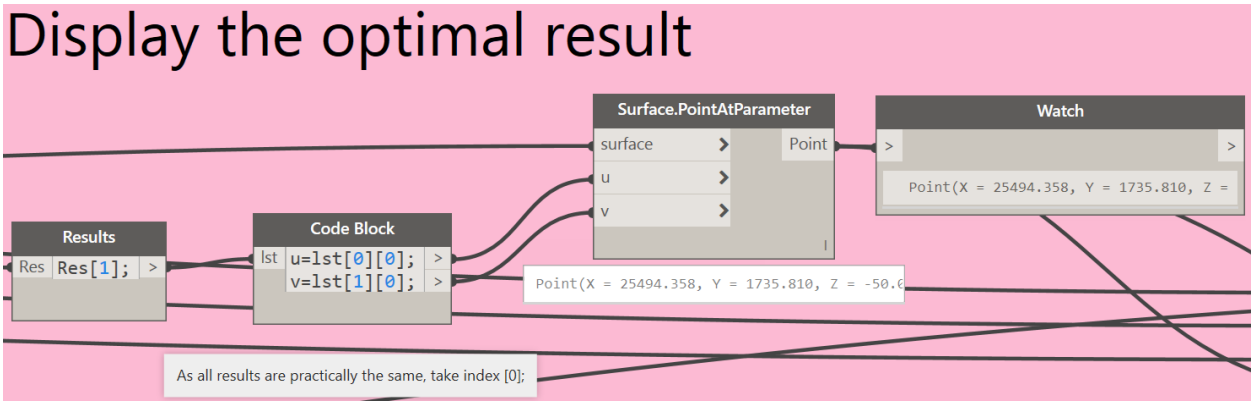
(1) The first item (a[0]) is the number of iterations that has been applied. In this case the genetic algorithms have been called for 8 runs.

(2) The second item (a[1]) represents the final population list (variables) and the achieved objectives (Lift Score). In this case the population size used is 30.
The first 2 sub lists contains the variables:
a[1][0] = the result for the optimal U-division
a[1][1] = the result for the optimal V-division
The last sub list (a[1][3]) contains the resulting Lift Score for each population.

There is a single optimal solution if all values in one sub list are equal.

In case of an optimal solution you can take the first item of each population variable and connect it in this case with the *Surface.PointAtParameter* node, to create the point that represents the optimal crane position.

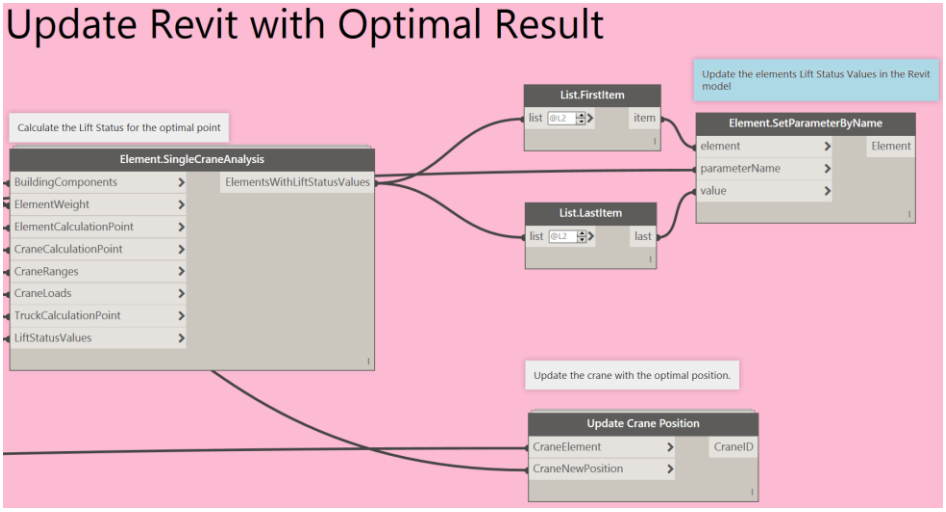
When the results are not all equal, then there is no real good convergence found, which could mean there are multiple solutions, or which means that the *Number of Iterations* that has been set is too small.



### UPDATE REVIT WITH THE OPTIMAL RESULT

That optimal point is then used to recalculate the ElementWithLiftStatusValues and use these results to update the Revit model with the Lift Status values and the optimal crane position.

This is similar to what is explained in [this section].



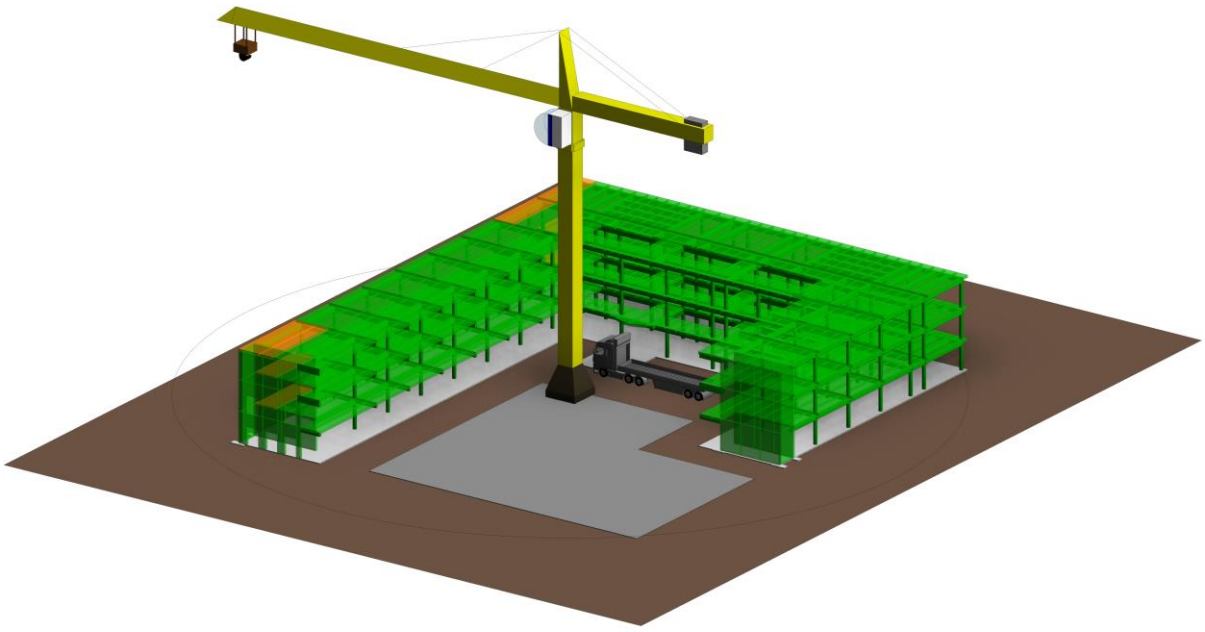



*FIG. 15 - CRANE POSITION AFTER OPTIMIZATION*

## Double Crane - Method 1 Situation Calculation

In this method the current combined position of **two** tower cranes is analyzed and evaluated. The result is a Lift Score and Lift Status Values assigned to the Revit elements, which can be used as base for View Filters then.

### DATASETS

**REVIT**

Start model:
*Double Crane Optimization – Case 2 – 00 Start.rvt*

Finalized model:
*Double Crane Optimization – Case 2 – 01 Situation Calculation.rvt*

**DYNAMO**

Workspace:
*DoubleCraneAnalysis - 01 Situation Calculation.dyn*

Custom nodes:
*From the BIM4Struc.CraneAnalysis package*

External input:
LiftRangeCapacity.xlsx

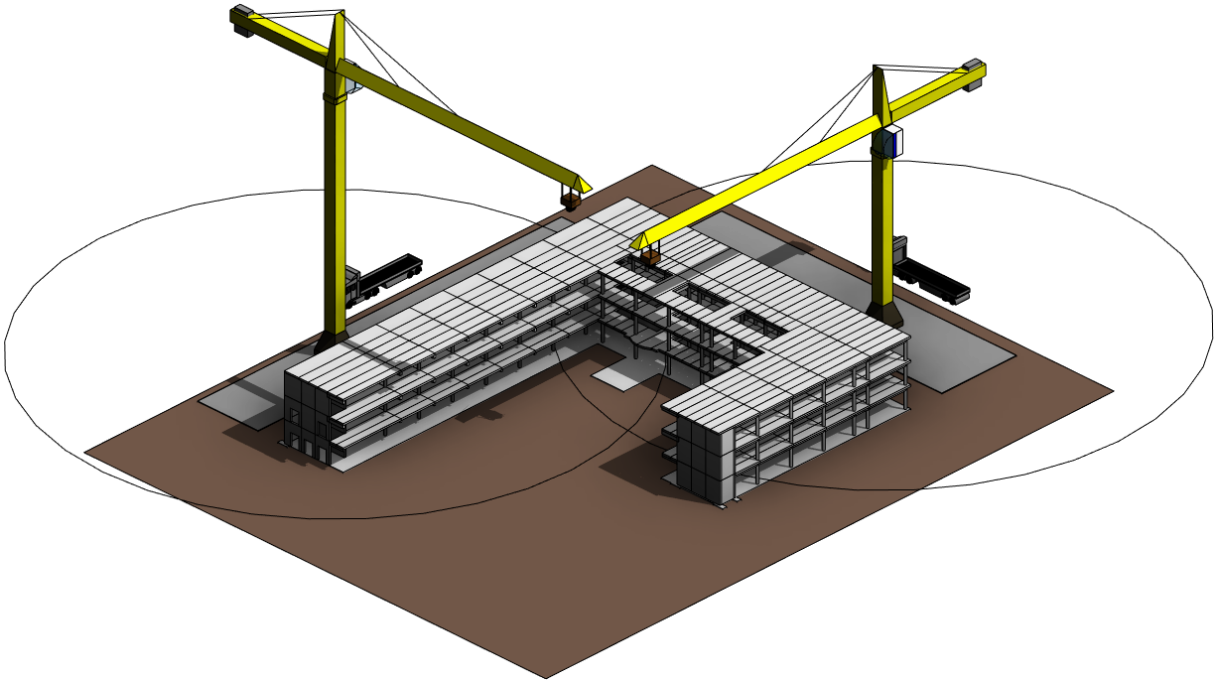
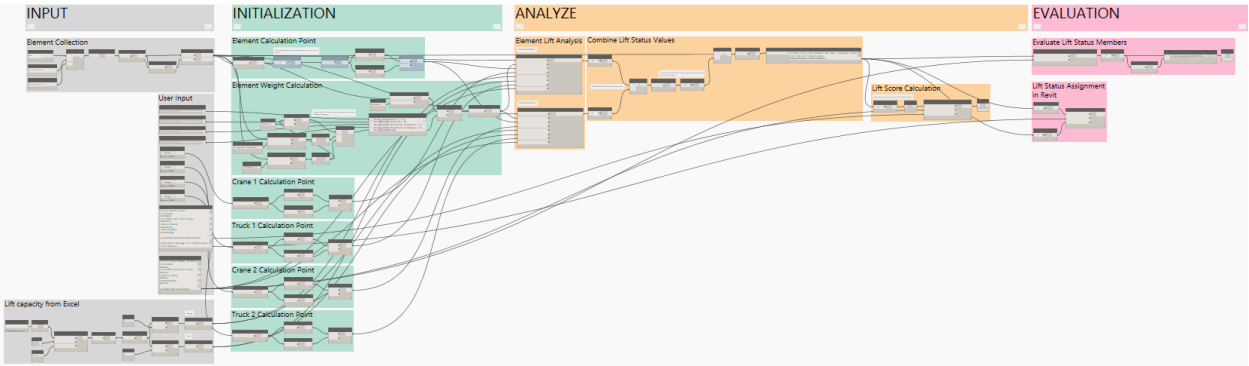


*FIG. 16 - DOUBLE CRANE OPTIMIZATION – BEFORE ANALYSIS*

### *General Overview*

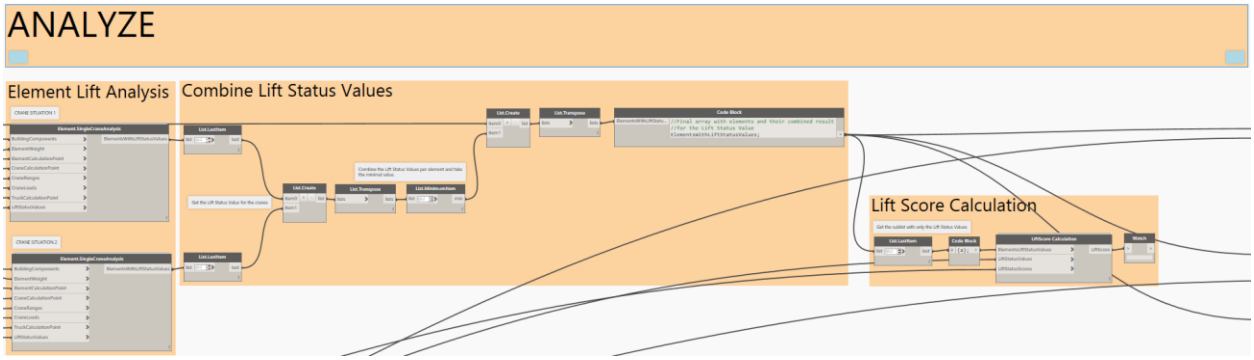



The method consist of 4 phases:

- **Input:** Collecting elements from Revit, reading external data and custom user input. Read this section.
- **Initialization:** Section used for calculating and consolidating the information from the user input. This phase is similar to the one for a Single Crane. The difference is the fact that the calculation points of 2 cranes and 2 trucks is returned here.
- **Analyze:** Performs the actual analysis for both cranes and combines the results in to one Lift Score.
- **Evaluation:** Evaluates the Lift Score and returns feedback in Revit and Dynamo.

### *Analyze*

The *Analyze* phase doesn't differ that much from the one from a Single Crane analysis.




#### ELEMENT LIFT ANALYSIS

In this part the *Element.SingleCraneAnalysis* is executed on all elements, for both cranes individually. This means that the resulting Lift Status values don't take into account the influence of the other crane, yet…

The contents of the custom node are explained in this section. The output for both nodes look like the image below.

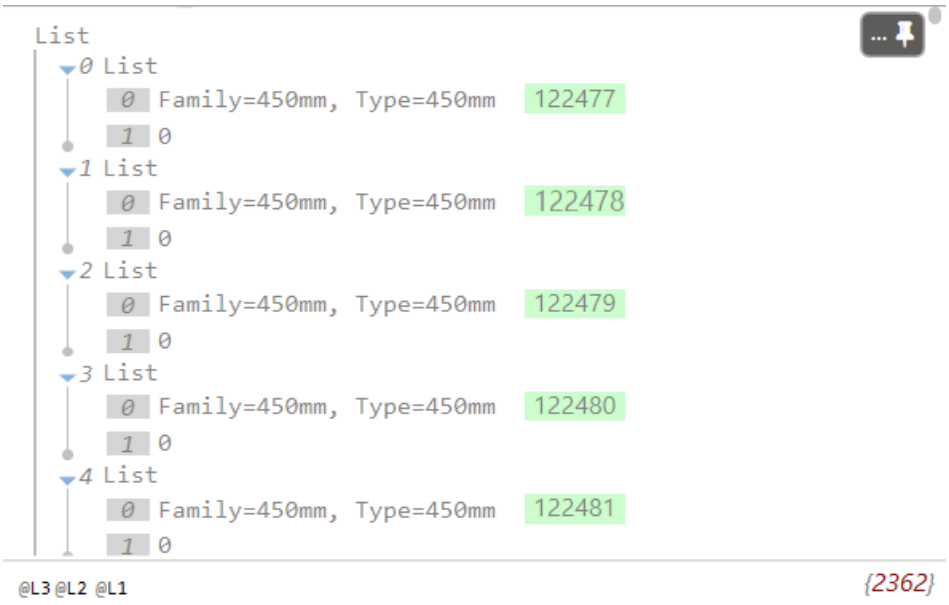
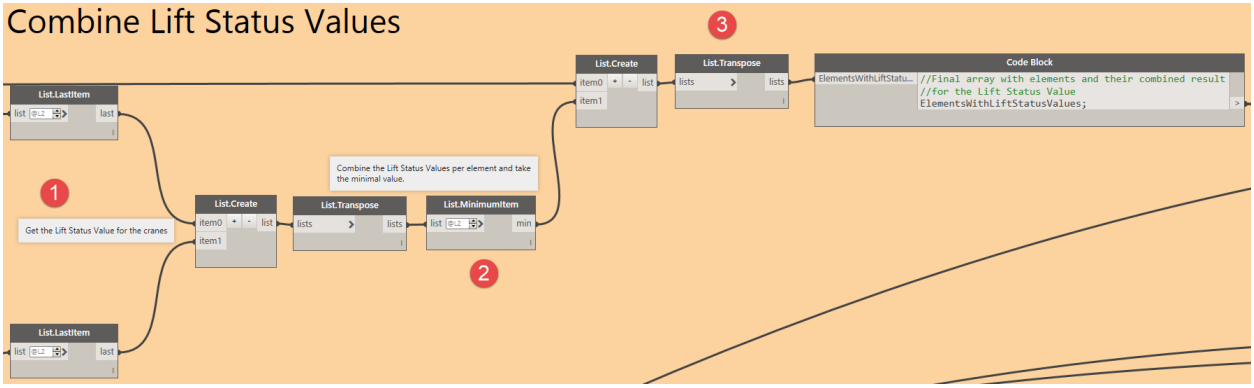



