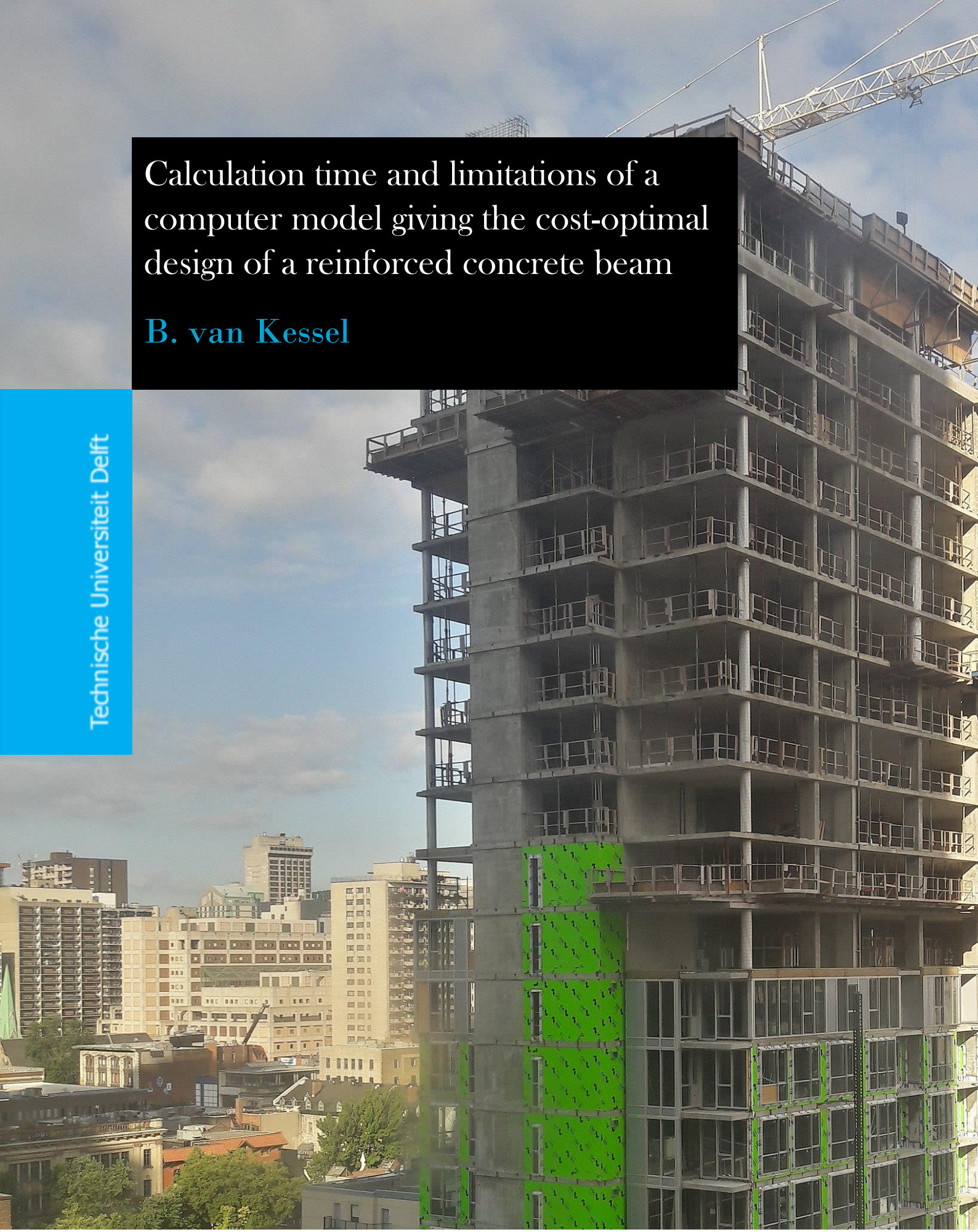


Calculation time and limitations of a computer model giving the cost-optimal design of a reinforced concrete beam

B. van Kessel

Technische Universiteit Delft

TU Delft Delft University of Technology

Challenge the future

Calculation time and limitations of a computer model giving the cost-optimal design of a reinforced concrete beam

By

B. van Kessel

In partial fulfilment of the requirements for the degree of

Bachelor of Science
in Civil Engineering

at the Delft University of Technology,
Faculty of Civil Engineering and Geosciences

June 18, 2019

Student number: 4552318

Project duration: April 22, 2019 – June 18, 2019

Advisors: Dr. ir. P.C.J. Hoogenboom

Dr. ir. drs. C.R. Braam

TU Delft

Abstract

With the increase in computational power and the current development in artificial intelligence, new software applications have become possible. Humans have been replaced by such applications in multiple sectors (Huang, 2018). In the engineering sector many tools are used that also make use of artificial intelligence, but they are not yet able to replace humans. Because of the current development in the field of computer science and electrical engineering this could also happen in the engineering sectors. A software application using artificial intelligence has been built by M. Arafa and M. Alqedra, which was able to make a good estimation of the total costs of a construction project before the design of this structure (Arafa, 2011). In the future it might be possible to develop software that is able to design the structure as well.

In this thesis, the possibilities of such software is investigated. By building a simple computer program that is able to design a rectangular cast-in-situ reinforced concrete beam, the calculation time of such software was researched. The program was designed to find the cost-optimal dimensions of a beam for a given span and load. During the design some problems were found and reported in this thesis. The concrete beam that is designed by the program is tested for the ultimate limit state (ULS). Both the moment capacity and the shear force capacity of the beams are calculated by the program which then performs the strength checks according to the Eurocode. The equations used to calculate the maximum forces and the capacities are also reported in the thesis.

Two different models were built to give the cost-optimal dimensions of a beam. Both models design the beam using 4 different variables: the height of the beam, the width of the beam, the diameter of the reinforcement bars and the number of reinforcement bars. The first model uses multiple *for*-loops to check every combination that could be made with these four variables, where the height and width where given in steps of 10 millimetres and for the diameter the most common reinforcement bar diameters were used (12 mm, 16 mm, 20 mm, 25 mm, 32 mm). The cheapest option that passed the strength checks was then returned by the program. The second model uses an optimisation algorithm from a python package called SciPy. The boundaries and constraints for the function were specified and then inputted in the *minimize* function of this package. The boundaries represent the minimum and maximum value for the four variables and the constraints contain inequalities of the Eurocode requirements. The output given by the function was then rounded up to the closest values that could be used in practice.

The first model turned out to be very fast for short beams but the calculation time increases quadratically with the beam length. For beams around 3 to 4 metre this is approximately 1 second, while a the calculation takes nearly 2 minutes for a 15 metre long beam. The calculation time of the second model depends less on the beam length, but is greatly affected by the convergence of the *minimize* function, which gives very random results for the calculation time. The dimensions of the beam show a clear pattern. The width of the beam is kept as small as possible but is increased so the reinforcement bars fit within this width. To keep the beam's width as small as possible, mainly large diameter bars are used. The height and reinforcement area is increased when the maximum moment increases.

When this computer program would be further developed to design entire structures, the amount of components would increase a lot. The amount of combinations would then increase exponentially with the respect to the components, which would greatly affect the calculation time of the first model. The second method gives results that are far from optimal due to the rounding up of the variables. When the optimisation method would use discrete variables it could be useful in practice. During this research stability of the structure was neglected, which would add a lot of tests to the program which would increase the calculation time. With current computational power this would take too long to be of any use in practice. If it would be further developed it could assist engineers in dimensioning different components, but the layout of the entire structure should be designed by the engineers themselves.

Acknowledgements

This thesis was written as final part of my Bachelor's degree in Civil Engineering at Delft University of Technology. I really enjoyed working on this project, as I could combine my interests in Civil Engineering and Computer Science. It also made me more enthusiastic about programming and made me realize that I want to learn more about it.

During my bachelor Civil Engineering at Delft University, my teachers and peers have taught me a great set of skills in engineering and collaboration, which I'd like to thank them for. I also want to thank some people in particular. First of all, I'd like to thank my first advisor, Pierre Hoogenboom, for helping me find a subject for my bachelor's thesis and giving advice and guidance throughout this period in which I was doing my research. Secondly, I'd like to thank my other advisor, for taking the time to review my work and giving feed-back of the flash report.

Bart van Kessel

Delft, June 2019

Contents

1 – Introduction

1.1 Problem statement

Computers become more powerful every year and a lot of new software is developed to make use of the increasing computational power. There are many software tools created that are useful for engineers. Artificial Intelligence is currently used in many applications and has already replaced human employees in multiple sectors (Huang, 2018). M. Arafa and M. Alqedra have also shown in their research that a good estimation of the total costs of a construction project can be made with the help of artificial intelligence (Arafa, 2011). A software program that designs a construction project using artificial intelligence has not yet been developed, but could be developed in the future.

In 2016, T. Wubs did a research for her bachelor thesis on optimizing the design of a cantilever truss and has shown that it is possible for a computer model to determine the cost-optimal design in six hours (Wubs, 2016). It might also be possible to create modules that give the optimal design for other structural parts (e.g. trusses, beams, joints, slabs, etc.) and their connections. A computer program consisting of a combination of all these modules would then be able to design a structure. To optimise every structural part of a structure and their connections a lot of calculations are needed. To perform these calculations, a lot of time and computational power is required.

1.2 Objective

The aim of this research is to determine the calculation time and limitations of a computer program that gives the cost-optimal dimensions of a rectangular, cast in situ reinforced concrete beam.

The following questions have to be answered to obtain the objective:

1. How can the optimal design of a concrete beam be determined?
    a. Eurocode equations
    b. parameters
    c. validity range
2. Which optimisation algorithms are suitable?