*FIG. 17 - ELEMENTS WITH LIFT STATUS VALUES*

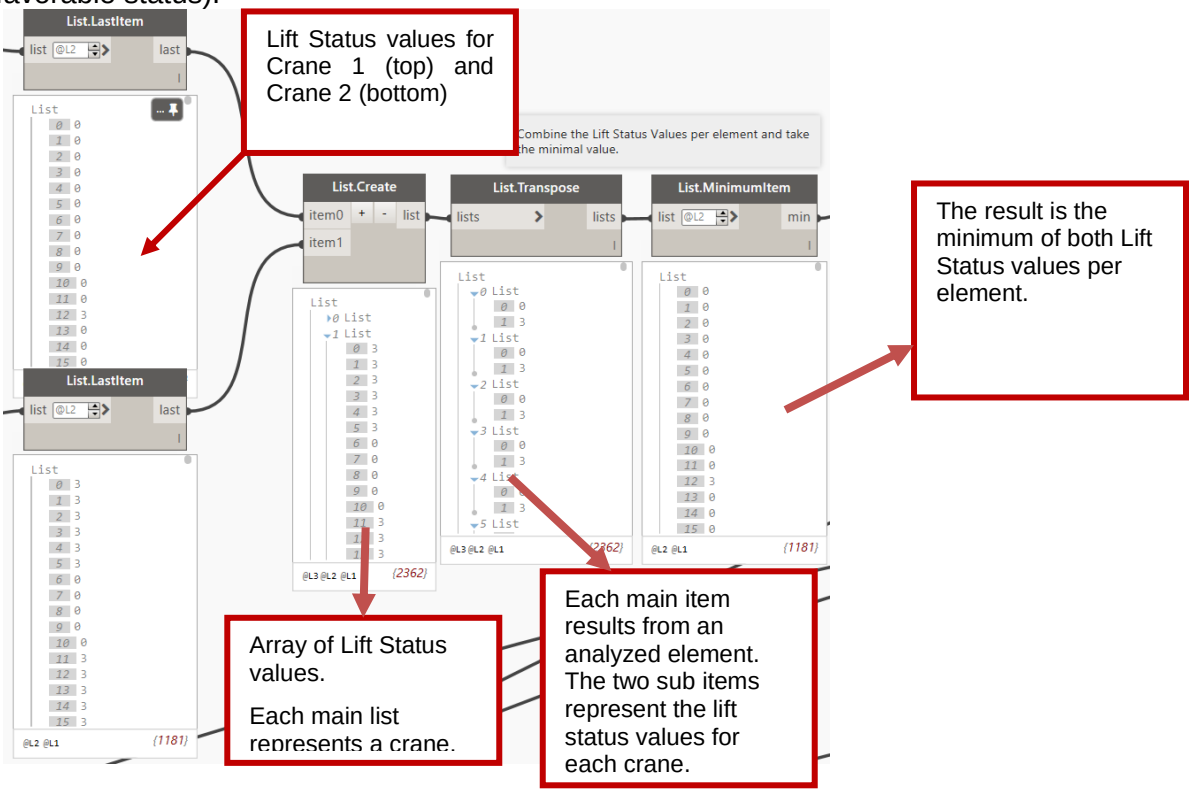
## COMBINE LIFT STATUS VALUES

The Lift Status results for all the elements are currently only based on the position of 1 crane. This means that an element can be lifted by crane 1 (Lift Status 0) but not by crane 2 (Lift Status 2). But generally the element is 'Liftable' (Lift Status 0). To get to this result we need to combine the individual outputs. This is done in the node group below.



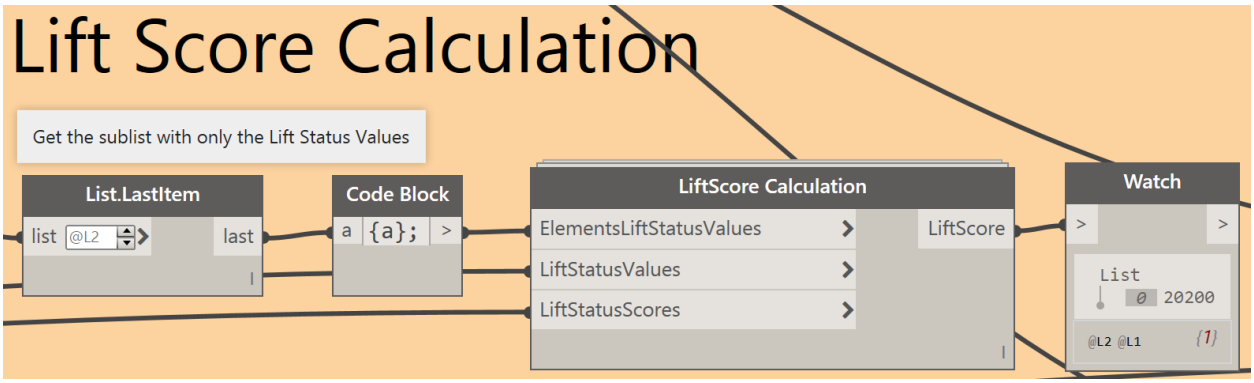

**(1)** Get the *Lift Status* values for each crane analysis. The values are stored in the last sub item of each result item.

**(2)** Combine the results in a new transposed list and detect the minimal value (thus the most favorable status).




**(3)** Then use these results to create a new array of *ElementsWithLiftStatusValues*. This array is later used for the *Lift Score* calculation.

### LIFT SCORE CALCULATION




This part works similar as for a Single Crane Situation Calculation. Read this section.

#### *Evaluation*

This part is explained in the Single Crane Situation Calculation.

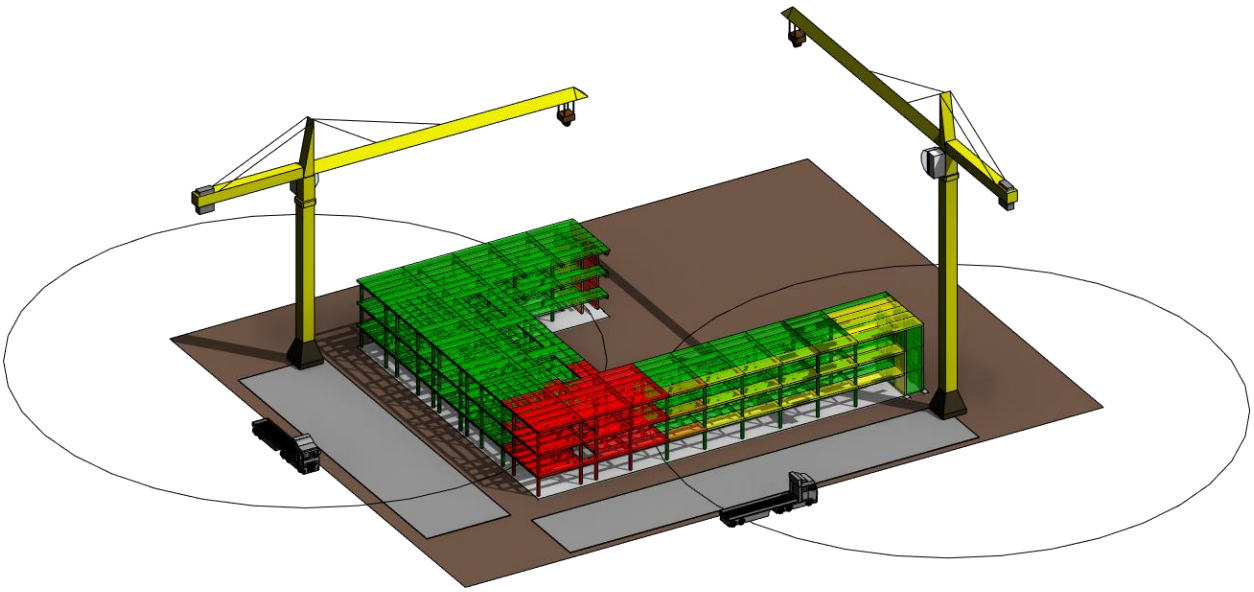


*FIG. 18 - DOUBLE CRANE SITUATION CALCULATION - RESULT IN REVIT*

## Double Crane – Method 2a Parametric Run

The script for a Parametric Run for a double crane setup is identical to a Single Crane Parametric Run, except for the Analyze part. In that phase some additional processing of the data is needed to combine the results of both cranes into one Lift Status value per element.

### DATASETS

**REVIT**

Start model:
*Double Crane Optimization – Case 2 – 00 Start.rvt*

Finalized model:
*Double Crane Optimization – Case 2 – 02 Parametric Run result.rvt*

**DYNAMO**

Workspace:
*DoubleCraneAnalysis – 02a Parametric Run.dyn*

Custom nodes:
*From the BIM4Struc.CraneAnalysis package*

External input:
LiftRangeCapacity.xlsx

### *General Overview*

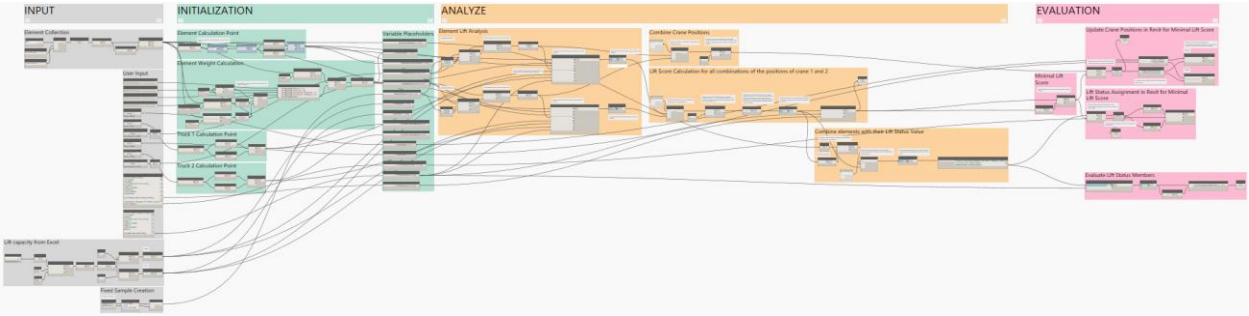



*FIG. 19 - DOUBLE CRANE - PARAMETRIC RUN*

The method consist of 4 phases:

- **Input**
  Collecting elements from Revit, reading external data and custom user input. Also a fixed set of samples is defined on which the analysis needs to run. This phase is already explained in this section.

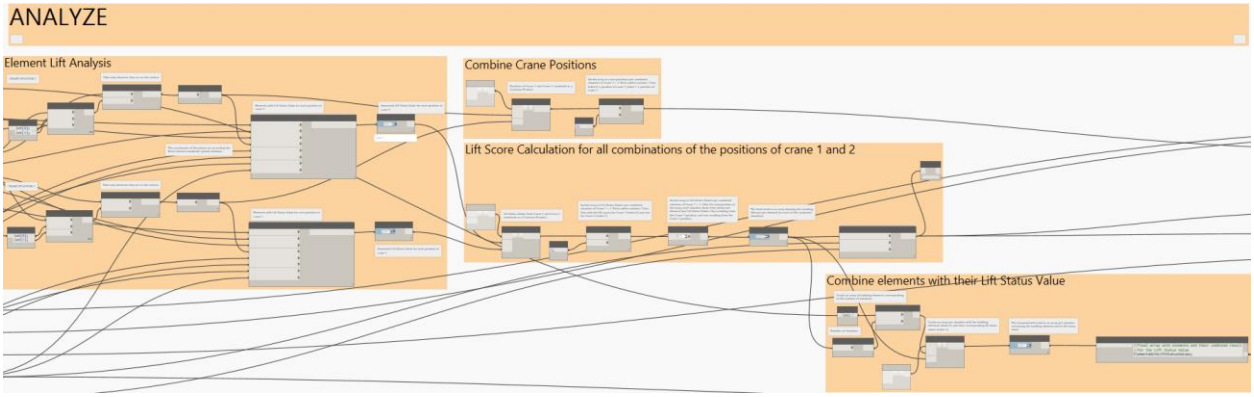
- **Initialization**
  Section used for calculating and consolidating the information from the user input. For more information, read this section.

- **Analyze**
  Performs the actual crane analysis on the sample list and returns the Lift Scores for each situation.

- **Evaluation**
  Evaluates the Lift Score by finding the situation with the minimal score and returns feedback in Revit and Dynamo.

### *Analyze*

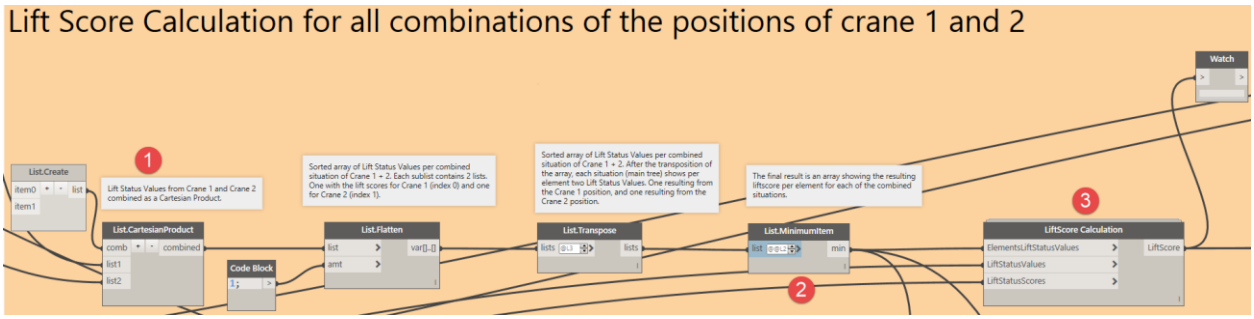
The “Analyze” phase is the part where the input and initialized data are put together and analyzed. The result is a list of Lift Scores, one for each situation.




#### ELEMENT LIFT ANALYSIS

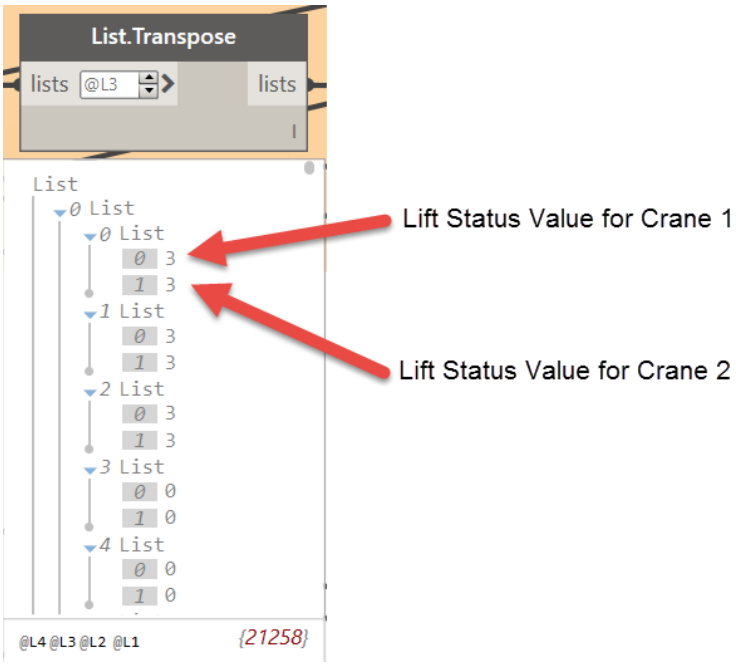
This analysis of the cranes is similar as for a single crane, but as we deal with two cranes, it is copied and executed twice.

#### LIFT SCORE CALCULATION




**(1)** The Lift Status values for both crane configurations need to be combined, as explained in previous method. In this parametric run we have results for a fixed number of positions for both cranes. This means that we need to combine the Lift Status values for all possible combinations of both cranes.

This can be achieved by using the *List.CartesianProduct* node.