1.3 Approach

By building a computer program that works out the cost-optimal dimensions of a single structural component, one can find potential problems for a program that designs an entire structure and get a better idea of the required calculation time. So a first step would be to create a program that gives the cost-optimal dimensions of a simple structural component, a two support concrete beam. To create such a program, information on concrete design and strength requirements is needed and was collected first. Optimisation algorithms in python were then researched. When this computer program worked, the results were validated. With hand calculations a check was performed, to determine if the program indeed gives a valid and optimal design. A few test cases were made to check the results.

The calculation time of such a computer program does not give a clear view on the total required time to design an entire structure. The calculation has to be performed for all the parts and their connections. For every structural part there are multiple alternatives and therefore there will be many different possible combinations for the design of a structure. Based on the number of combinations and the calculation time of a single element, the estimation of the total required time was made.

1.4 Outline of thesis

Section 2 of this report provides a literature overview of the optimisation algorithms and how they have been applied in the design of concrete structures.

In section 3, the design of a concrete beam is described. The calculations concerning loads and stresses were given here, along with the strength requirements according to the Eurocode. Furthermore the steps that have been taken to design a concrete beam were specified.

The design of the computer models is described in section 4. A clear overview of the steps that the models perform was given in charts and a short description.

In section 5 the results of the research are presented. A conclusion drawn from the results and suggestions are presented in section 6 of this thesis.

Some pictures from additional files were added in the appendices, along with the test cases that were used to verify the results.

2 – Literature review

The design of concrete structures with the use of optimisation algorithms has been researched many times. A lot of different approaches are used in these researches. A program that gave the optimal dimensions for a certain moment capacity was made in an earlier research (Chakrabarty, 1992). This research confirmed that there is a large number of alternative beam dimensions and reinforcement ratios that give the same moment resistance. An alternative approach was recommended where the bending moment in the beam is the input and the unique least-cost beam section is the output.

In a more recent research, the environmental costs of the design of a concrete column was also taken into consideration instead of only the financial costs (de Medeiros & Kripka, 2014). An environmental scoring system for each input of reinforced concrete was used. The results of this research are quite similar to the results from financial optimisation methods, as both methods tend to go for a minimum amount of materials. Although they are quite similar there is one big difference, the model with environmental cost optimisation prefers using non-rectangular beams. This reduces the amount of materials needed but increases the financial costs, which makes it less relevant for financial cost optimisation.

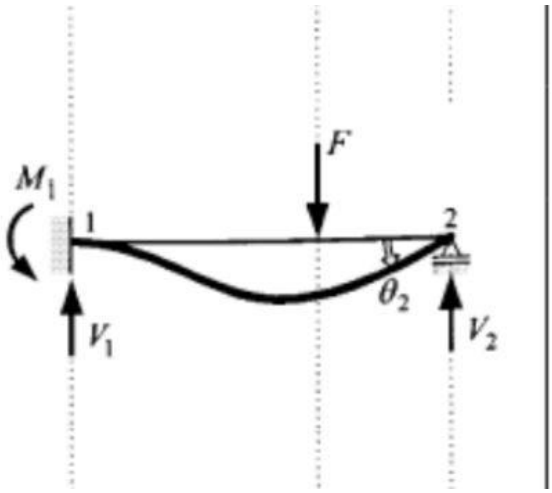
A model giving the optimal design of a three dimensional multi-story concrete structure using meta-heuristic algorithms was presented in a Iranian paper (Kaveh & Behnam, Design optimization of reinforced concrete 3D structures considereing frequency constraints via a charged system search, 2013). This model made use of the charged system search algorithm. The charged search system algorithm is a recently developed optimisation method, that outperforms many evolutionary algorithms (Kaveh & Talatahari, Novel heuristic optimization method: charged system search, 2010). While the required time to perform the design is not stated in the report, it does say that it would be a suitable tool to in practice although the program needs a lot of computational power. Another computer model that was made to research the possibilities of optimisation in reinforced concrete design, made use of another method, namely DMPSO (Esfandiari, Urgessa, Sheikholarefin, & Dehghan Manshadi, 2018). This algorithm is a combination between multi-criteria decision-making (DM) and Particle Swarm Optimization (PSO). The algorithm gave optimal dimensions efficiently for three dimensional reinforced concrete structures.

3 – Concrete beam design

Every construction component has to fulfil certain requirements. These requirements can be found in the Eurocode and a national annex. In this section, the calculations that should be performed to check the strength of a beam are given.

3.1 Loads

When the loads on a beam are known and the type and location of the supports, one can calculate the maximum bending moment and shear force in the beam. For this research only beams on two supports are considered. For many different statically undetermined situations formulas have been made that give the support forces and bending moments inside the beam for a given load. In the figures below the used formulas can be found:



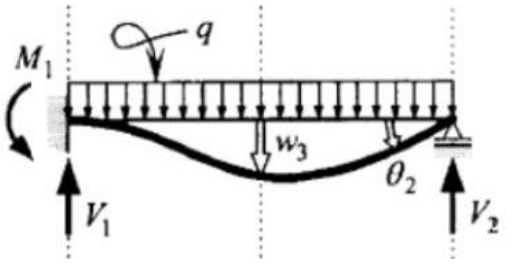
$$M_1 = \frac{Fb(\ell^2 - b^2)}{2\ell^2} = F\ell\left(\frac{a}{\ell} - \frac{3}{2}\frac{a^2}{\ell^2} + \frac{1}{2}\frac{a^3}{\ell^3}\right)$$

$$V_1 = \frac{Fb(3\ell^2 - b^2)}{2\ell^3} = F\left(1 - \frac{3}{2}\frac{a^2}{\ell^2} + \frac{1}{2}\frac{a^3}{\ell^3}\right)$$

$$V_2 = \frac{Fa^2(3\ell - a)}{2\ell^3} = F\left(\frac{3}{2}\frac{a^2}{\ell^2} - \frac{1}{2}\frac{a^3}{\ell^3}\right)$$

$$\theta_2 = \frac{Fa^2 b}{4EI\ell} = \frac{F\ell^2}{4EI}\left(\frac{a^2}{\ell^2} - \frac{a^3}{\ell^3}\right)$$

Figure 1: Equations for statically indeterminate beam with a fixed and a hinge support and a point load (van Rotterdam, 2005).



$$\theta_2 = \frac{1}{48}\frac{q\ell^3}{EI}; \quad w_3 = \frac{1}{192}\frac{q\ell^4}{EI}$$

$$M_1 = \frac{1}{8}q\ell^2; \quad V_1 = \frac{5}{8}q\ell; \quad V_2 = \frac{3}{8}q\ell$$

Figure 2: Equations for statically indeterminate beam with a fixed and hinge support and a distributed load (van Rotterdam, 2005).

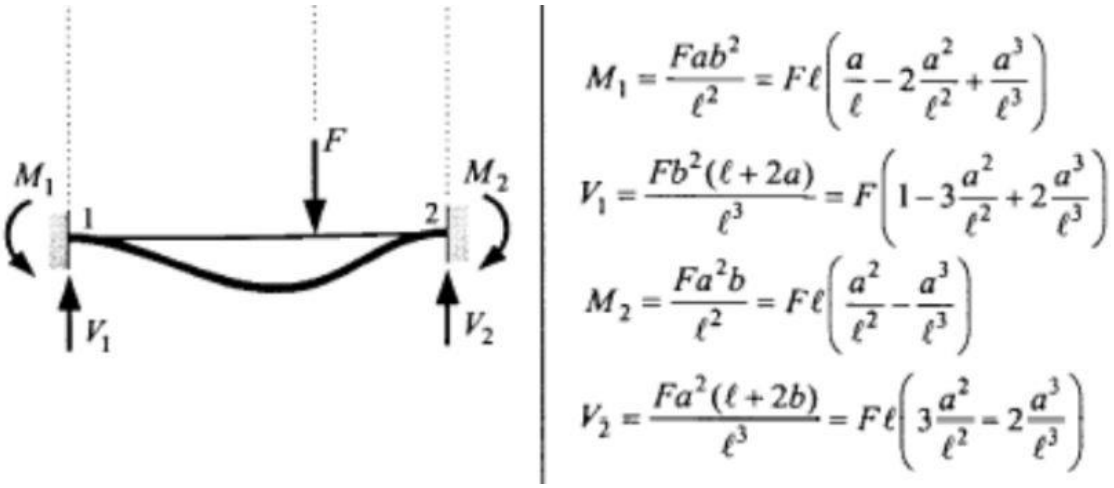


Figure 3: Equations for statically indeterminate beam with two fixed supports and a point load (van Rotterdam, 2005).

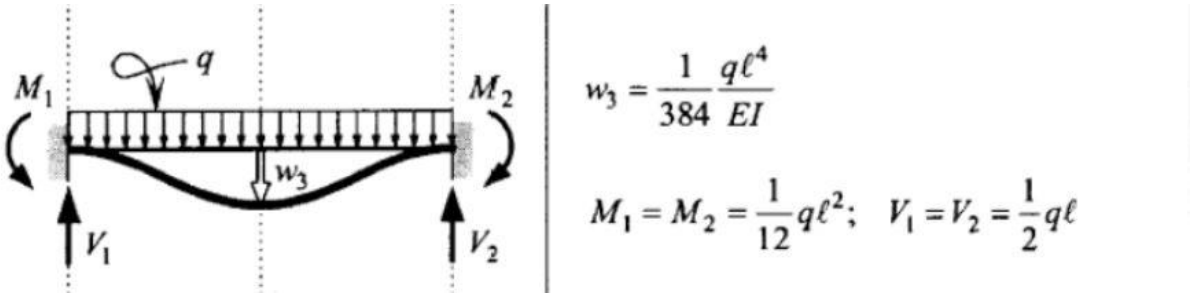


Figure 4: Equations for statically indeterminate beam with two fixed supports and a distributed load (van Rotterdam, 2005).

For cases with a statically determinate beam, the support forces are calculated by solving the equilibrium equations for the external forces (Hartsuijker, 1999). This gives the following formulas for the support forces:

$$A_v = \frac{F \cdot b}{a+b}; B_v = \frac{F \cdot a}{a+b} \qquad (3.1)$$

$$A_v = B_v = \frac{1}{2} \cdot q \cdot l \qquad (3.2)$$

Where *a* is the distance from the left support to the location of the point load and *b* is the distance from the right support. Distributed loads that do not cover the entire length of the beam can't be inputted in the program. After performing the calculations the line of the shear force can be created and with the shear force line, the moment line can be created.

Loads are always multiplied by a safety factor depending on the type of load and the type of calculation. As the model only checks the beam for the ultimate limit state (ULS), the factors for permanent and variable loads are 1.2 and 1.5 respectively (Wagemans, Soons, & Raaij, 2014). The factors on the loads that are inputted in the model should already be applied, as the program can't distinguish permanent and variable loads. The factor on permanent loads for the self-weight of the beam is included in the model and has a value of 1.2.

3.2 Moment capacity

The maximum bending moment that a concrete beam can resist depends on the dimensions of the beam and the strength of the reinforcement steel and concrete. It is assumed that the reinforcement steel is yielding at the maximum bending moment and that the concrete has cracked in the tensile area and does not take any tensile forces (Hordijk & Lagendijk, B2 Buiging, 2017). The program does not take into account a normal force in the beam and therefore no 2nd and 3rd order moments as well. The force in the reinforcement steel can be calculated with the following formula:

$$N_s = A_s \cdot f_{yd} \tag{3.3}$$

Where A_s is the total area of the reinforcement bars and f_{yd} the design yield stress of the steel. The total compressive force in the concrete N_c has to be equal to N_s in the opposite direction for a horizontal force equilibrium. The moment in the beam can be calculated by multiplying the force by the arm *z*. The arm depends on the height of the beam, the concrete cover, the diameters of the reinforcement and the height of the concrete compression zone. The height of the compression zone can be determined with the following formula:

$$N_s = N_c = f_{cd} \cdot b \cdot \alpha \cdot x_u \tag{3.4}$$

Where f_{cd} is the design concrete strength, b is the width of the beam, x_u is the concrete compression zone height and α is the shape factor. The value for α depends on the concrete strength and is 0.75 for a concrete strength lower than or equal to C50/60. As all values except for x_u are known it is possible to determine the height of the concrete compression zone. The arm *z* can then also be calculated:

$$x_u = \frac{A_s \cdot f_{yd}}{f_{cd} \cdot b \cdot \alpha} \tag{3.5}$$

$$z = h - c - \emptyset_{sh} - 0.5 \cdot \emptyset_{long} - \beta \cdot x_u \tag{3.6}$$

When the lever arm and the normal force in the concrete and reinforcement steel are known the moment capacity of the beam can be calculated by multiplying these values. This calculation is shown in the following equation:

$$M_{Rd,max} = N_s \cdot z \tag{3.7}$$

Some additional checks have to be performed during this calculation. There are restrictions to the amount of longitudinal reinforcement used and the height of the concrete compression zone. Eurocode suggests a minimum reinforcement ratio according to the following equation (EN:1992-1-1 9.2, 2004):

$$\frac{A_s}{A_c} = 0.26 \cdot \frac{f_{ctm}}{f_{yk}} \tag{3.8}$$

For the maximum reinforcement a ratio of 4% is suggested, these values may differ in any national annex but for the design of our computer model the values in the Eurocode were used. The maximum amount of steel is also limited by the maximium concrete compression zone x_u. The maximum ratio between effective height of the beam and the height of the concrete compression zone is given by the following equation (Hordijk & Lagendijk, B2 Buiging, 2017):

$$\frac{x_u}{d} \leq \frac{\varepsilon_{cu} \cdot 10^6}{\varepsilon_{cu} \cdot 10^6 + 7 \cdot f_{yd}} \qquad (3.9)$$

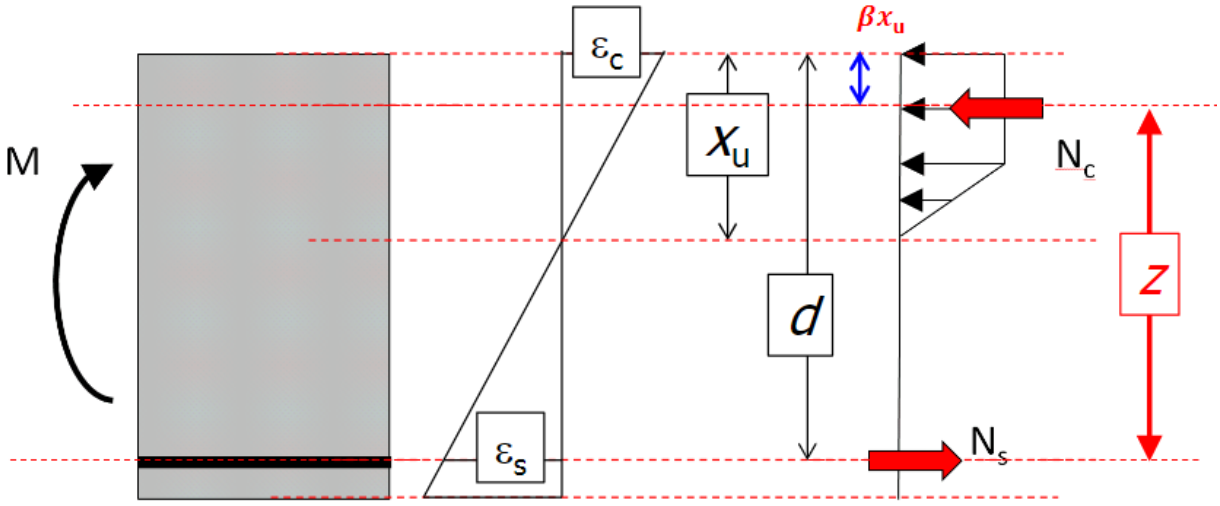


Figure 5: Schematisation of concrete and steel force and the internal lever arm (Hordijk & Lagendijk, B2 Buiging, 2017).

3.3 Shear resistance

Besides moment capacity, the strength of the beam is also checked for the shear force. The shear force requirements are also given by the Eurocode. There are equations for a beam with and without shear reinforcement. If a beam would have sufficient shear resistance without shear reinforcement, minimal reinforcement is applied, with a spacing of 300 millimetres. The strength calculations for both situations are described below.

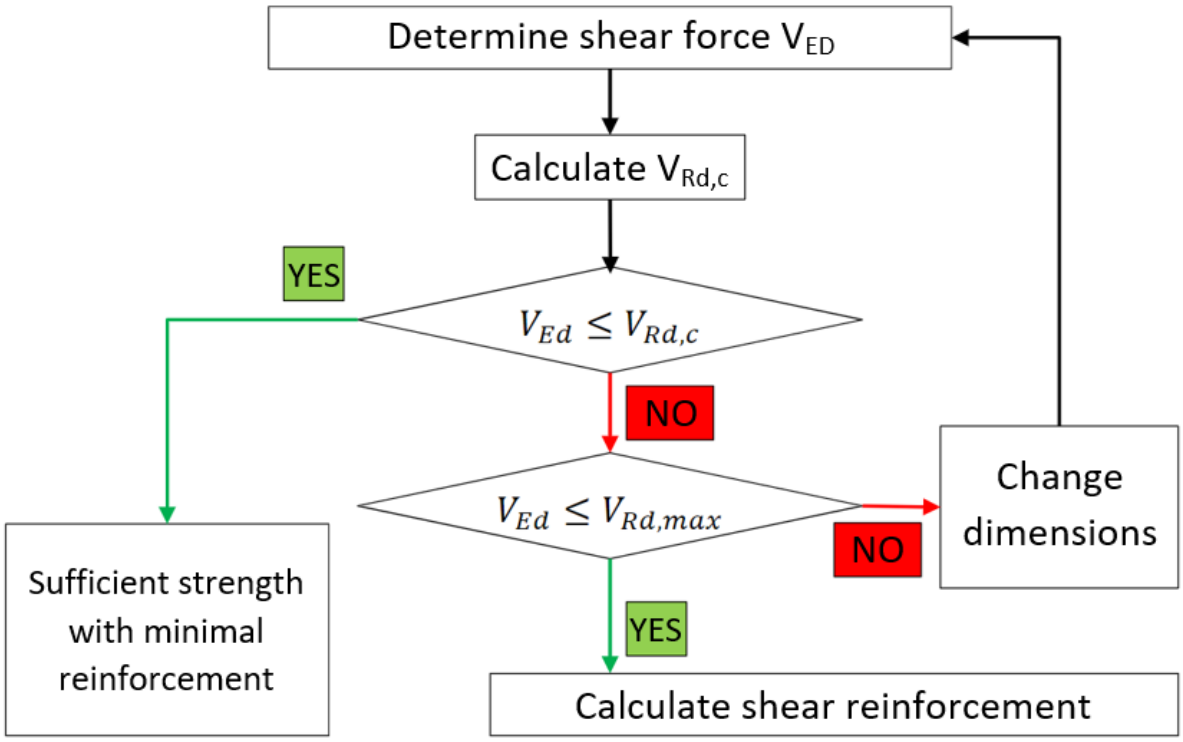


Figure 6: Design process of shear reinforcement (Hordijk & Lagendijk, CTB2220 Beton & Staalconstructies, 2017).