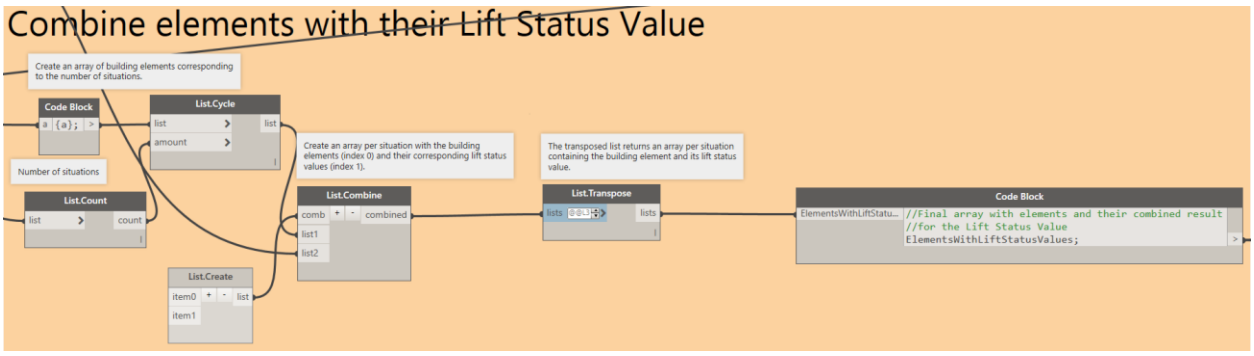

**(2)** Then process the results in a way you get the final result for each element in an array where the main list represents the crane position combinations.

**(3)** Then finally use the values to calculate the Lift Score.

Finally, the main length of this list should be equal to this formula: $S = P^n$
where: S = number of situations
P = number of sample points
n = number of cranes

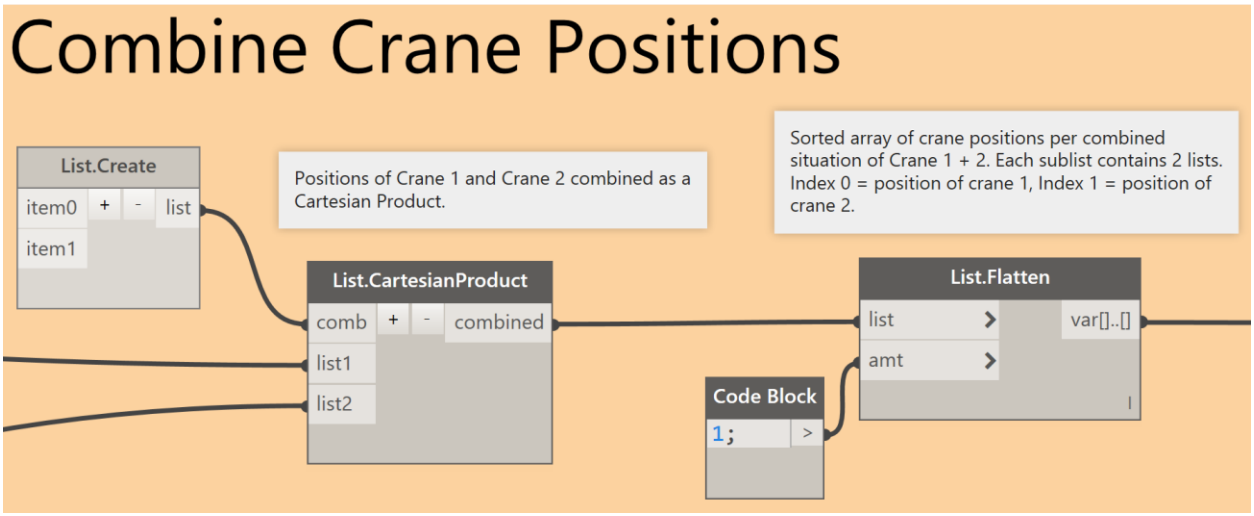
## COMBINE ELEMENTS WITH THEIR LIFT STATUS VALUE

Once the final Lift Status values are known they need to be assigned to the Revit elements. Therefore a sub list is created as in previous examples, for each crane combination. This will be handy in a Parametric Run with Capture.



### COMBINE CRANE POSITIONS

This part is not influencing the analysis, but this is done in order to have the crane position combinations grouped in a list, with the same length as the Lift Score results list.




#### *Evaluation*

The *Evaluation* phase is explained in a previous section.

## Double Crane – Method 3 Genetic Optimization

As explained in the Single Crane – Method 3, the use of Genetic Algorithms can speed up the process in finding the optimal solution of two cranes. The setup of this method is similar to the one for a single crane, with some small additions described below.

In this example the **Optimo** package for Dynamo is used again.

### DATASETS

**REVIT**

Start model:
*Double Crane Optimization – Case 2 – 00 Start.rvt*

Finalized model:
*Double Crane Optimization - Case 2 - 03 Optimization result.rvt*

**DYNAMO**

Workspace:
*DoubleCraneAnalysis – 03 Genetic Optimization.dyn*

Custom nodes:
*BIM4Struc.CraneAnalysis package*
*Optimo package*

External input:
LiftRangeCapacity.xlsx

### *Representation of the Optimization Problem*

Before you start building up the script, it is important to first represent the optimization problem which consists of an “objective”, “design variables” or “population” and the “design variables domain”.

#### OBJECTIVE

With the optimization we want to find a crane configuration that has a minimal *Lift Score.* The *Lift Score* is calculated depending on the lift status of each calculated element in the construction. Read more in this section.

#### DESIGN VARIABLES

The design variables for this optimization are expressed with two parameters:

- U-division of the boundary surface for the 1st crane
- V-division of the boundary surface for the 1st crane
- U-division of the boundary surface for the 2nd crane
- V-division of the boundary surface for the 2nd crane

These divisions are used to define a point on the surface, which could be a potential crane position, until an optimal solution is found.

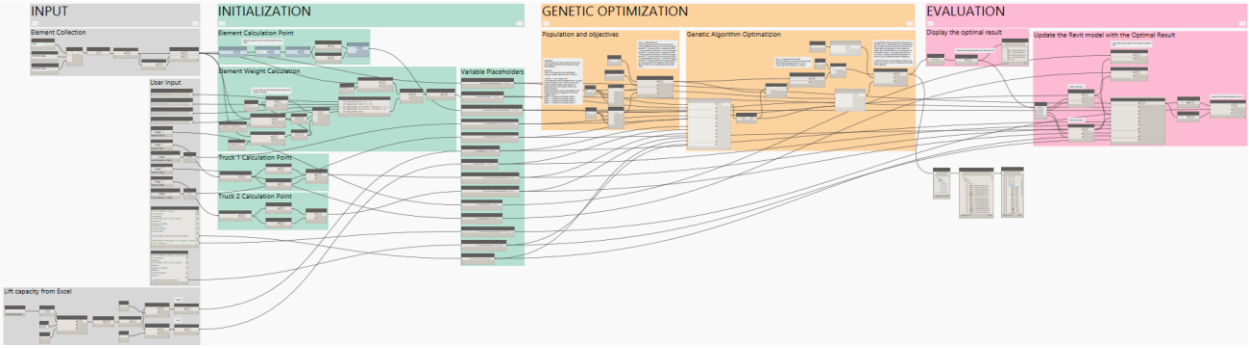
#### DESIGN VARIABLES DOMAIN

We need to give some restrictions for the design variables and give the optimization routine a boundary within to search an optimal solution for. In this case the domains are set as follows:

- $0 \leq U_1 \leq 1$
- $0 \leq V_1 \leq 1$
- $0 \leq U_2 \leq 1$
- $0 \leq V_2 \leq 1$

The UV-divisions need to be numbers between 0 and 1 to work with the Surface.PointAtParameter node in the analysis part. But this could be constrained between 0.2 and 0.8 as well.

### *General overview*




The total script consists of 4 phases:

- **Input**
  Collecting elements from Revit, reading external data and custom user input. Also a fixed set of samples is defined on which the analysis needs to run. This phase is already explained in this section.

- **Initialization**
  Section used for calculating and consolidating the information from the user input. This phase is already explained in this section.

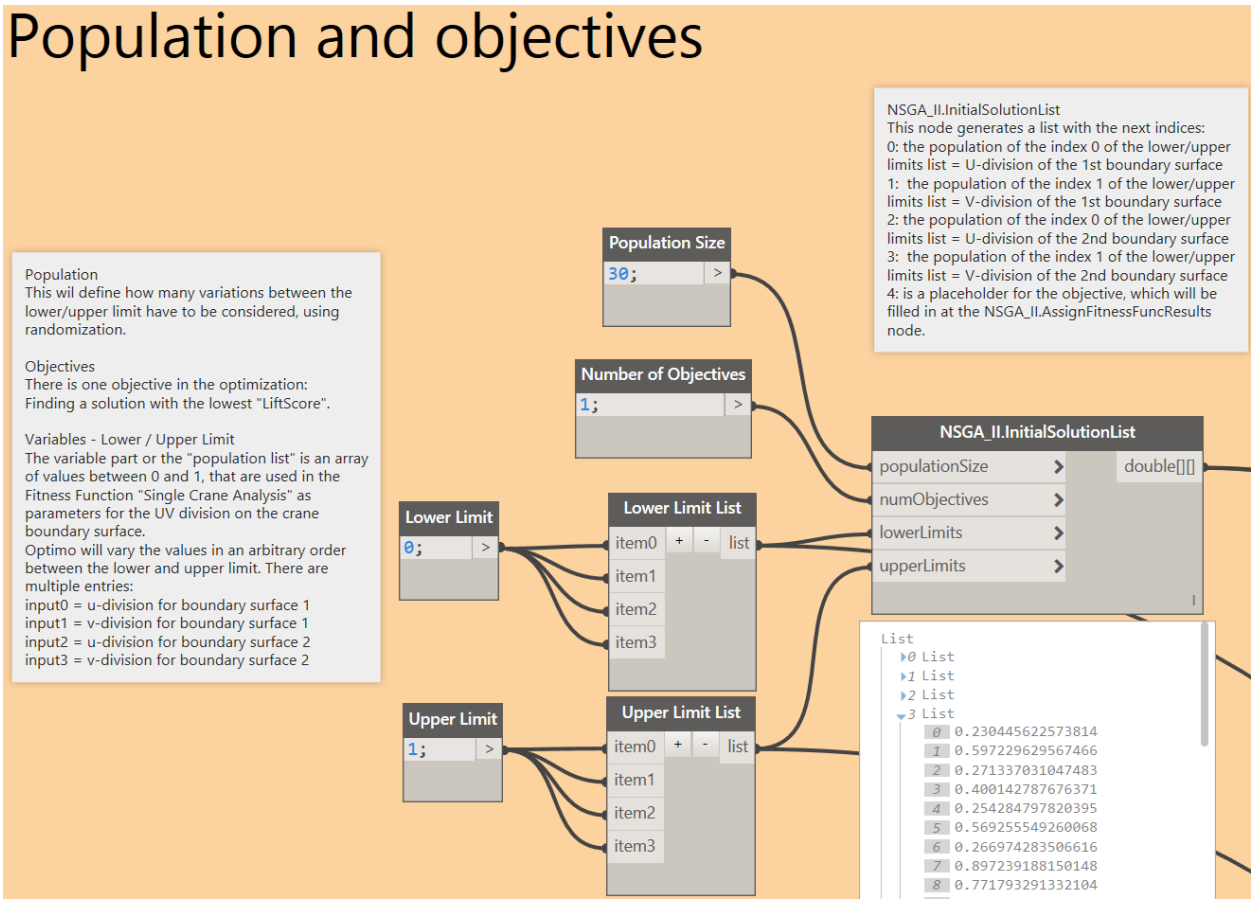
- **Genetic Optimization**
  The initial population list is generated and the design variables are optimized using Genetic Algorithms until the objective of a minimal Lift Score is reached.

- **Evaluation**
  Evaluates the Lift Score and returns feedback in Revit and Dynamo.

### *Genetic Optimization*

This phase generates the initial solution list, based on design variables within a defined domain, and calculates the *Lift Score* with a fitness function. In the next iterations genetic algorithms (GA) are used to define a better population until an optimal solution is found or the number of iterations is reached.

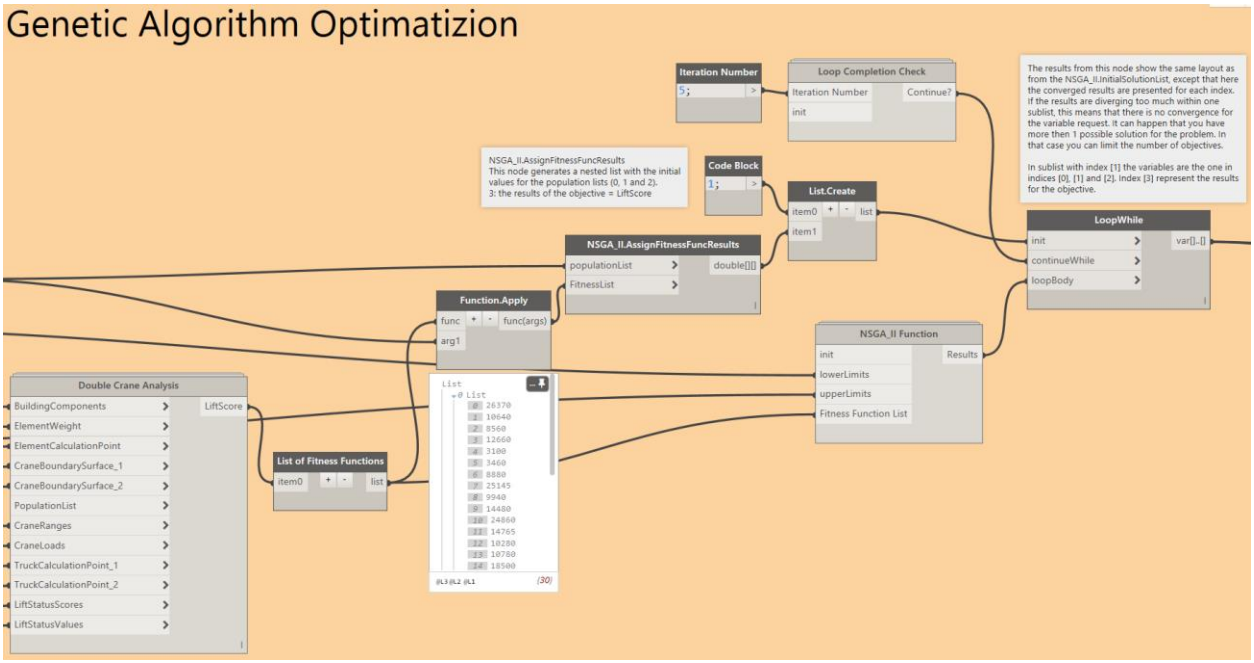
## POPULATION AND OBJECTIVES



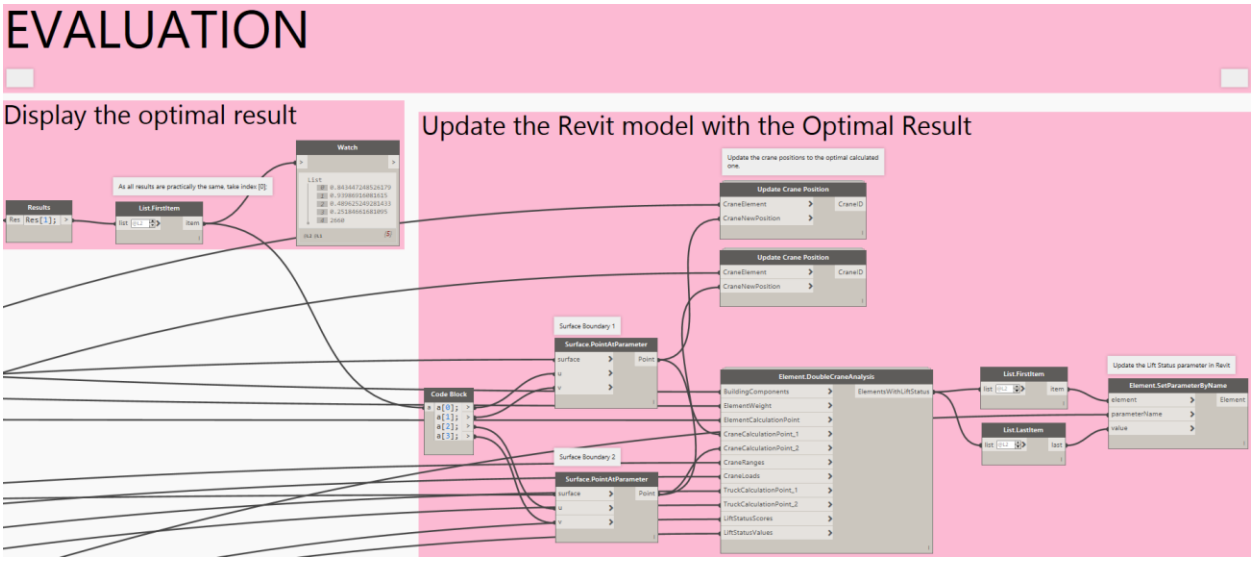

Read more about this on [Single Crane population]. In this case the *Design Variables* consist of **4** lower and upper limits, which results in list of 5 items: index 0-3 are the UV parameters for both surfaces, index 4 is the placeholder for the *Lift Score*.

### GENETIC ALGORITHM OPTIMIZATION

More information can be found in this section. The analysis makes use of the *Double Crane Analysis* fitness function.