3.3.1 Shear resistance without shear reinforcement

The shear resistance without shear reinforcement depends on multiple factors. The strength is determined based on the characteristic concrete strength f_{ck}, the effective height d of the beam and the reinforcement ratio in longitudinal direction using the effective height. Eurocode gives two equations to calculate the strength, a design value and a minimum value (EN:1992-1-1 6.2.2). Both these equations use a variable k. The following equations are used to determine the shear resistance:

$$k = \min\left(1 + \sqrt{\frac{200}{d}}; 2.0\right); d \text{ in } mm \qquad (3.10)$$

$$v_{Rd,c} = 0.12 \cdot k \cdot (100 \cdot \rho_l \cdot f_{ck})^{1/3} \qquad (3.11)$$

$$v_{min} = 0.035 \cdot k^{3/2} \cdot {f_{ck}}^{1/2} \qquad (3.12)$$

The stresses are then multiplied by the width of the beam and the effective height to obtain the total shear resistance of the cross-sectional area. If the design value is lower than the minimum value, the minimum value will be used.

3.3.2 Shear resistance with shear reinforcement

For the shear resistance with reinforcement Eurocode gives again two different formulas, one for the design shear resistance and one for the maximum shear resistance.

$$V_{Rd,s} = \frac{A_{sw}}{s} \cdot z \cdot f_{ywd} \cdot \cot\theta \quad (3.13)$$

$$V_{Rd,max} = \frac{\alpha_{cw} \cdot b_w \cdot z \cdot v_1 \cdot f_{cd}}{\cot\theta + \tan\theta} \quad (3.14)$$

The values for α_{cw} and v_1 are also given in the Eurocode and are 1 and 0.6 respectively if no prestress is used and concrete strength is below 60 MPa. f_{ywd} represents the yield stress of the shear reinforcement. The spacing s is determined by the program based on the other variables. The angle between the longitudinal reinforcement and the concrete struts is called θ. The value of θ must satisfy the following condition:

$$21.8° \leq \theta \leq 45°$$

There is also a minimum requirement for the ratio between the cross-sectional area of the shear reinforcement and a product of the width of the beam and the spacing (EN:1992-1-1 9.2.2). This equation is presented below:

$$\rho_{w,min} = \frac{A_s}{b \cdot s} = \frac{0.08 \cdot \sqrt{f_{ck}}}{f_{yk}} \quad (3.15)$$

The total shear resistance can be calculated by multiplying the design shear stress with the effective height and the beam width.

3.4 Material properties

For the design of the beam, many different types of steel and concrete can be used. There are many different concrete strength classes, but only strength classes up to C50 can be used by the program as some equations change for concrete above this strength class. For the reinforcement steel, only B500B is used, as this is the most common type of reinforcement steel. For both the materials there are also different material factors that should be used to calculate the design strength. An overview of the concrete classes and the material factors can be found below.

Table 1: Material factors according to EN:1992-1-1 2.4.

Design situations	γ_C for concrete	γ_S for reinforcing steel	γ_S for prestressing steel
Persistent & Transient	1,5	1,15	1,15
Accidental	1,2	1,0	1,0

Table 2: Concrete classes from table 3.1 in EN:1992-1-1 3.1.

Strength classes for concrete															Analytical relation / Explanation
f_{ck} (MPa)	12	16	20	25	30	35	40	45	50	55	60	70	80	90	
$f_{ck,cube}$ (MPa)	15	20	25	30	37	45	50	55	60	67	75	85	95	105	2.8
f_{cm} (MPa)	20	24	28	33	38	43	48	53	58	63	68	78	88	98	$f_{cm} = f_{ck}+8$(MPa)
f_{ctm} (MPa)	1,6	1,9	2,2	2,6	2,9	3,2	3,5	3,8	4,1	4,2	4,4	4,6	4,8	5,0	$f_{ctm}=0,30\times f_{ck}^{(2/3)} \leq$C50/60 $f_{ctm}=2,12\cdot\ln(1+(f_{cm}/10))$ > C50/60
$f_{ctk,0,05}$ (MPa)	1,1	1,3	1,5	1,8	2,0	2,2	2,5	2,7	2,9	3,0	3,1	3,2	3,4	3,5	$f_{ctk,0,05} = 0,7\times f_{ctm}$ 5% fractile
$f_{ctk,0,95}$ (MPa)	2,0	2,5	2,9	3,3	3,8	4,2	4,6	4,9	5,3	5,5	5,7	6,0	6,3	6,6	$f_{ctk,0,95} = 1,3\times f_{ctm}$ 95% fractile
E_{cm} (GPa)	27	29	30	31	33	34	35	36	37	38	39	41	42	44	$E_{cm} = 22[(f_{cm})/10]^{0,3}$ (f_{cm} in MPa)
ε_{c1} (‰)	1,8	1,9	2,0	2,1	2,2	2,25	2,3	2,4	2,45	2,5	2,6	2,7	2,8	2,8	see Figure 3.2 $\varepsilon_{C1}(\varepsilon_{C1}(^0/_{00})) = 0,7\ f_{cm}^{0,31} \leq 2,8$
ε_{cu1} (‰)	3,5									3,2	3,0	2,8	2,8	2,8	see Figure 3.2 for $f_{ck} \geq 50$ Mpa $\varepsilon_{cu1}(^0/_{00})=2,8+27[(98-f_{cm})/100]^4$
ε_{c2} (‰)	2,0									2,2	2,3	2,4	2, 5	2,6	see Figure 3.3 for $f_{ck} \geq 50$ Mpa $\varepsilon_{c2}(^0/_{00})=2,0+0,085(f_{ck}-50)^{0,53}$
ε_{cu2} (‰)	3,5									3,1	2,9	2,7	2,6	2,6	see Figure 3.3 for $f_{ck} \geq 50$ Mpa $\varepsilon_{cu2}(^0/_{00})=2,6+35[(90-f_{ck})/100]^4$
n	2,0									1,75	1,6	1,45	1,4	1,4	for $f_{ck} \geq 50$ Mpa $n=1,4+23,4[(90-f_{ck})/100]^4$
ε_{c3} (‰)	1,75									1,8	1,9	2,0	2,2	2,3	see Figure 3.4 for $f_{ck} \geq 50$ Mpa $\varepsilon_{c3}(^0/_{00})=1,75+0,55[(f_{ck}-50)/40]$
ε_{cu3} (‰)	3,5									3,1	2,9	2,7	2,6	2,6	see Figure 3.4 for $f_{ck} \geq 50$ Mpa $\varepsilon_{cu3}(^0/_{00})=2,6+35[(90-f_{ck})/100]^4$

3.4.1Costs

To find the cost-optimal design of a beam, it is important to know how the costs of that beam can be calculated. The cost of the construction of a beam does not only depend on the cost of the materials, but also on the costs of labour. For this reason shear reinforcement is more expensive than longitudinal reinforcement. An estimation of the price for both types of reinforcements was given by Dr. ir. drs. C.R. Braam, a professor in concrete structures at Delft University of Technology. The values that were used for the reinforcement costs are 2 €/kg and 3 €/kg for longitudinal and shear reinforcement respectively. The cost of concrete depends on the strength class of the concrete and is a lot cheaper than steel. The price of concrete that was used is 130 €/m^3 for concrete class C30. For concrete classes higher or lower than C30 the price changes with 10 €/m^3 per concrete class. The total amount of longitudinal reinforcement, shear reinforcement and concrete is calculated by the program and is multiplied by the values given above to give the total cost of the beam.

3.5 Additional information

Besides the above mentioned checks, there are also some other things that need to be taken into account when designing a concrete beam. In the Eurocode there is also a rule for the minimum concrete cover, the distance between the reinforcement steel and the surface area of the beam. This value depends on the lifetime of the structure and the environment that it is placed in. The value should be taken according to the table below. The exposure classes can be found in the other table.

Table 3: Concrete cover according to table 4.4N from EN:1992-1-1.

Environmental Requirement for $c_{min,dur}$ (mm)							
Structural Class	**Exposure Class according to Table 4.1**						
	X0	XC1	XC2 / XC3	XC4	XD1 / XS1	XD2 / XS2	XD3 / XS3
S1	10	10	10	15	20	25	30
S2	10	10	15	20	25	30	35
S3	10	10	20	25	30	35	40
S4	10	15	25	30	35	40	45
S5	15	20	30	35	40	45	50
S6	20	25	35	40	45	50	55

Table 4: Exposure classes related to environmental conditions according to table 4.1 from EN:1992-1-1.

Class designation	Description of the environment	Informative examples where exposure classes may occur
1 No risk of corrosion or attack		
X0	For concrete without reinforcement or embedded metal: all exposures except where there is freeze/thaw, abrasion or chemical attack For concrete with reinforcement or embedded metal: very dry	Concrete inside buildings with very low air humidity
2 Corrosion induced by carbonation		
XC1	Dry or permanently wet	Concrete inside buildings with low air humidity Concrete permanently submerged in water
XC2	Wet, rarely dry	Concrete surfaces subject to long-term water contact Many foundations
XC3	Moderate humidity	Concrete inside buildings with moderate or high air humidity External concrete sheltered from rain
XC4	Cyclic wet and dry	Concrete surfaces subject to water contact, not within exposure class XC2
3 Corrosion induced by chlorides		
XD1	Moderate humidity	Concrete surfaces exposed to airborne chlorides
XD2	Wet, rarely dry	Swimming pools Concrete components exposed to industrial waters containing chlorides
XD3	Cyclic wet and dry	Parts of bridges exposed to spray containing chlorides Pavements Car park slabs
4 Corrosion induced by chlorides from sea water		
XS1	Exposed to airborne salt but not in direct contact with sea water	Structures near to or on the coast
XS2	Permanently submerged	Parts of marine structures
XS3	Tidal, splash and spray zones	Parts of marine structures
5. Freeze/Thaw Attack		
XF1	Moderate water saturation, without de-icing agent	Vertical concrete surfaces exposed to rain and freezing
XF2	Moderate water saturation, with de-icing agent	Vertical concrete surfaces of road structures exposed to freezing and airborne de-icing agents
XF3	High water saturation, without de-icing agents	Horizontal concrete surfaces exposed to rain and freezing
XF4	High water saturation with de-icing agents or sea water	Road and bridge decks exposed to de-icing agents Concrete surfaces exposed to direct spray containing de-icing agents and freezing Splash zone of marine structures exposed to freezing
6. Chemical attack		
XA1	Slightly aggressive chemical environment according to EN 206-1, Table 2	Natural soils and ground water
XA2	Moderately aggressive chemical environment according to EN 206-1, Table 2	Natural soils and ground water
XA3	Highly aggressive chemical environment according to EN 206-1, Table 2	Natural soils and ground water

Eurocode also gives a minimum value for the clear distance between reinforcement bars. The minimum distance is given by the highest value from the following equations:

$$\max(k_1 \cdot d,\ \ k_2 \cdot d_g,\ \ 20) \qquad (3.16)$$

Where k_1 and k_2 are constants that are given in the National Annex. The recommended values according to EN:1992-1-1 are 1 mm and 5 mm respectively. The maximum grain size in the concrete mixture is given by d_g. This minimum distance is required so that the concrete can be casted and compacted between the bars and it is necessary for an adequate bond.

4 – Computer model design

4.1 Design of computer model 1 (without optimisation algorithm)

The first model that was designed, was a simple computer model that does not make use of optimisation algorithms. The program consists of many checks that are performed within the main loop. Four variables in the dimensions of the beams are used, the other values are user input and remain constant during the design of the beam. Input arrays are created for these four variables: beam height, beam width, diameter of reinforcement bars and the number of bars. For the beam height and width the following rule-of-thumbs were used:

$$h = {}^{1}/_{10} - {}^{1}/_{15}\ \ l \qquad\qquad b = {}^{1}/_{3} - {}^{1}/_{2}\ \ h$$

A small factor is used to make sure that the optimal dimensions are inside the range. For the diameter of the reinforcement steel, a standard array was made with most commonly used diameters, namely: 12 mm, 16 mm, 20 mm, 25 mm and 32 mm. For the number of bars, 2 is used as a minimum to have a proper connection to the shear reinforcement in the corners of the beam. It is then increased with a step of 1 until the maximum number of cables that would fit within the width of the beam.

By using four for loops, all the combinations of this input is used to check if the beam fulfils all the requirements and what the price of the beam would be. The checks that are performed are described in section 3 of this report. When all the checks pass, the cost is compared to the cheapest combination that passed the tests and the dimensions are saved in an array that is printed at the end of the program. The schematisation on the next page gives an overview of what the main loop looks like.

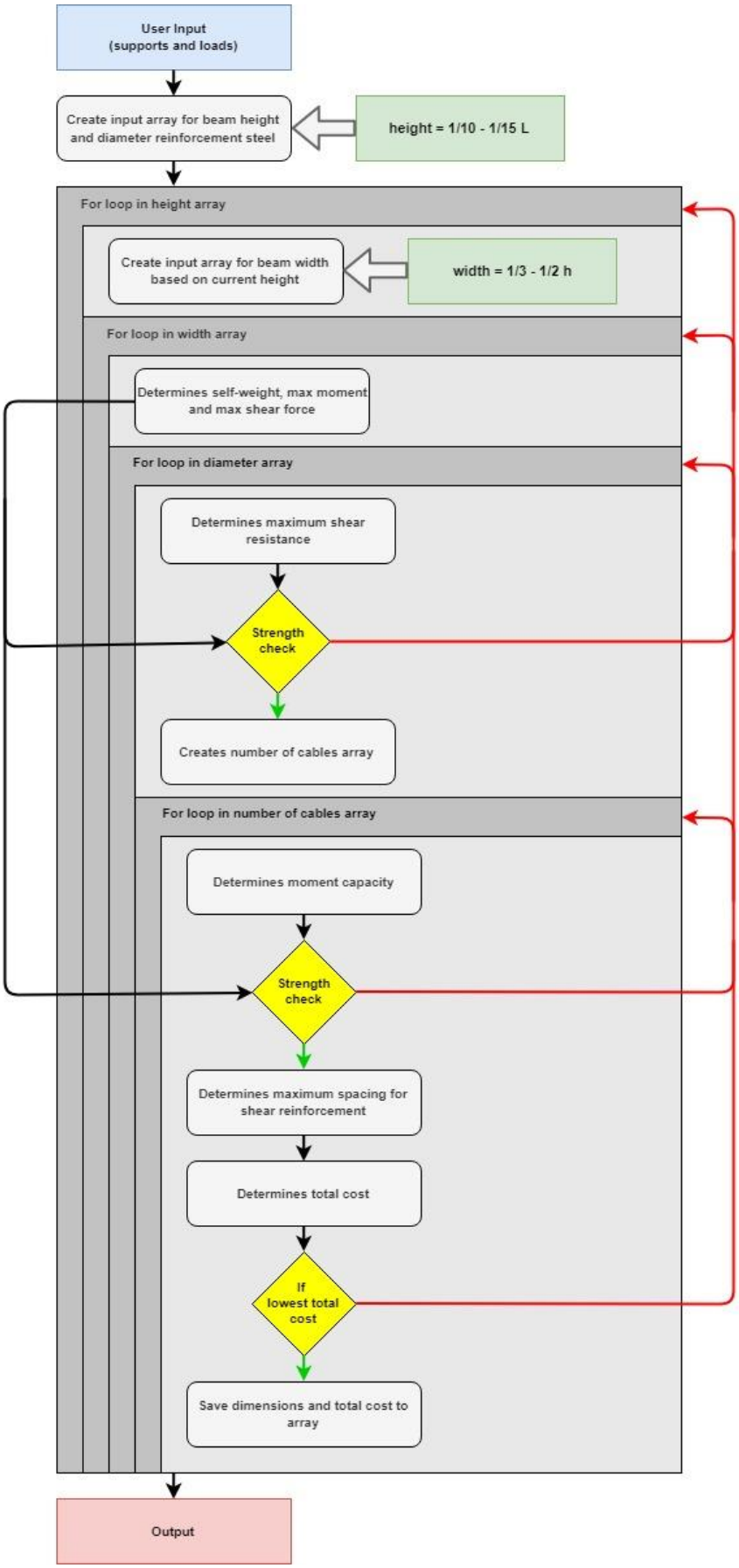


Figure 7: Schematisation of the computer model.

4.2 Design of computer model 2 (with optimisation algorithm)

Another model was created to see if the calculation time could be reduced. This model makes use of an optimisation package from python called *scipy.optimize*. Inside this package are multiple tools, one of these tools is *minimize* (SciPy.org, 2019). This tool is used to find the minimum value of a function with different possible optimisation methods. For the design of the beam there are many known boundaries for the optimal dimensions of the beam, given by the rule-of-thumbs and available reinforcement cable diameters. Some optimisation methods of the *minimize* tool can take into account these boundaries to reduce the calculation time and only give valid answers.

Other checks that have to be performed, the moment capacity, shear force resistance and the minimum and maximum reinforcement ratio are also inputted in the model. These checks were put in the program by defining them as constraints. Some of the methods cannot deal with both constraints and boundaries so there are only two possible methods left, namely: *SLSQP* and *trust-constr* (SciPy.org, 2019). The model makes use of the *SLSQP* method, which is the default method when boundaries and constraints are given. The *SLSQP* method replaces the objective function with the quadratic approximation and the constraints functions are replaced by linear approximations (NEOS, 2019).

The main function for this model is also very different than the main function for model 1. The checks are now defined as constraints and are therefore not included in the main function. The main function now only determines the maximum shear force and the required spacing to resist this shear force and it returns the cost of the beam. The *minimize* function searches for the lowest cost within the boundaries and constraints but with continuous variables. The output contains very specific numbers with a precision that can't be achieved in real life. Concrete beams are dimensioned with a precision of several millimetres, reinforcement cables are only available at certain diameters and the number of cables can only be a whole number. The output of the function is therefore rounded up to more realistic values which also increases the costs of the beam.

An initial guess is also needed for the *minimize* function to start the calculation. This is again done by using the rule-of-thumbs. For the diameter and number of cables the minimum is taken so that it will fit within the width of the beam in all cases. When all the data is inputted, the program returns an array with the optimal dimensions and the cost of the beam. The schematisation below gives an overview of the model.

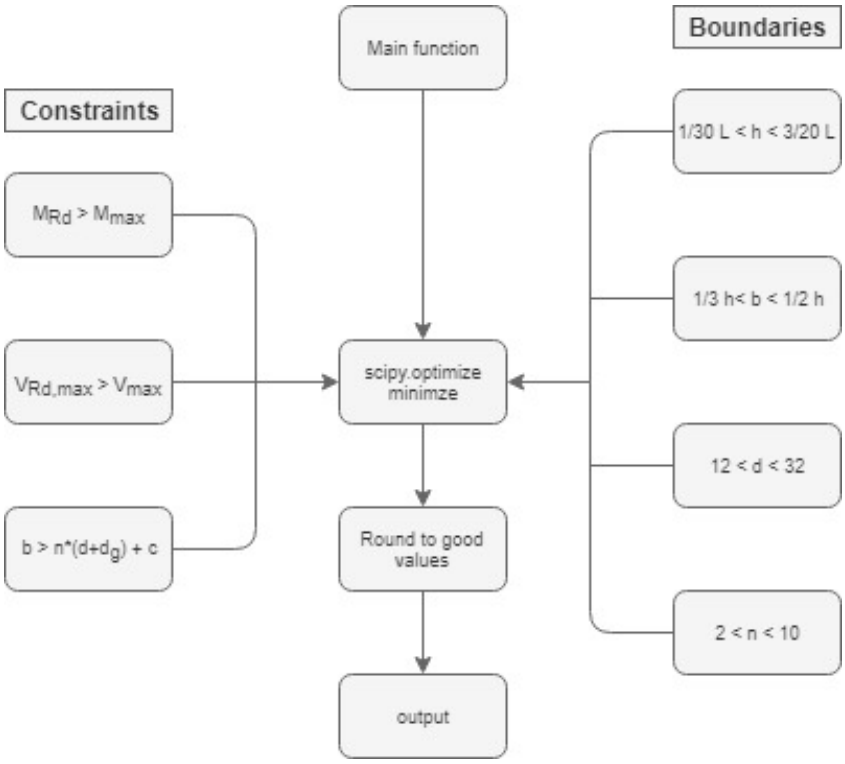


Figure 8: Schematisation of the computer model with the SLSQP optimisation method.