*Evaluation of Results*




## DISPLAY THE OPTIMAL RESULT

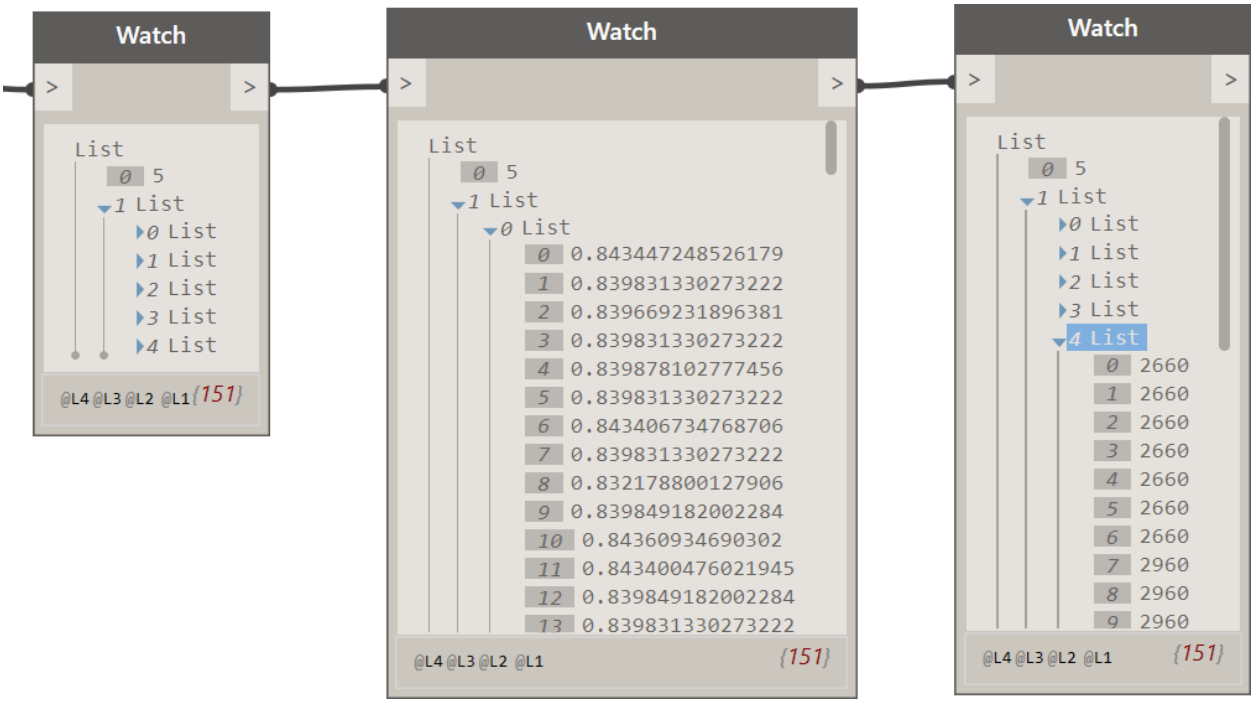



*FIG. 20 - OPTIMAL RESULT FOR A DOUBLE CRANE SETUP*

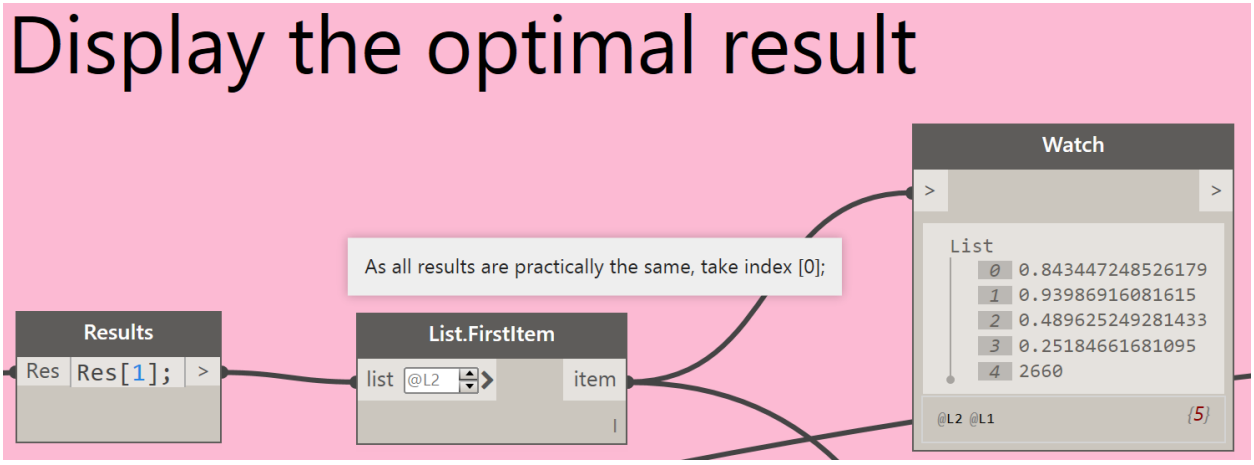
The results list consists of two main lists:

(1) The first item (a[0]) is the number of iterations that has been applied. In this case the genetic algorithms have been called for 5 runs.

(2) The second item (a[1]) represents the final population list (variables) and the achieved objectives (Lift Score). In this case the population size used is 30.
The first 4 sub lists contains the variables:
a[1][0] = the result for the optimal U-division parameter for the 1st surface
a[1][1] = the result for the optimal V-division parameter for the 1st surface
a[1][2] = the result for the optimal U-division parameter for the 2nd surface
a[1][3] = the result for the optimal V-division parameter for the 2nd surface
The last sub list (a[1][5]) contains the resulting Lift Score for each population.

<u>There is a single optimal solution if all values in one sub list are equal.</u>

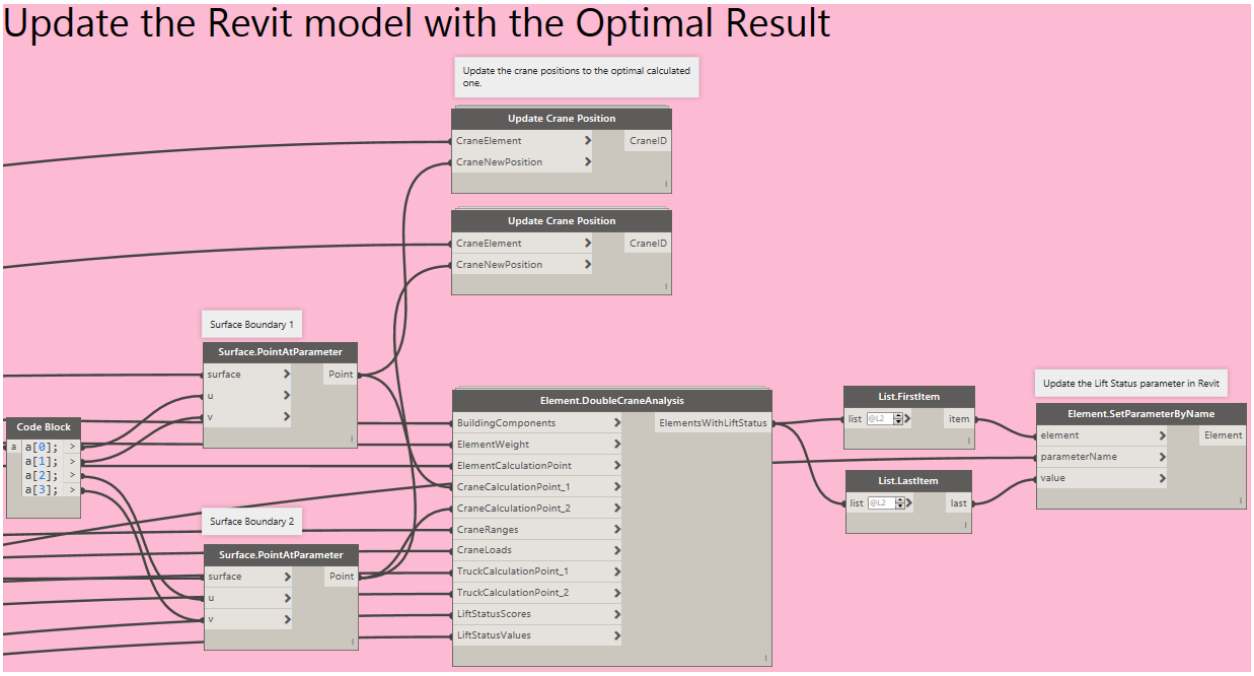
In case of an optimal solution you can take the first item of each population variable and connect it in this case with the appropriate *Surface.PointAtParameter* node, to create the point that represents the optimal crane position.



### UPDATE REVIT WITH THE OPTIMAL RESULT

The optimal points are then used to recalculate the ElementWithLiftStatusValues with the *Element.DoubleCraneAnalysis* node. The Revit model can then be updated with the Lift Status values and the optimal crane position.

This is similar to what is explained in [this section for the single crane optimization.]



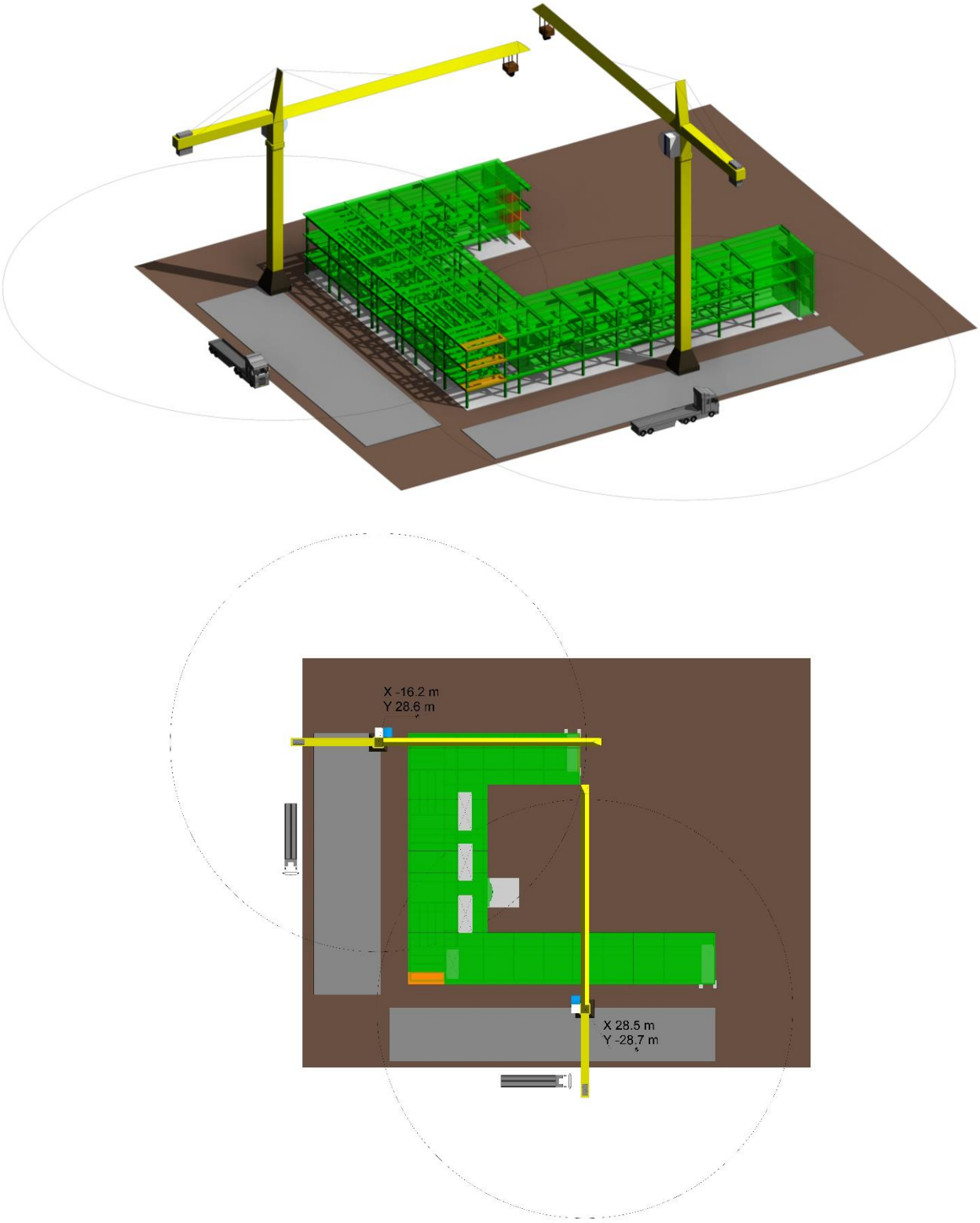



*FIG. 21 - UPDATED REVIT MODEL AFTER OPTIMIZATION*

## Crane Analysis in the Cloud with Project Fractal

The previous scripts perform the crane analyses within the Revit model. These methods can take some time to generate all the design options. Besides that, the Revit model only shows the latest result and doesn't store all the design options or design parameters.

In this part, we're using a Labs Preview (or alpha version) of Project Fractal, which is a cloud platform which allows to generate and store multiple design.

Project Fractal can only explore parametric models created in **Dynamo Studio**. As Dynamo Studio doesn't link with Revit, this means that the Revit model can't be uploaded to the platform directly. With the version 1.2.0.2871 of Dynamo Studio 2017, it is possible to upload .SAT and .CSV files to the cloud. Therefore we're making an exploded SAT model from the Revit project, which will be analyzed with the same scripts as previously made.

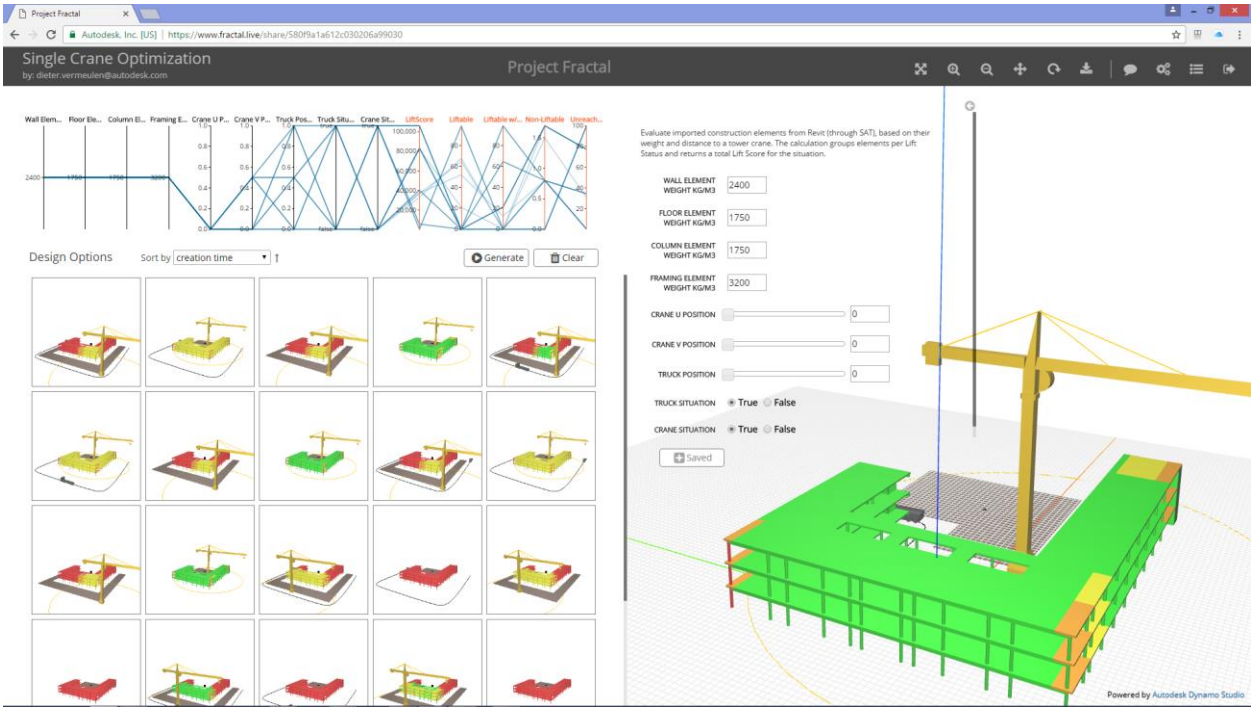



*FIG. 22 - DESIGN OPTION EXPLORATION IN PROJECT FRACTAL*

### *Design Variables*

As this approach with the cloud platform offers more computational resources, we can introduce more design variables in the script. The analysis of the elements is still done according the flowchart, explained in *FIG. 6 - Element Lift Analysis Flowchart*.