The values that the *minimize* function returns are changed by a function. This function rounds the height and width up to the closest multiple of 10 mm. Then it goes through two for-loops to check which combination of diameter and number of bars is closest to the area that was found with the *minimize* function. It also checks if the bars fit within the width of the beam, if this is not the case then the width is increased by 10 mm and it runs through the for-loops again.

5 – Results

Both models have been run several times to check the output. The checks that were performed according to the Eurocode were also done by hand and MatrixFrame. There were 3 different test cases which were ran by both models and the check by hand and the output data can be found in the appendices. The output of the models met the requirements by the Eurocode so the program works properly. Some other tests were performed to check the calculation speed and the prices of the beams. During all of the tests the following variables remained constant at the following values:

- Concrete cover 35 [mm]
- Yield stress steel f_{yk} 500 [N/mm^2]
- Concrete compressive strength f_{ck} 30 [N/mm^2]
- Shear reinforcement diameter $\emptyset_{sh}$ 8 [mm]
- Strut angle θ 30 [deg]
- Concrete cost 130 [€/m^3]
- Longitudinal reinforcement cost 2 [€/kg]
- Shear reinforcement cost 3 [€/kg]

5.1 Calculation time

The calculation speed of the model without optimisation algorithm mainly depends on the length of the beam. The program runs through multiple arrays to check all the combinations. The height and width array are created based on the length of the beam and they get longer as the beam gets longer. This explains the increase in calculation time for the model without optimisation algorithm. The calculation time of the model with the optimisation algorithm is less dependent on the beam length. As the beam gets longer, the boundaries of the *minimize* function go farther apart so there are more possible values for which the cost could be minimal, but it doesn't perform a calculation for every combination. The graph below shows the calculation time against the beam length. The load is kept constant at 10 kN/m during these tests. With the *curve_fit* function from *scipy.optimize* a line was created that is quite accurate for the model without optimisation algorithm (SciPy, 2019). The line shows a quadratic relation between the time and length.

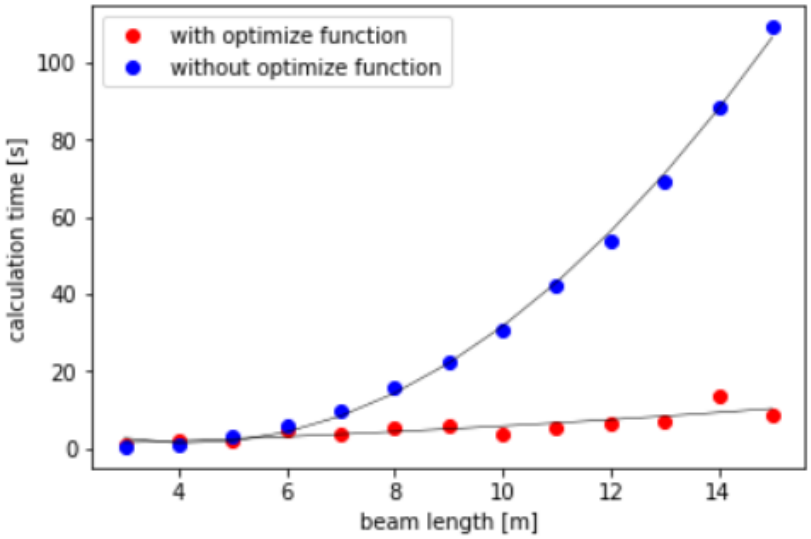


Figure 9: Graph of the calculation time against the beam length for a load of 10 kN/m.

From the graph it is clear that the model with the optimisation algorithm is a lot faster for longer beams while for shorter beams the needed time is almost the same. The speed of the program without optimisation algorithm could be improved by changing the formulas that define the boundaries for the height and width. This would make the arrays that are used to check every possibility shorter and therefore less possibilities are checked.

The models have also been ran with different loads and the calculation time of every calculation has been written down. For smaller beam lengths these values are very small so the difference in calculation time can be neglected. For longer beams you can see this difference in calculation time in the graph below. For every length of the beam, the time for a load of 10 kN/m is the shortest. The expected result would be that the time required for higher loads is shorter, as more strength checks will fail and then the cost calculation and possibly some other strength checks will be skipped. The difference in calculation time could also be caused by a difference in available computational power of the computer during the tests.

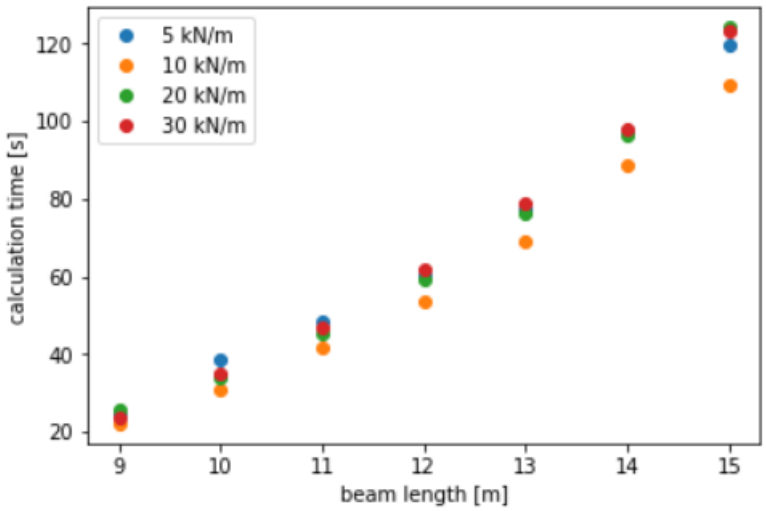


Figure 10: Calculation time for model 1 with different loads.

When looking at the same results for model 2, there is no clear pattern in the calculation time. The results are scattered and there is no load for which the required calculation time is clearly lower. There is also an unexpected outlier at a length of 7 meters and a load of 5 kN/m, the program was therefore ran multiple times with this input with the same result every time. The difference in calculation time could be caused by a wrong initial guess or because the algorithm has trouble finding the optimal value.

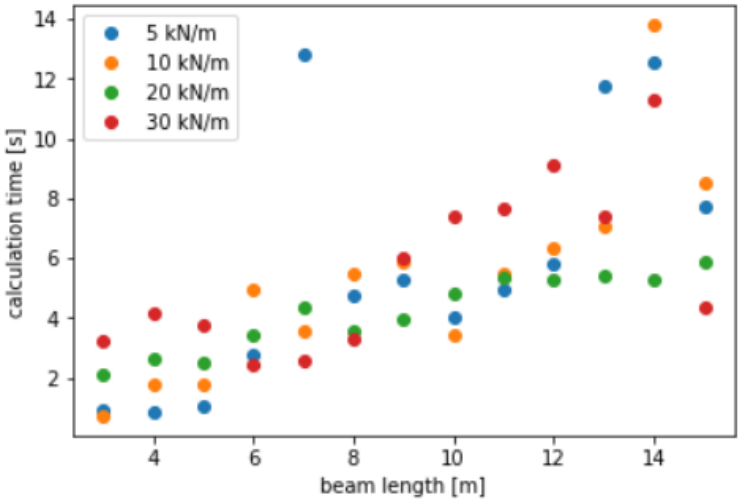


Figure 11: Calculation time for model 2 with different loads.

5.2 Costs

Besides the calculation time it is interesting to look at the dimensions that both models return and the costs of those beams. To compare both models, the program was ran with 4 different loads, namely: 5 kN/m, 10 kN/m, 20 kN/m and 30 kN/m. For every load the length was changed multiple times between 3 and 15 meter.

In the graph below you can see the average cost of a beam that was designed by model 2 with the optimisation algorithm before the values were rounded up. In this graph you can see that the increase in costs rises with the beam length. This graph is best described by a quadratic function. When the beam gets longer more material is needed and the cross-sectional area of the beam and reinforcement bars increases due to the increase in the maximum moment. In the left graph you can see the costs for the beams that were given by model and model 2 divided by the optimal cost. For model 1 you can see that the output is farther from the optimal cost as the beam gets longer. The output from model 2 is in most cases more expensive than the output from model 1 and doesn't show a clear pattern. This seems logical as the optimal design can lie close to a valid design, but when the area of the reinforcement bars doesn't lie close to a realistic value the beam can be over dimensioned which is reflected in the costs as well.

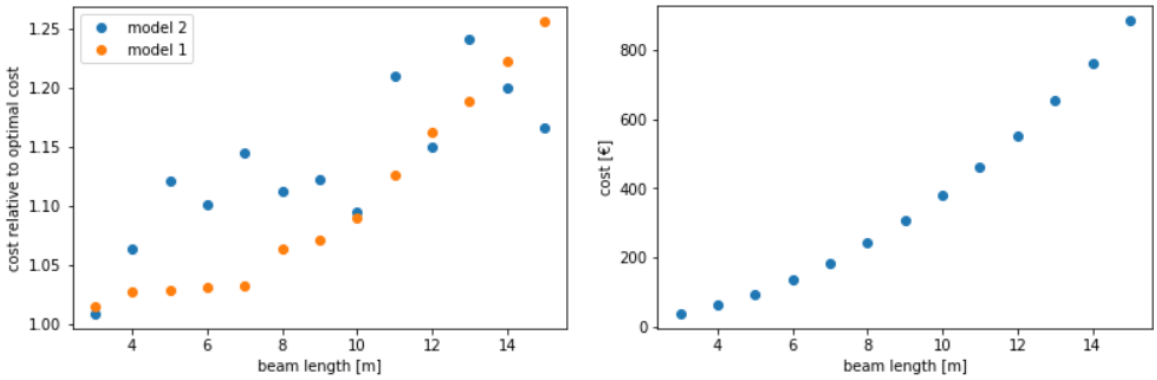


Figure 12: Average cost of a beam relative to the optimal cost (left) and average optimal cost (right).