These geometrical variables are added in this analysis method:

- Several surfaces are considered for hosting the crane.
- The truck can have several positions along a chosen path

- Several paths are considered for hosting the truck

***Create SAT model from the Revit project***

**DATASETS**

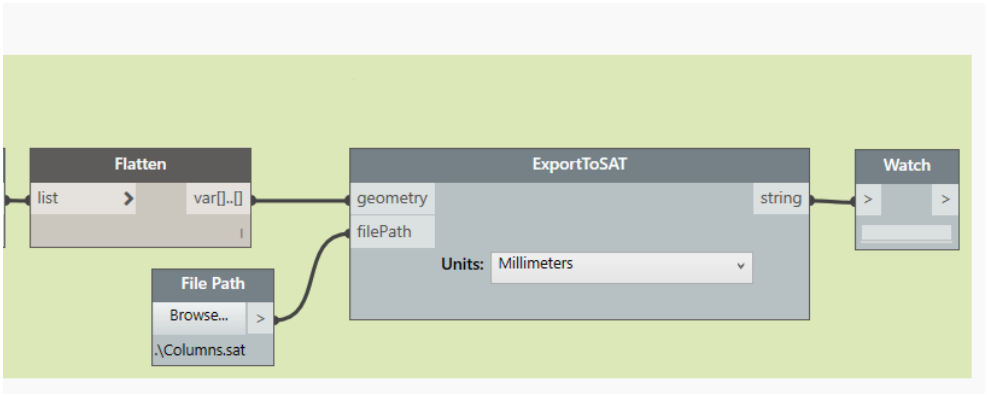
**REVIT**

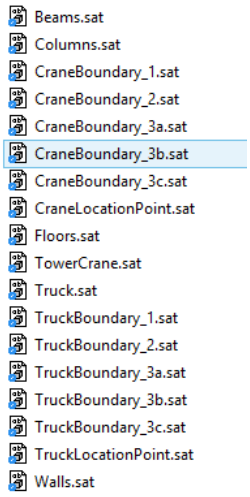
Models
*Single Crane Optimization.rvt*
*Double Crane Optimization.rvt*

**DYNAMO**

Workspace:
*01a Revit Crane Boundary SAT Export.dyn*
*01b Revit Single Crane SAT Export.dyn*

The Dynamo scripts (run through Dynamo in the Revit environment) break up the model in several parts according to the element categories and export the geometries to dedicated .SAT files.




It's necessary to have every element category in a separate SAT file, because they are analyzed differently, when it comes to the Element Weight Calculation (see this chapter).

Besides that we want to use the same location point as in Revit for the truck and crane (which is the insertion point). These are extracted separately from the truck and crane geometry as well.

### *Crane Analysis with Dynamo Studio*

Before starting the optimization in the cloud, we need to build the crane analysis method in Dynamo Studio. Basically you can reuse most of the scripts described above.

The main difference lays in the Geometry Input, which happens through SAT import in this case.

**DATASETS**

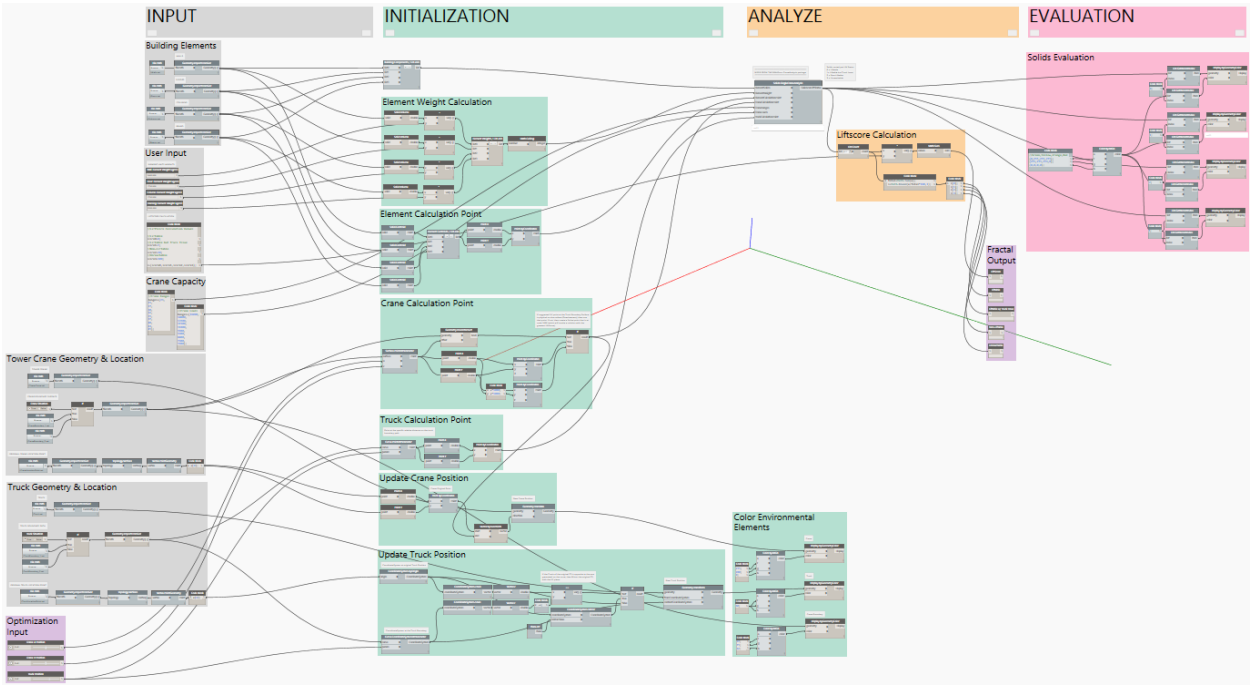
**DYNAMO STUDIO**

Workspaces:
- *02a Single Crane Optimization - Fixed Boundaries.dyn*
- *02b Single Crane Optimization - Variable Boundaries*
- *03a Double Crane Optimization - Fixed Truck Position*
- *03b Double Crane Optimization - Variable Truck Position*

Custom nodes:
- *BIM4Struc.CraneAnalysis package*

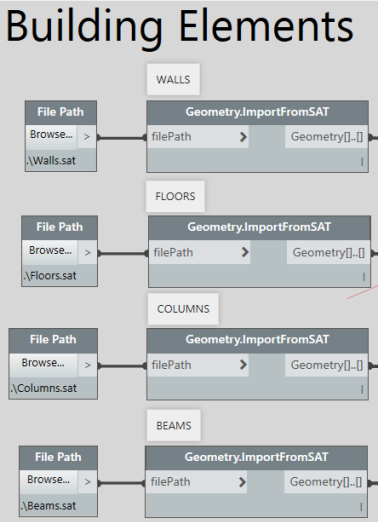
**FRACTAL**

External links:
- *Single Crane Optimization - Fixed Boundaries*
- *Single Crane Optimization - Variable Boundaries*
- *Double Crane Optimization - Fixed Truck Position*
- *Double Crane Optimization - Variable Truck Position*

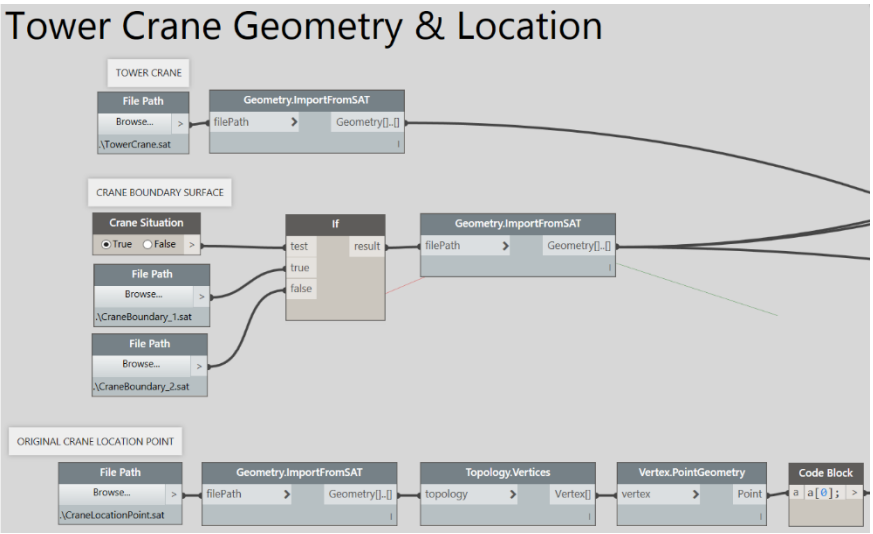



The purple node groups represent the Input & Output parameters that will be used by the Project Fractal interface.

## STEP 1 – INPUT

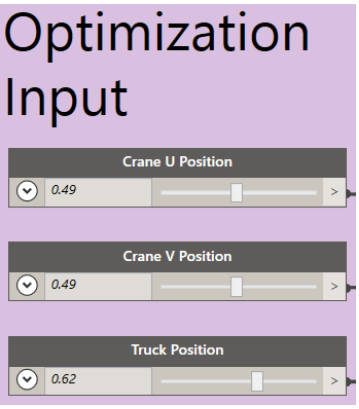



Read the geometry from all the geometry files (SAT) for the columns, beams, walls and floors.




The same is done for the solid geometries of the crane and truck, but this is pure for graphical representation.

Besides that also the tower and crane location points are imported. They represent their calculation points for the further analysis, as explained in previous chapters.




These number sliders represent the design variables for the crane position, but also for the truck position along a chosen path.

### STEP 2 – INITIALIZATION

The *Initialization* phase consists of similar parts as in the previous methods. They can be simplified a bit because of the use of several files per element category.

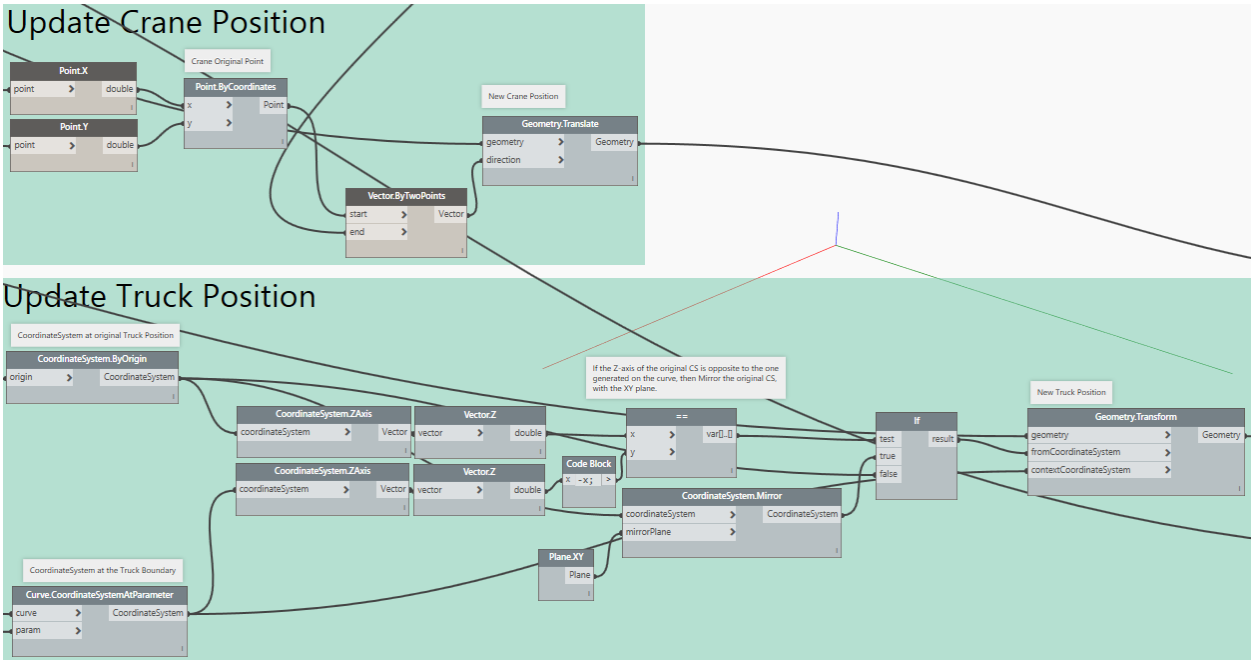
**Element Weights** are calculated based on the volume of each solid and a given unit weight per RC element.

**Element Calculation Point** is based on the *Solid.Centroid*, reduced to Z=0.

**Crane Calculation Point** is based on the imported *Truck LocationPoint.sat* geometry. With the *Geometry.DoesIntersect* node the point is verified to be on the crane boundary surface. If not than a fictive point that is very far away from the structure is generated. This point will invoke a lift status of “unreachable” for each element.

During the initialization the **Crane Position** and **Truck Position** are graphically updated, so that you can see where the crane and truck are positioned for each Lift Score. This comes in handy when running the analysis in Project Fractal.

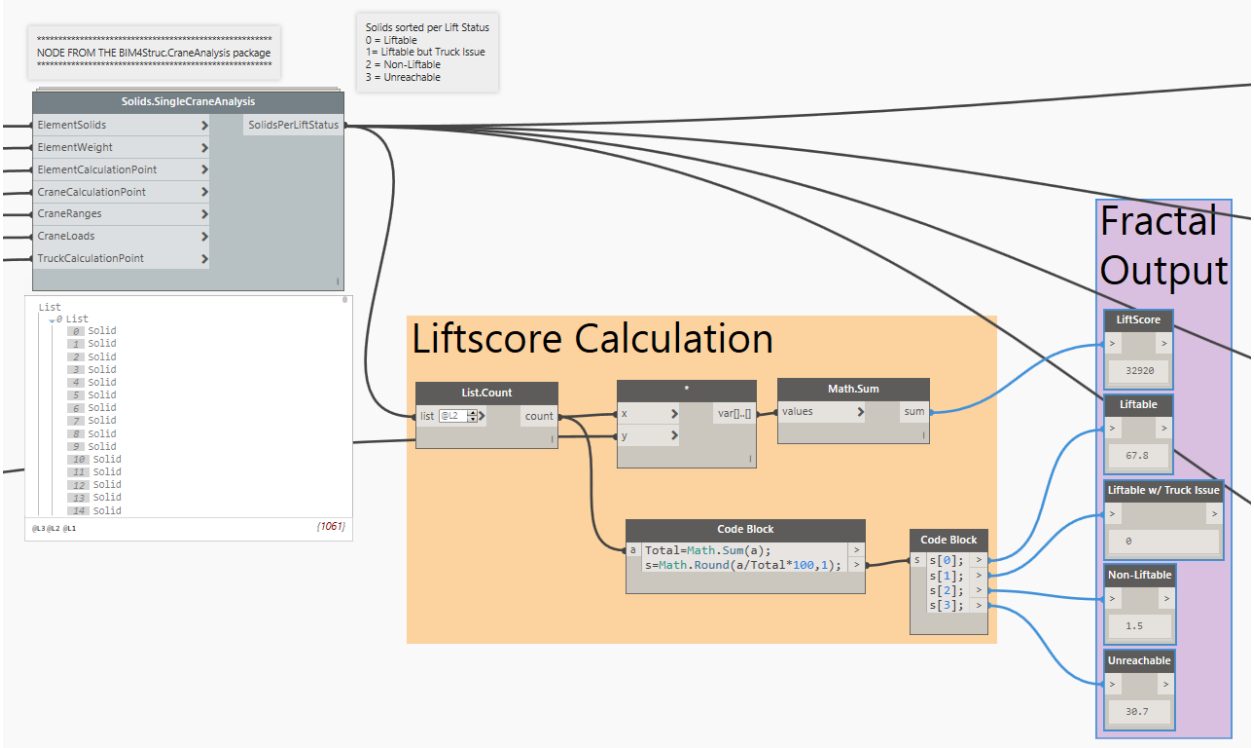
For the crane position a simple *Geometry.Translate* by vector can be used.




For the Truck it’s a bit more complicated as we want to Truck to be aligned graphically with the curves of a truck boundary path. Therefore the truck position is changed with the *Geometry.Transform* node by using Coordinate Systems.

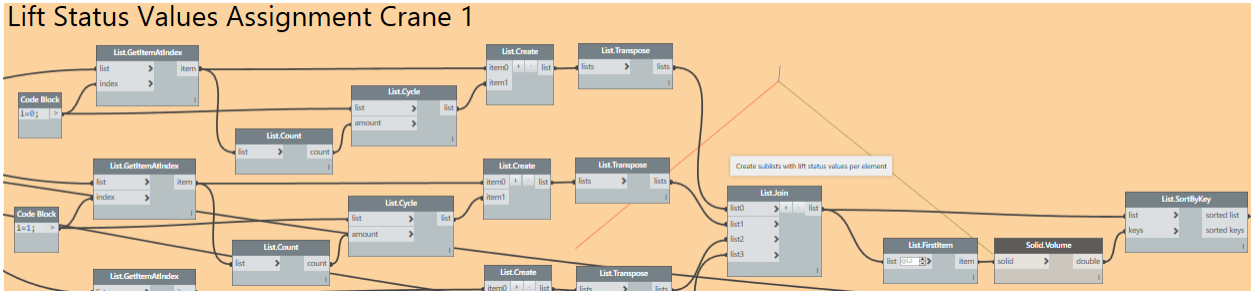
### STEP 3 – ANALYZE

The analysis for a single crane is done in the same way as described in this chapter. The output is in this case limited to an array with solids sorted according their lift status.




The *Lift Score* is then calculated based on the number of elements per status. The purple node group contains *Watch* nodes. These are required when you want to show outcomes in Project Fractal.

The analysis of a double crane situation has a few more operations to find the combined lift status result. In previous cases this was already covered by the custom nodes. In this case each solid gets a lift status assigned in an array. The solids are sorted according their volume, to be sure that the combined lists have the same structure.



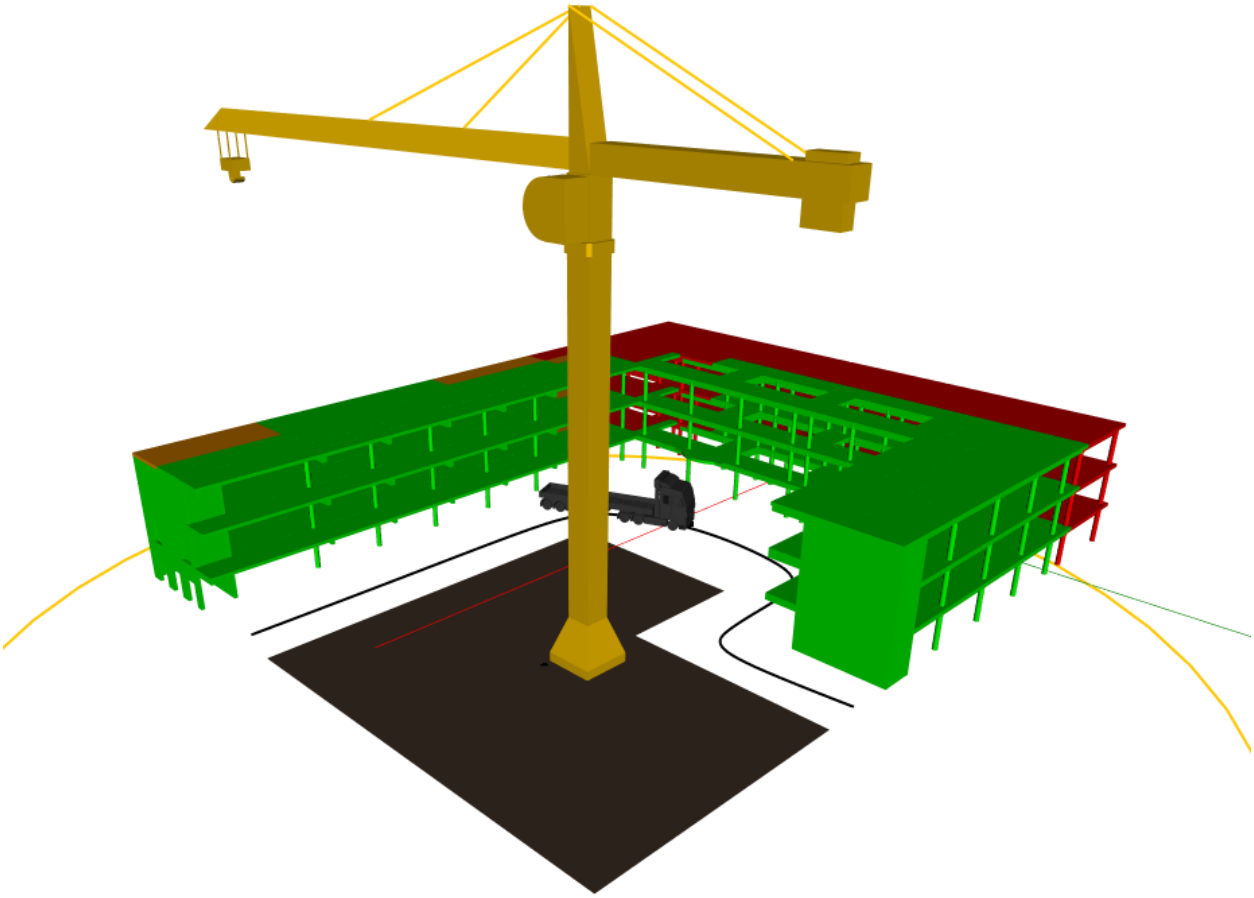
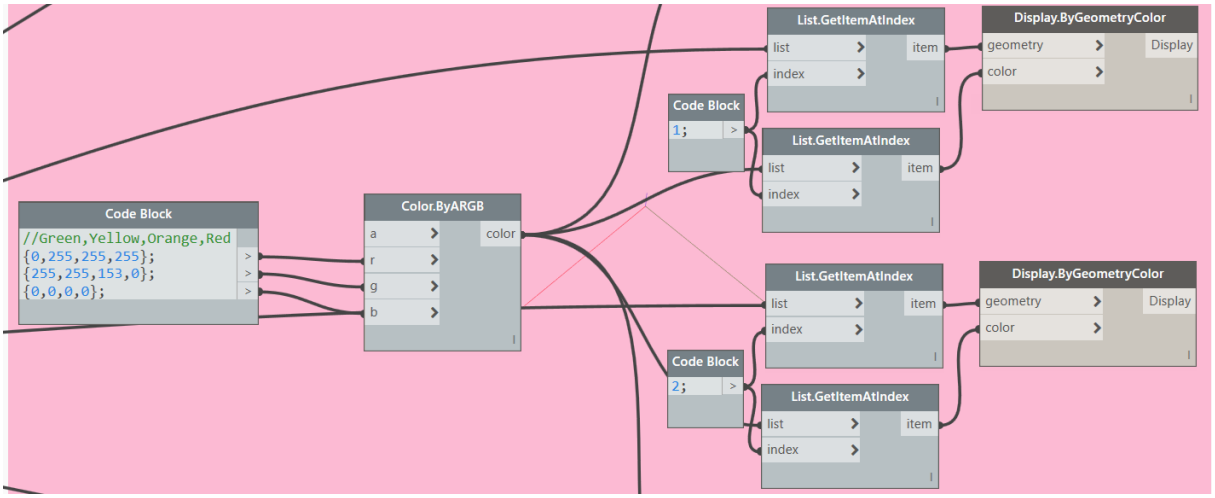


*FIG. 23 - SINGLE CRANE ANALYSIS RESULT IN DYNAMO STUDIO*

## STEP 4 – EVALUATION OF SOLIDS

The Lift Status of each solid is evaluated by using colors. The solid geometry gets a specific color with the *Display.ByGeometryColor* node.



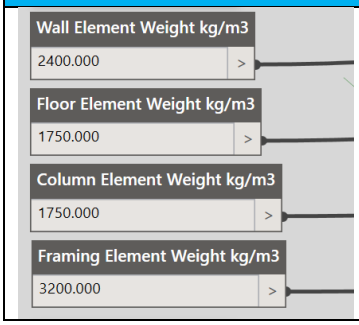
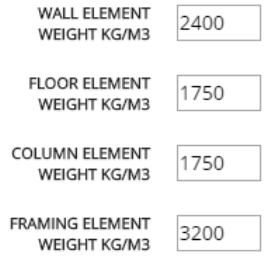
*Crane Analysis with Fractal*

## PREPARE A SCRIPT FOR FRACTAL

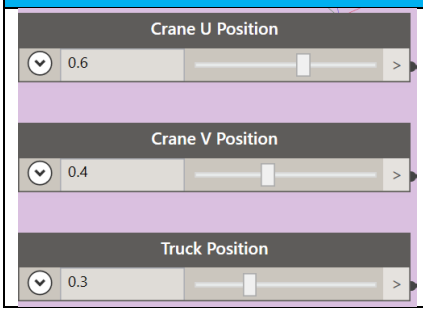
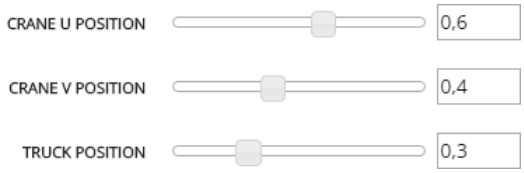
The scripts that are built in the previous steps can now be send to the web (*Dynamo Reach portal)* to be run by Project Fractal.

Before you upload the designs, it's important to keep the next rules in mind:

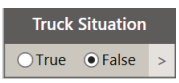
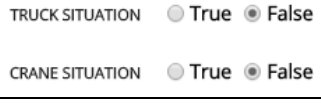
- Code Blocks don't appear as UI objects in Fractal
- Rename the nodes in Dynamo that can appear in the Fractal UI, to have the right parameter reference.
- *Number* and *Integer* nodes appear as input boxes in Fractal, and represent static (constant) values. Rename the nodes in Dynamo by double-clicking the title bar of the node.

| DYNAMO | FRACTAL |
|---|---|
|  |  |

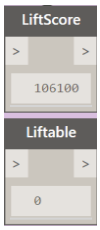
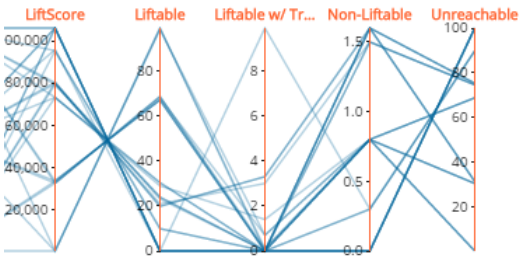
- *Number Sliders* and *Integer Sliders* appear as sliders in Fractal as well, and can be driven by the "Generate" functionality in Fractal to create multiple design options.

| DYNAMO | FRACTAL |
|---|---|
|  |  |

- *Booleans* are represented in Fractal as option boxes and can be driven automatically by Fractal.

| DYNAMO | FRACTAL |
|---|---|
|  |  |

- Use *Watch* nodes in Dynamo Studio to represent the resulting values in Fractal in the parallel coordinate diagram.

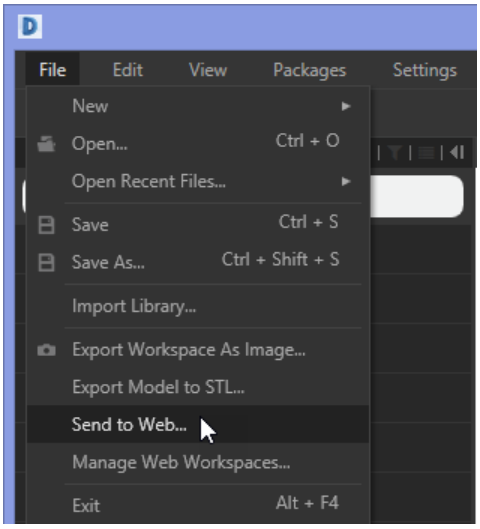
| DYNAMO | FRACTAL |
| --- | --- |
|   |   |

- .SAT and .CSV files can be exported to Fractal

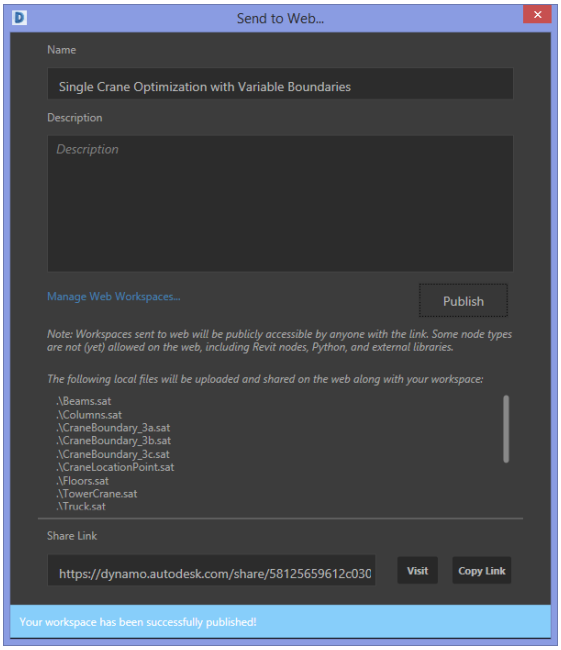
## UPLOAD TO FRACTAL

To send a script to the portal that can be accessed by Fractal (the so called “Dynamo Reach” servers):

1. Open the script in Dynamo Studio
2. Make sure you’re logged in with your Autodesk ID (at the top right)
3. Click on the menu File > Send to Web…



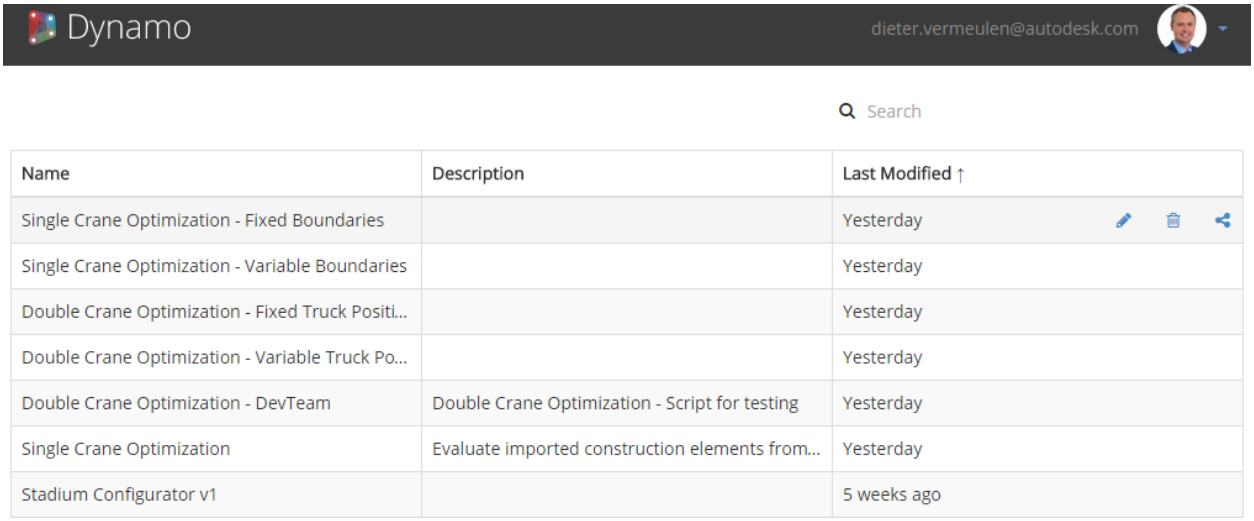

4. Give a name and description for the script and Publish



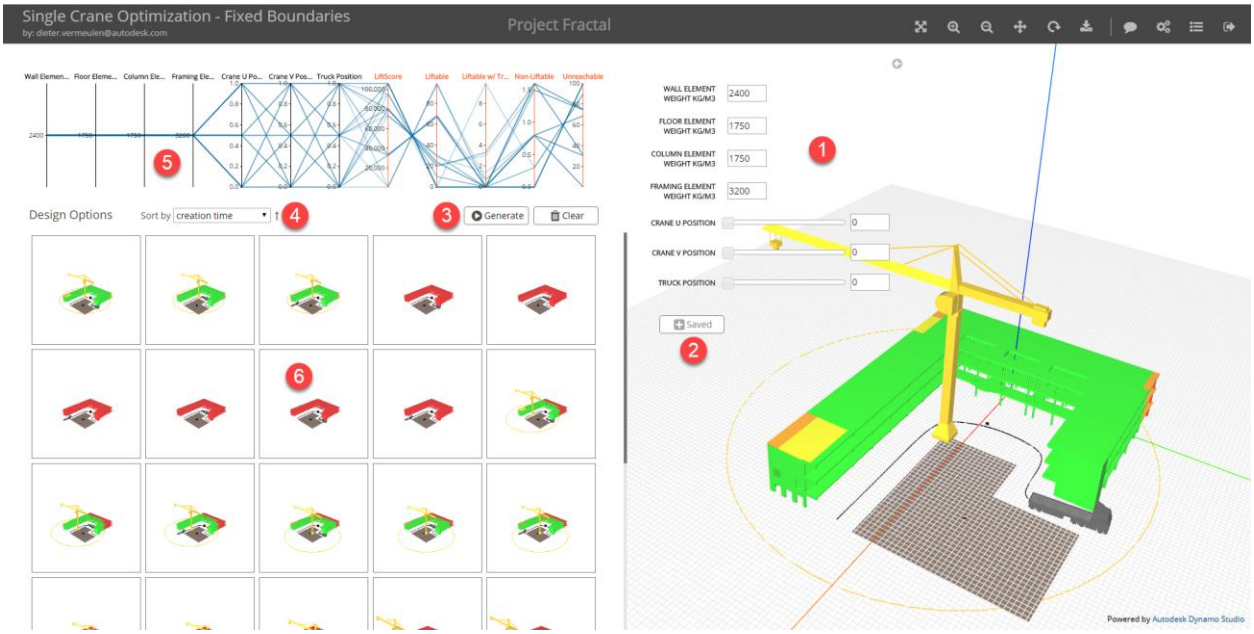

At the bottom you get a link that can be opened directly from here (*Visit*). This will open the *Dynamo Customizer* or you can share the link of the customizer with another person.

5. When you have a Fractal account, go to https://www.fractal.live/manager.html . When you don't have an account, you first need to follow the steps described on this page: https://home.fractal.live/

6. Once logged in, you can open the script from the list like shown below. With the icons at the right of each title, you can edit, delete or share the script.