5.3 Dimensions

Another interesting thing to look at is the dimensions that both models give. In the tables below you can find the results from both models when using a load of 20 kN/m. You can clearly see here that the width is kept very small as this contributes little to the moment capacity compared to the reinforcement area and the height of the beam. The width is increased when the required reinforcement area increases, so that the bars fit within this width. For model 1 the width is also increased when the beam gets longer, this is because the array of the width that is used to find the optimal dimensions depends on the height of the beam and varies between a 1/3 and 1/2 of the beam height. It also becomes clear that the height of the beams designed by model 2 continuously increases, while the beams of model 1 decrease when the area of the reinforcement bars increases.

Table 5: Results of the tests with a load of 20 kN/m.

L =	3	4	5	6	7	8	9	10	11	12	13	14	15
q = 20 kN/m													
Model 1													
time	0,314	1,182	3,157	6,493	10,774	16,885	25,655	33,888	45,131	59,424	76,394	96,547	124,194
height	220	330	350	470	440	550	700	570	670	810	680	780	910
width	200	200	200	200	210	210	230	220	220	270	290	290	300
diameter	16	16	20	20	25	25	25	32	32	32	32	32	32
number of bars	2	2	2	2	2	2	2	2	2	2	3	3	3
spacing	300	300	300	300	300	300	300	293,2	300	300	251,3	262,1	274,4
cost	42,73	71,58	111,32	157,6	219,75	285,06	383,2	467,88	554,27	733,23	927,89	1058,7	1239,88
Model 2													
time	2,091	2,607	2,473	3,456	4,388	3,56	3,987	4,841	5,374	5,269	5,422	5,278	5,906
height	240	300	370	430	500	560	630	700	760	830	900	960	1030
width	200	200	200	200	200	200	200	200	200	200	250	250	250
diameter	12	16	16	25	25	25	32	32	32	32	32	32	32
number of bars	3	3	3	2	2	2	2	2	2	2	3	3	3
spacing	300	300	300	300	300	300	300	300	300	300	300	300	300
cost	41,8	80,16	112,76	182,76	229,39	279,42	423,71	493,91	566,98	646,28	975,51	1086,18	1205,27
cost continuous var.	40,911	70,21	108,83	155,05	209,94	275,18	348,2	430,54	524,33	626,09	737,82	862,14	994,63

The results of the same tests with different loads are similar. The width stays very small while the height increases a lot. The diameter of the bars increases first from low values and when it is at 32 mm it will stay very high so that the width can be kept minimal. The results for the other loads can be found in the appendices.

6 – Conclusion

The aim of this research was to determine the calculation time and limitations of a computer model that gives the cost-optimal dimensions of a concrete beam. The calculation time has been recorded and was presented in section 5 of this report. Limitations that were found during this research were reported here.

Now that the calculation time of the model designing a single beam is known we can finally say something about how long it would take to design an entire structure using such software. The model is able to design a beam with a maximum of 2 minutes time, making it useful for engineers as this could be done many times on a single working day. However, when the number of components increases, the number of combinations increases exponentially. So using the same method as used in model 1, checking every combination and saving the cheapest option, would require a lot of time. Assuming that the calculation time increases linearly with the number of combinations, then model 1 would not be suitable for the design of a structure with many components. A structure of 5 components can have a calculation time of up to 100^5 s, which is too long to be of any use in practice.

Both models only take into account the strength requirements of the beam. The stability of all the components and the entire structure is not checked. A lot of extra formulas are needed to check the stability of a structure. Though the checks should be possible to program, this would add a lot of extra checks that have to be performed. Also designing and checking the connection between different components is not implemented in the models. With all these extra checks that should be performed the calculation time would again increase substantially, and the current method would not be of any use in practice.

While these models or potential further developed models would not be able to replace engineers and design a whole structure independently. They could be used by engineers during the design process to dimension separate components, while the overall layout of the structure is designed by the engineers themselves.

6.1 Discussion

Some choices that were made during the design of the computer models could be considered as wrong choices. The costs of the beams designed by model 2 were often a lot higher than the prices for the beams designed by model 1. This was mainly caused by the way that the values are rounded up. The optimal value that was found, with the continuous variables, was just rounded up to the closest valid values. This could be more efficient if the program would make small arrays around these values to find the optimal solution, as the design is now often over dimensioned. The choice for optimisation with discrete variables could also be made. This wasn't done because there are no packages in python that can easily optimize a function with discrete variables, while there are some mathematical methods to do this (Rajeev & Krishnamoorthy, 1992). When this would be implemented in the program, the results could be closer to the cost-optimal design.

The width that model 1 designs for beams with a large length is now often higher than required, which unnecessarily increases the cost of the designed beams. The program now makes a width

array based on the height of the beam with a minimum value of 1/3 of the height. This is not a requirement given in the Eurocode and if it was omitted, then the width would be closer to those found by model 2.

6.2 Recommendations

Based on the results in this report, some new discussion points emerged. Recommendations for further research are:

- A better design is obtained if the used optimisation method in the program makes use of a discrete optimisation algorithm instead of a continuous method.
- Use of other types of beams could be included in the program to give better alternatives if the load or span becomes too high. And to reduce the amount of materials that is needed to fulfil the strength requirements. The design of prefabricated concrete with prestressed reinforcement could for example be implemented in the program.
- The shape of the beam could be made variable. This could lead to designs that make better use of the materials, increasing the internal lever arm with the same amount of materials.
- The ranges of height and width of the beam could be improved. That will improve the calculation speed of the models. When defining this range, the load on the beam should also be included in the function. Some research should be done to get a function that gives an accurate estimation.

7 - Bibliography

Arafa, M. A. (2011). Early stage cost estimation of buildings construction projects using artificial neural networks. *Journal of Artificial Intelligence, 4*(1), 63-75. Retrieved from http://hdl.handle.net/20.500.12358/24892

Chakrabarty, B. (1992). Models for optimal design of reinforced concrete beams. *Computers & Structures, 42*(3), 447-451. doi:https://doi.org/10.1016/0045-7949(92)90040-7

de Medeiros, G., & Kripka, M. (2014). Optimization of reinforced concrete comlumns according to different environmental impact assessment parameters. *Engineering Structures, 59*, 185-194. doi:https://doi.org/10.1016/j.engstruct.2013.10.045

Esfandiari, M., Urgessa, G., Sheikholarefin, S., & Dehghan Manshadi, S. (2018). Optimum design of 3D reinforced concrete frames using DMPSO algorithm. *Advances in Engineering Software, 115*, 149-160. doi:https://doi.org/10.1016/j.advengsoft.2017.09.007

European Union. (2004, April). Eurocode 2: Design of concrete structures - Part 1-1 : General rules and rules for buildings. *EN 1992-1-1 9.2*.

Hartsuijker, C. (1999). *Toegepaste Mechanica, deel 1 - Evenwicht.* Academic Service.

Hordijk, D., & Lagendijk, P. (2017, November 21). *CTB2220 Beton & Staalconstructies.* Retrieved from B2 Buiging: https://brightspace.tudelft.nl/d2l/le/content/49671/viewContent/684155/View

Hordijk, D., & Lagendijk, P. (2017, December 4). *CTB2220 Beton & Staalconstructies*. Retrieved from B5 Dwarskracht: https://brightspace.tudelft.nl/d2l/le/content/49671/viewContent/696872/View

Huang, M. R. (2018). Artificial Intelligence in Service. *Journal of Service Research, 21*(2), 155-172. doi:https://doi.org/10.1177/1094670517752459

Kaveh, A., & Behnam, A. (2013). Design optimization of reinforced concrete 3D structures considereing frequency constraints via a charged system search. *Scientia Iranica, 20*(3), 387-396. doi:https://doi.org/10.1016/j.scient.2012.11.017

Kaveh, A., & Talatahari, S. (2010). Novel heuristic optimization method: charged system search. *Acta Mechanica, 213*(3), 267-289. doi:https://doi.org/10.1007/s00707-009-0270-4

NEOS. (2019). *neos-guide.org*. Retrieved from Sequential Quadratic Programming: https://neos-guide.org/content/sequential-quadratic-programming

Rajeev, S., & Krishnamoorthy, C. (1992). Discrete Optimization of Structures Using Genetic Algorithms. *Journal of Structural Engineering, 118*(5), 1233-1250. doi:https://doi.org/10.1061/(ASCE)0733-9445(1992)118:5(1233)

SciPy. (2019, May 17). *scipy.org*. Retrieved from scipy.optimize.curve_fit: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html

SciPy.org. (2019, May 17). *scipy.org*. Retrieved from scipy.optimize.minimize: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html

SciPy.org. (2019, May 17). *scipy.org*. Retrieved from Optimization and Root Finding: https://docs.scipy.org/doc/scipy/reference/optimize.html

van Rotterdam, E. (2005). *Sterkteleer 1: toegepaste mechanica.* Delta Press.

Wagemans, L., Soons, F., & Raaij, B. v. (2014). *Quick Reference.* Delft: Section Structural and Building Engineering.