## GENERATE DESIGN OPTIONS




**(1)** Design variables can be changed manually by means of the input boxes or the sliders

**(2)** You can *Save* a design to the solution list at the left.

**(3)** With *Generate* you let Fractal change the design variables (sliders and booleans) to create multiple design options. The analysis and evaluation that was defined in the Dynamo script is now running in the cloud. You get instant feedback in the results with the parallel coordinates diagram above (5) and the visual results below (6).

**(4)** The design options can be sorted according multiple criteria.

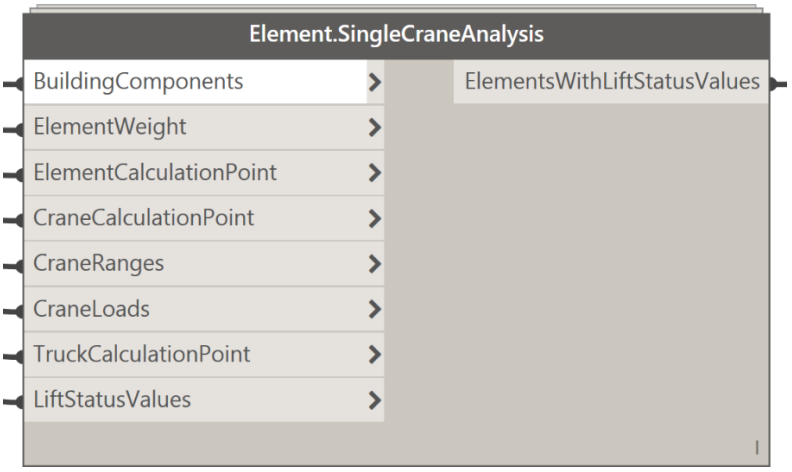
**(5)** The parallel coordinates connect the values for the design variables (blue) and the results (red). You can drag the columns in a different order.

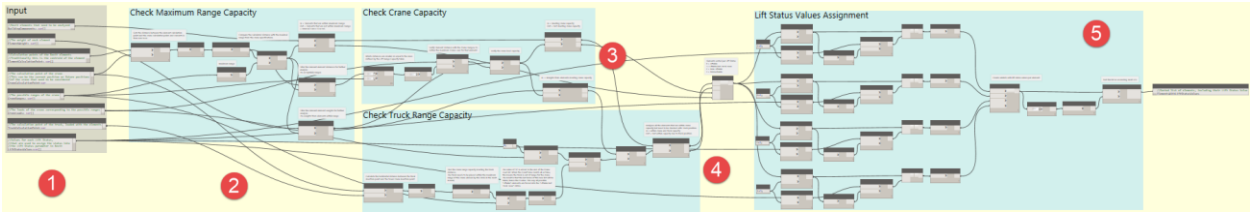
**(6)** The thumbnails give visual feedback on the design option that is created. When hoovering on a thumbnail you can see a tooltip with the values for variables and results. When clicking on the thumbnail, the right pane updates with these values.

## BIM4Struc.CraneAnalysis node list

### *Element.SingleCraneAnalysis*

The base node for all calculation methods is packed in the ***BIM4Struc.CraneAnalysis*** package and is called *Element.SingleCraneAnalysis*. This custom node analyzes the elements to be lifted by a crane, checking a set of requirements and returns an array with the sorted list of elements and their assigned Lift Status Value.




The content of the analysis consists of 5 main parts:

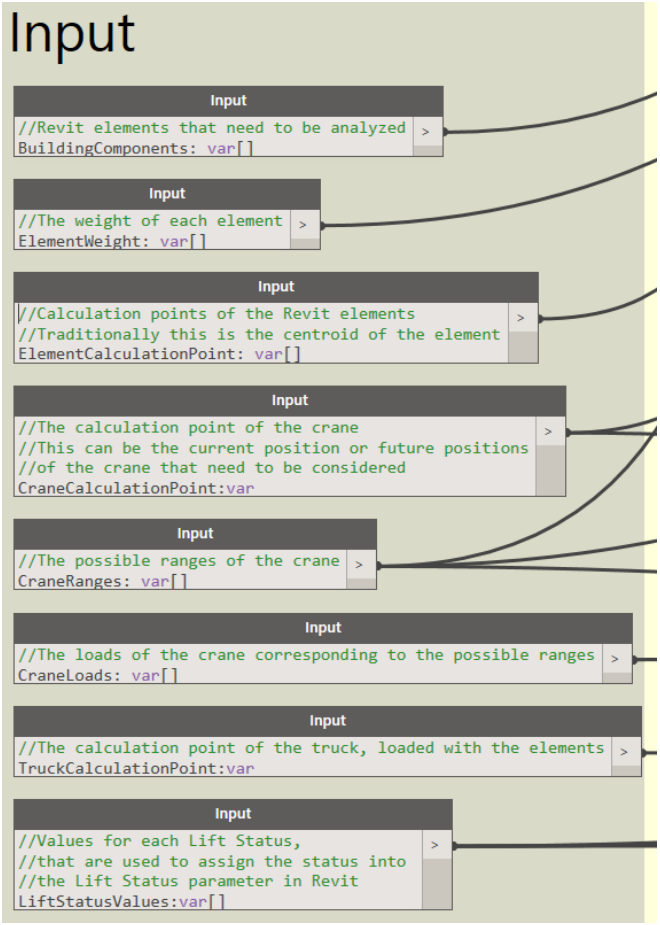
- Step 1 – Input parameters and read analysis set
- Step 2 – Check Maximum Range Capacity of the Crane
- Step 3 – Check Crane Capacity of each element
- Step 4 – Check Truck Range Capacity
- Step 5 – Assign Lift Status Values to the Elements

Each of the steps is explained below.

### STEP 1 – INPUT PARAMETERS

The custom node needs these important input parameters, listed in the image below. The meaning of each input is explained within the code blocks, in green comments.

How the parameters are calculated, is explained in this section.

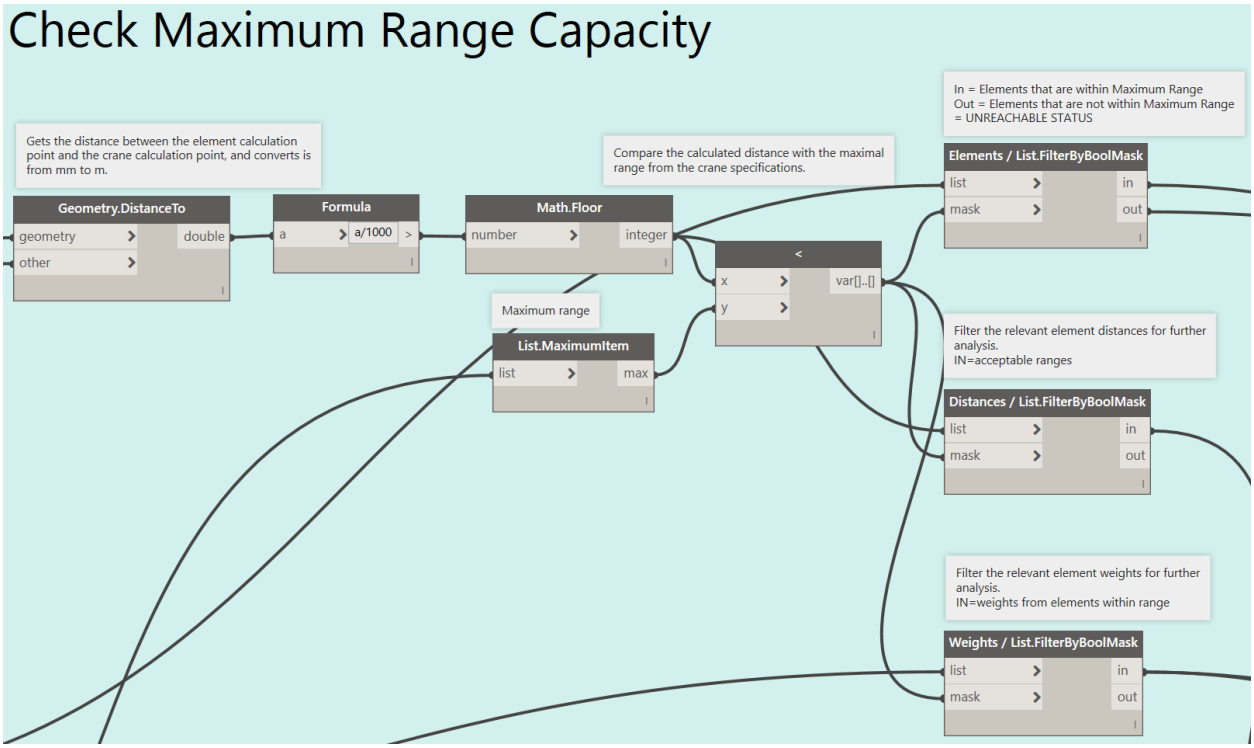



The type indicator behind the ":" in purple has different meanings:

var means a single variable

var[] means that the input is a list of values (with variable datatype)

### STEP 2 – CHECK MAXIMUM RANGE CAPACITY OF THE CRANE




In this part the distance between the element calculation point (centroid) and the crane calculation point (insertion point) is checked against the maximum range of the crane specifications. In this case the Maximum Range is 45 m, defined in the Excel sheet.

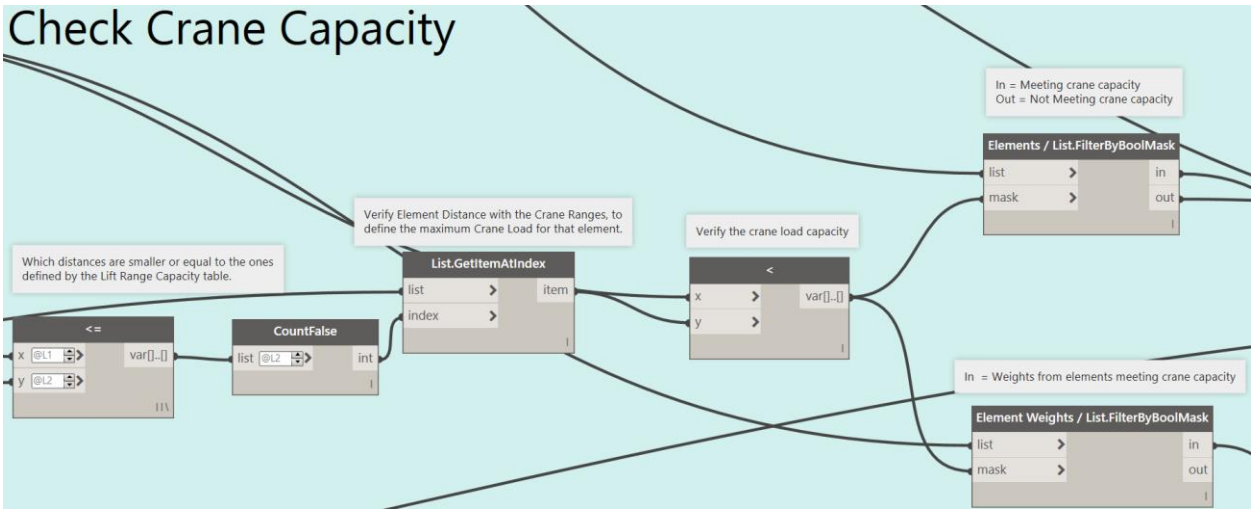
| | A | B |
|---|---|---|
| 1 | **Range** | **Load** |
| 2 | 23 | 16000 |
| 3 | 25 | 14800 |
| 4 | 27 | 13500 |
| 5 | 30 | 11900 |
| 6 | 32 | 11000 |
| 7 | 35 | 9900 |
| 8 | 37 | 9300 |
| 9 | 40 | 8400 |
| 10 | 43 | 7300 |
| 11 | 45 | 5900 |

***Note: the project is expressed in metric units [mm] and the Dynamo script in [m]. You might need to change the formula "a/1000" for your own unit conversion.***

The results are sorted with *List.FilterByBoolMask* nodes for the Elements, Distances and Element Weights, to be used further for the detailed analysis. Only the results from the "in" output ports are used. The output port "out" represents elements that are out of range, hence having the ***Unreachable*** lift status.

### STEP 3 - CHECK CRANE CAPACITY OF EACH ELEMENT

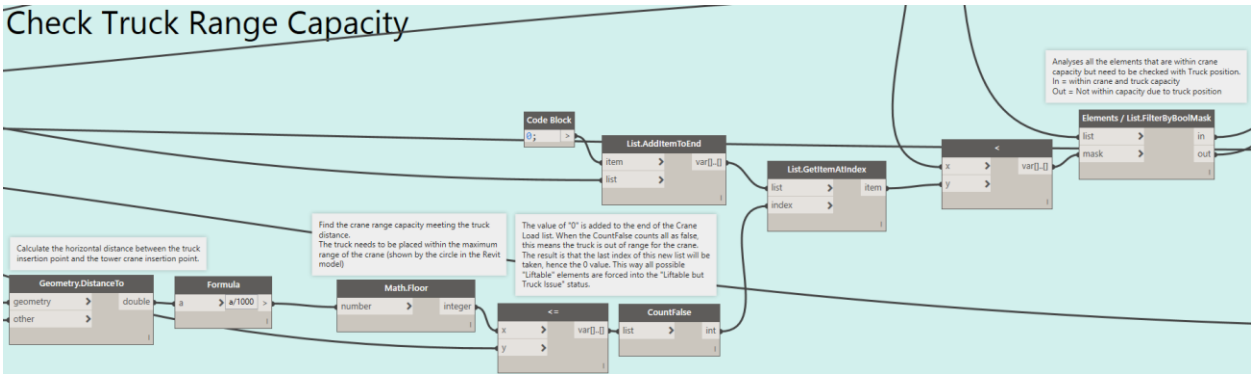
Once the elements that are out-of-range are filtered out, we can work with the remaining elements, their relative distances and their weights to verify if the crane can lift them to the delivery position.




The result of this analysis part is a filtered list with elements that are meeting the crane capacity (possibly liftable) through the “IN” port and elements that have the ***Non-Liftable*** status, through the “OUT” port.

### STEP 4 – CHECK TRUCK RANGE CAPACITY

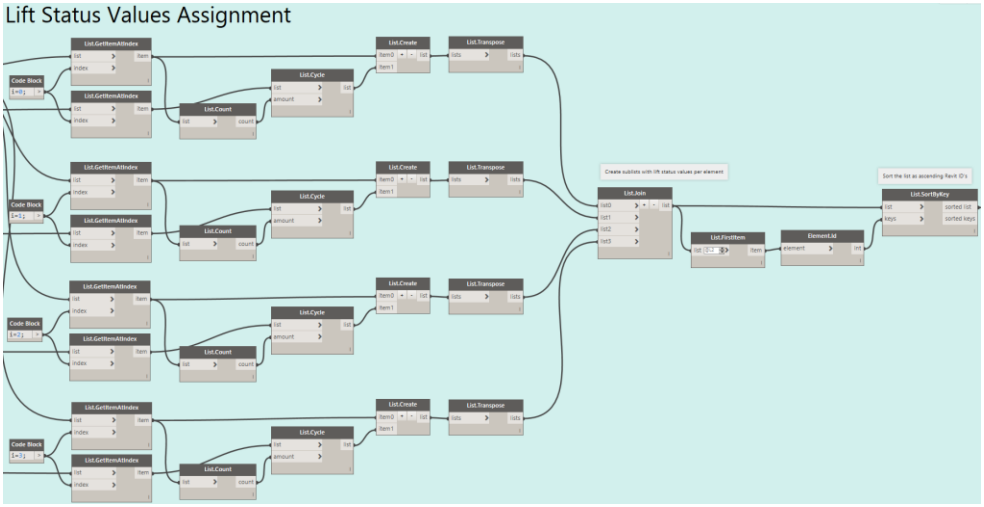
This analysis step is necessary to verify if the elements that can be lifted to the delivery position (possibly liftable) also can be lifted up from the supply point, which is the truck in this case.



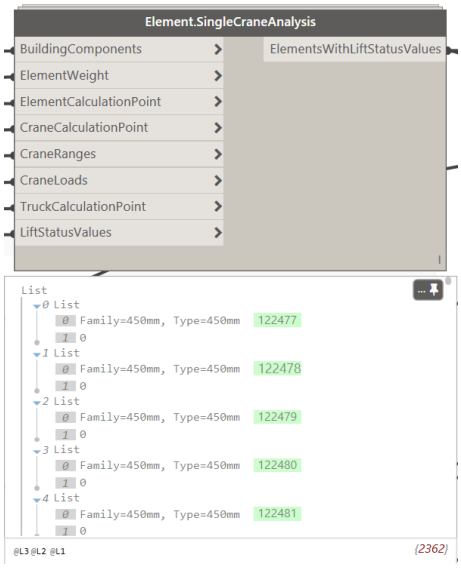

This additional check returns a list of elements that can be lifted from supply point (truck) to delivery point (element calculation point) and have the ***Liftable*** lift status (“IN” port). In case the Truck is out of range for a specific element (due to its combination distance-weight), then the element gets the status ***Liftable but Truck Issue*** (“OUT” port).

## STEP 5 – ASSIGN LIFT STATUS VALUES TO THE ELEMENTS

Once each element is filtered to one of the four lift status groups, a *Lift Status Value* is assigned to it. The values correspond to each of the 4 states. They are then reused further in a dedicated parameter of the element in Revit, as well as for item grouping in Dynamo.

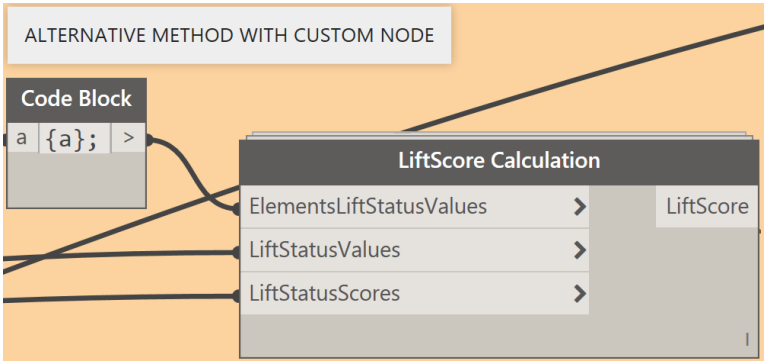



At the start of this part the elements where grouped according to their *Lift Status*. While "dismantling" the groups a corresponding *Lift Status Value* is assigned. For each status, the element and its *Lift Status Value* are put back into a nested list (after transposing), where index 0 = family instance and index 1 = Lift Status Value.
In the middle part all the elements, with their values, are grouped into a complete list, which is then sorted by their Revit ID (in accordance with the inputted format of the *BuildingComponents*, which is a list sorted by their Revit ID as well.)
The output is an array of sorted Revit family instances and their *Lift Status Value.*



### *LiftScore Calculation*

An alternative way to calculate the *Lift Score* is by using the custom node *LiftScore Calculation* from the ***BIM4Struc.CraneAnalysis*** package.




This custom node needs a nested list as input for the *ElementsLiftStatusValues* in order to calculate the total Lift Score for the whole construction. In case not, you will get separate lift scores per element.

The *LiftStatusValues* is a list with a base set for the value of each lift status. These values are used for evalution in Revit only. Read more here.

For each of these values there is a corresponding *LiftStatusScore*. This list also needs to be connected as in input with the custom node. Read more here.

This node is used mainly in the *Parametric Run* and *Genetic Optimization* methods

### *Update & Export Single Crane Results*

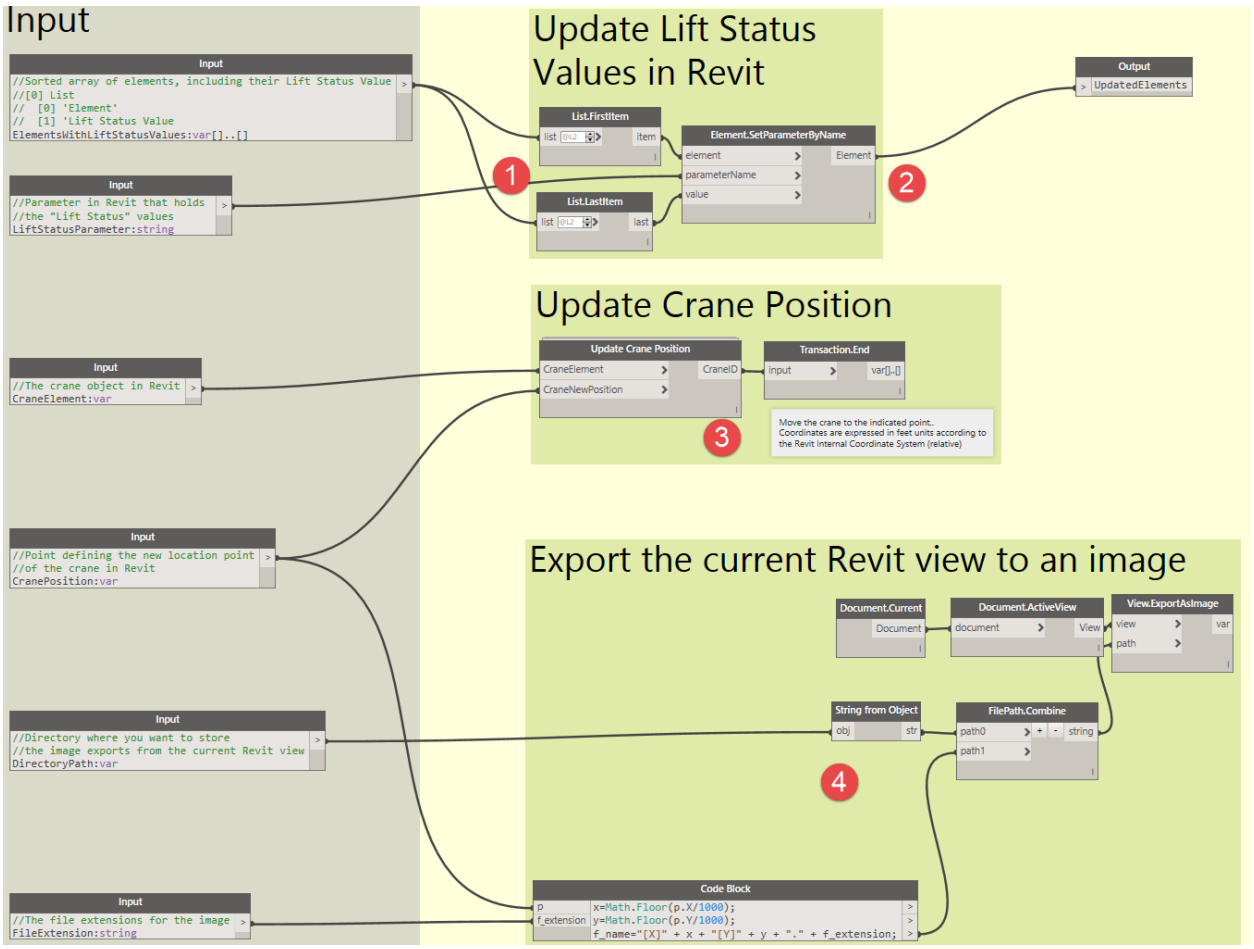



*FIG. 24 - UPDATE & EXPORT SINGLE CRANE RESULTS*

**(1)** Read the *Lift Status* results

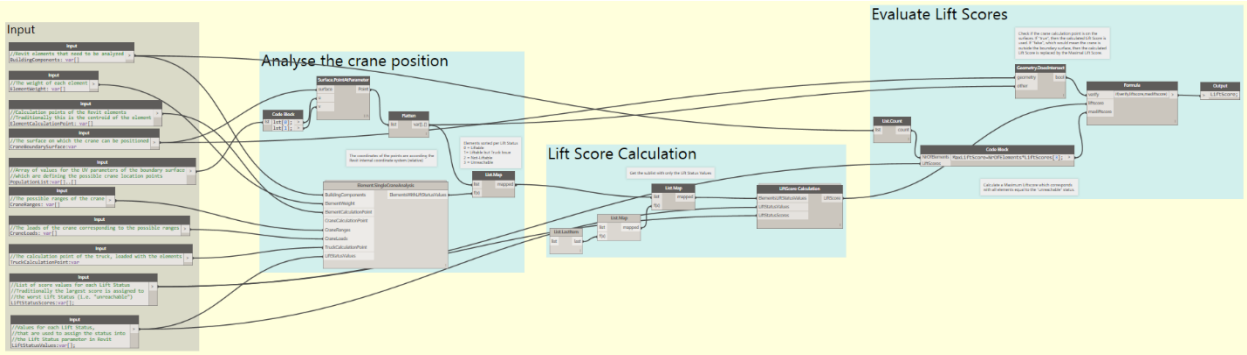
**(2)** Feed the values to the *Lift Status* parameter in Revit, and thus update the 3D view which is applied with View Filters.

**(3)** Update the crane to the analyzed position with the custom node *Update Crane Position*.

**(4)** Export the current Revit view to a “png” image with a naming convention that includes the X and Y parameters of the crane insertion point (according to internal Revit coordinates).

### *Single Crane Analysis Fitness Function*

A fitness function is a custom node that reads the population list, makes the calculations on it and returns a score. In this case this is the Lift Score, which is the single objective here.




## ANALYZE THE CRANE POSITION

As explained in the section Population and objectives, the population list consists of lists of variables and a list of objectives.

The *Code Block* node in the *Analyze the crane position* group, takes only the design variables from this population list:

Index [0] = the variables for the u-division
Index [1] = the variables for the v-division

**Note: Lacing Settings**
*The lacing for the Surface.PointAtParameter needs to be set to 'Longest'. This is opposite to what was done in the Parametric Run. The reason is because the combinations of the points is done by the Population list generation. The $n^{th}$ item in the U-division is combined with the corresponding $n^{th}$ item in the V-division list.*

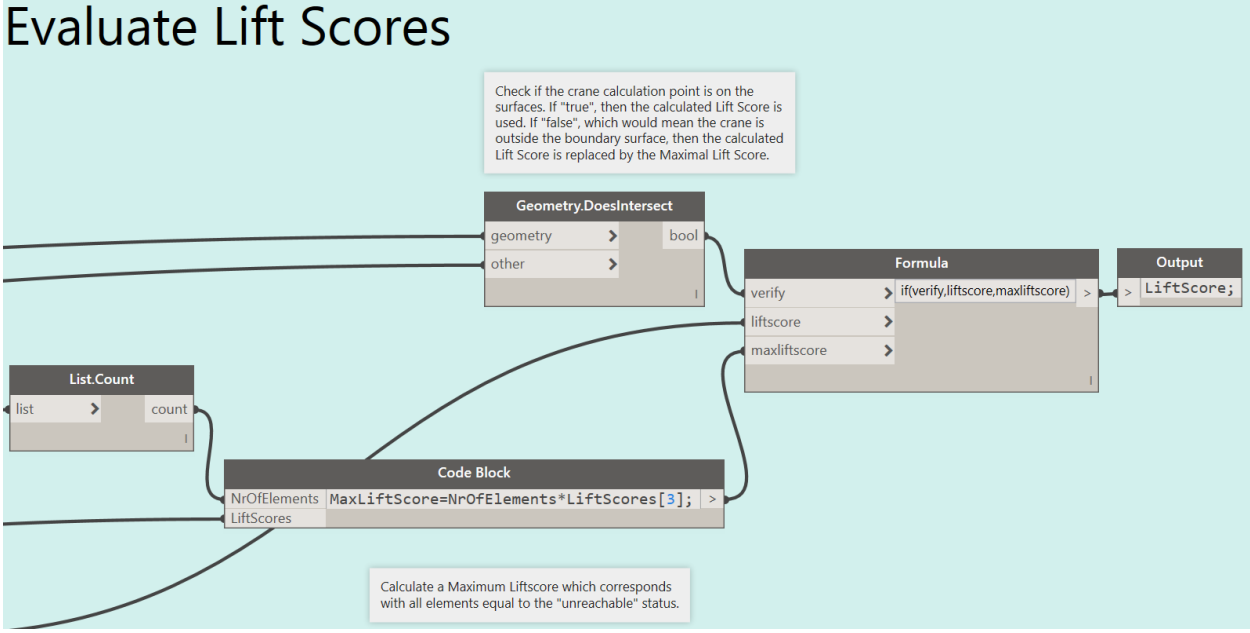
The lists of generated points on the surface are then combined with the *Element.SingleCraneAnalysis* custom node to calculate the Lift Status of each element in the Revit model.

## LIFT SCORE CALCULATION

In this part the Lift Status values are evaluated and a Lift Score for each item in the population list is returned.

Read more about the process on this section.

## EVALUATE LIFT SCORES



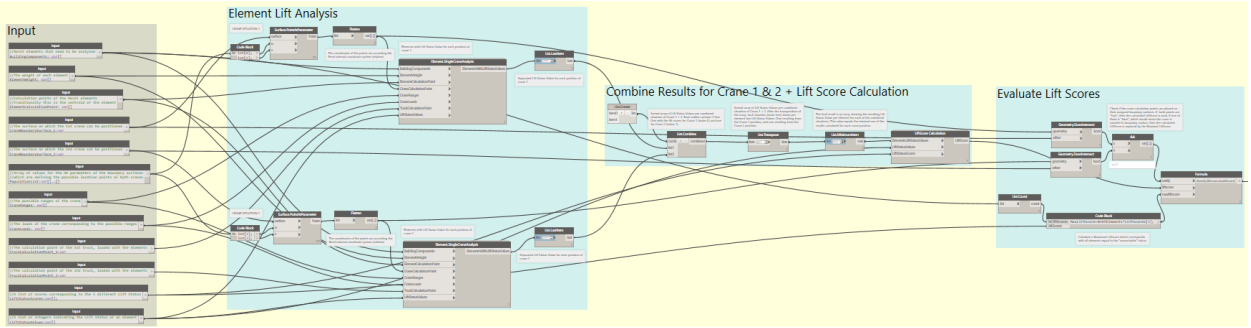

It may occur that, in case of non-rectangular boundary surfaces, there are crane position points generated in the population list, which are not laying on the surface. In a Parametric Run method you can exclude these points from the analysis (which you can read here).

In case of a Genetic Optimization, the number of results for the Lift Score should be identical to the population list. This means we can NOT exclude the generated points from the analysis.

To solve this, this part of the process, detects if the point is on the surface (*Geometry.DoesIntersect)*. In case yes, then the calculated Lift Score is used. In case not, then a maximal Lift Score is used. This max score equals the value for “Unreachable” multiplied with the number of elements. That way the optimization will automatically neglect this solution during the cross-over and mutation process.

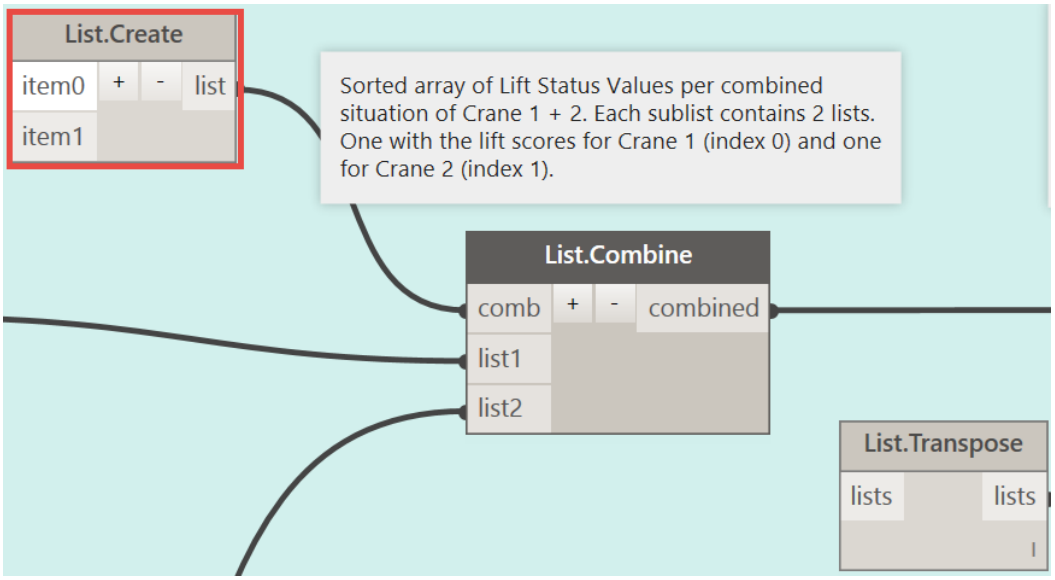
### *Double Crane Analysis Fitness Function*

A fitness function is a custom node that reads the population list, makes the calculations on it and returns a score. In this case this is the Lift Score, which is the single objective here.




The content of this custom node reuses a part of the *Analyze* phase of a Parametric Run.

**Note: Crane Position Combinations**
*The combinations of the crane points in a Parametric Run are done with a List.CartesianProduct node. In a Genetic Optimization process, this is not necessary, as this is arranged by the NSGA algorithm, using cross-over techniques. This means that the results for the ElementsWithStatusValues output are combined with a simple List.Create function.*



## Learning Resources

More learning resources on the products that have been used in this class can be found below:

| | |
|---|---|
| **Dynamo** | http://au.autodesk.com/au-online/classes-on-demand/search?full-text=dynamo<br>http://dynamobim.com/learn/<br>http://dynamoprimer.com/<br>http://dictionary.dynamobim.com<br>http://dynamobim.com/forums/forum/dyn/<br>https://www.lynda.com/Revit-tutorials/Dynamo-Essential-Training/455724-2.html<br>http://www.revitforum.org/dynamo-bim/24005-dynamo-learning-resources.html#post136270 |
| **Project Fractal** | https://home.fractal.live/ |
| **Optimo for Dynamo** | http://bim-sim.org/Optimo/index.html<br>http://github.com/BPOpt/Optimo/wiki/0_-Home<br>AU2015 - Dynam(o)ite Your Design for Engineers |
| **DynaWorks** | https://github.com/Gytaco/DynaWorks |

## Table of Figures