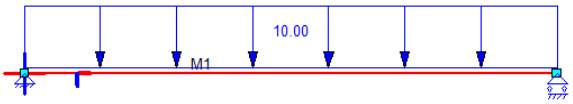
Wubs, T. (2016). *De mogelijkheden en beperkingen van een computermodel dat het optimale ontwerp bepaalt van een uitkragende, vlakke vakwerkligger.* Delft University of Technology. Retrieved from http://homepage.tudelft.nl/p3r3s/BSc_projects/eindrapport_wubs.pdf

Appendices

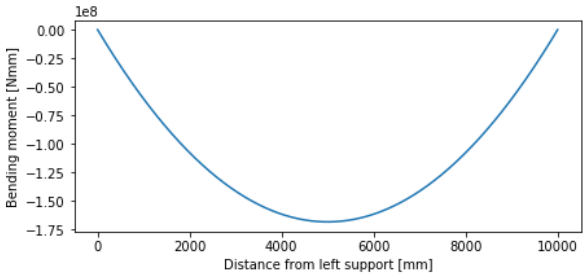
A - Test case A

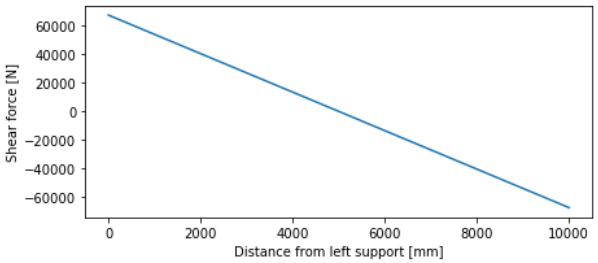
Model without optimisation algorithm

Input	
Left support	Hinge
Right support	Moveable
Length	10 [m]
Load	10 [kN/m^2]



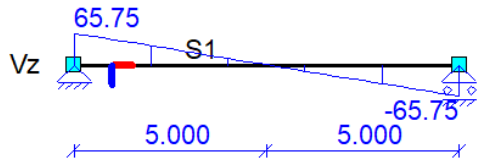
Output		
Height	500	mm
Width	210	mm
Used diameter	25	mm
Number of cables	2	-
Moment capacity	167	kNm
Max shear resistance	444	kN
Spacing shear reinf.	300	mm
Cost	336	€
Calculation time	34	s

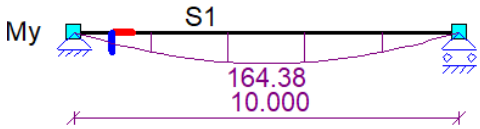




Verification

Output MatrixFrame:





Hand calculations:

$$A_s = 2 \cdot \frac{1}{4} \cdot \pi \cdot 25^2 = 981.75\ mm^2$$

$$N_s = 981.75 \cdot 434.78 = 426846.83\ N$$

$$x_u = \frac{426846.83}{20 \cdot 210 \cdot 0.75} = 135.51\ mm$$

$$z = 500 - 35 - 8 - 0.5 \cdot 25 - 0.39 \cdot 135.51 = 391.65\ mm$$

$$M_{Rd} = 427060.25 \cdot 391.65 \cdot 10^{-6} = 167.176\ kNm$$

Output model 2		
Height	520	mm
Width	200	mm
Used diameter	25	mm
Number of cables	2	-
Moment capacity	175	kNm
Max shear resistance	357	kN
Spacing shear reinf.	300	mm
Cost	335	€
Calculation time	3	s

Hand calculations:

$$A_s = 2 \cdot \frac{1}{4} \cdot \pi \cdot 25^2 = 981.75\ mm^2$$

$$N_s = 981.75 \cdot 434.78 = 426844.27\ N$$

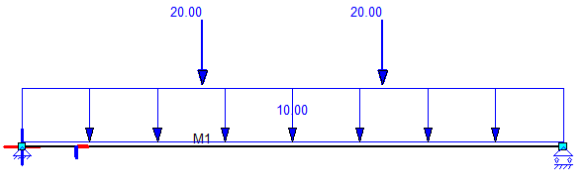
$$x_u = \frac{426844.27}{20 \cdot 200 \cdot 0.75} = 142.28\ mm$$

$$z = 520 - 35 - \ 8 - 0.5 \cdot 25 - 0.39 \cdot 142.28 = 409.01\ mm$$

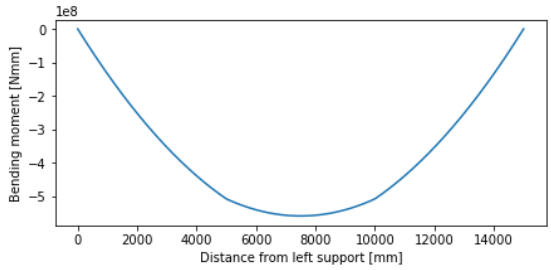
$$M_{Rd} = 426844.27 \cdot 409.01 \cdot 10^{-6} = 174.584\ kNm$$

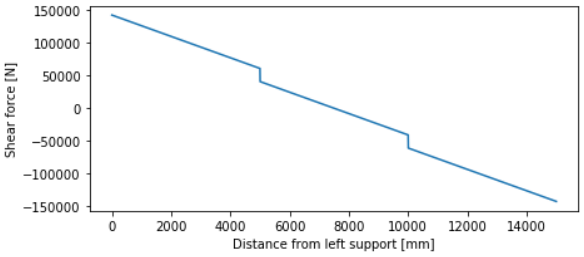
B - Test case B

Input	
Left support	Hinge
Right support	Moveable
Length	15 [m]
Distributed load	10 [kN/m^2]
Point loads	20 [kN]



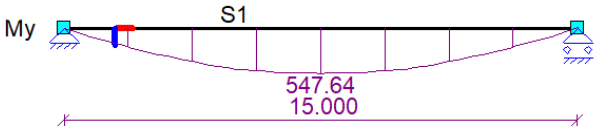
Output model 1		
Height	680	mm
Width	290	mm
Used diameter	32	mm
Number of cables	3	-
Moment capacity	553	kNm
Max shear resistance	856	kN
Spacing shear reinf.	300	mm
Cost	1051	€
Calculation time	118	s





Verification

Output MatrixFrame:



Hand calculations:

$$A_s = 3 \cdot \frac{1}{4} \cdot \pi \cdot 32^2 = 2412.74\ mm^2$$

$$N_s = 2412.74 \cdot 434.78 = 1049012.47\ N$$

$$x_u = \frac{1049012.47}{20 \cdot 290 \cdot 0.75} = 241.15\ mm$$

$$z = 680 - 35 - 8 - 0.5 \cdot 32 - 0.39 \cdot 241.15 = 526.95\ mm$$

$$M_{Rd} = 1049012.47 \cdot 526.95 \cdot 10^{-6} = 552.778\ kNm$$

Output model 2		
Height	870	mm
Width	250	mm
Used diameter	32	mm
Number of cables	3	-
Moment capacity	736	kNm
Max shear resistance	790	kN
Spacing shear reinf.	300	mm
Cost	1108	€
Calculation time	13	s

Hand calculations:

$$A_s = 3 \cdot \frac{1}{4} \cdot \pi \cdot 32^2 = 2412.74\ mm^2$$

$$N_s = 2412.74 \cdot 434.78 = 1049012.47\ N$$

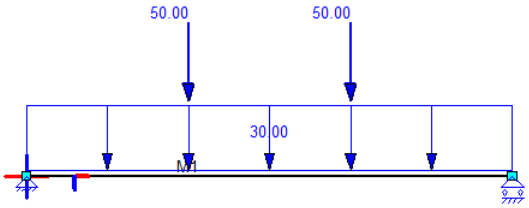
$$x_u = \frac{1049012.47}{20 \cdot 250 \cdot 0.75} = 279.74\ mm$$

$$z = 870 - 35 - 8 - 0.5 \cdot 32 - 0.39 \cdot 279.74 = 701.90\ mm$$

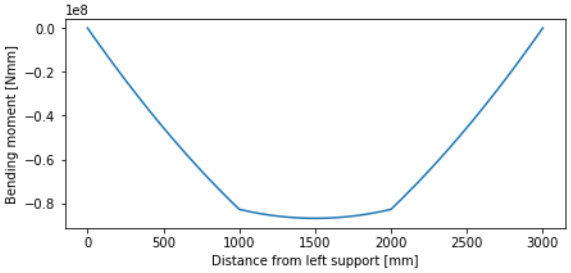
$$M_{Rd} = 1049012.47 \cdot 526.95 \cdot 10^{-6} = 736.305\ kNm$$

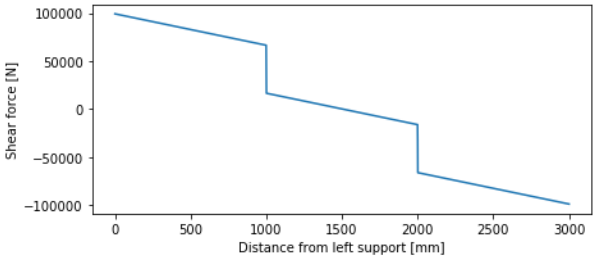
C - Test case C

Input	
Left support	Hinge
Right support	Moveable
Length	3 [m]
Distributed load	30 [kN/m^2]
Point loads	50 [kN]



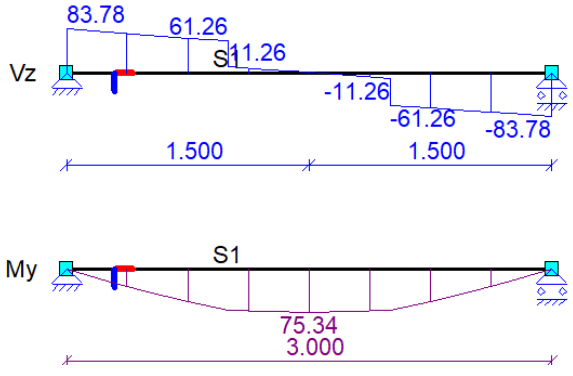
Output model 1		
Height	420	mm
Width	200	mm
Used diameter	20	mm
Number of cables	2	-
Moment capacity	91	kNm
Max shear resistance	349	kN
Spacing shear reinf.	253	mm
Cost	76	€
Calculation time	0.05	s





Verification

Output MatrixFrame:



Hand calculations:

$$A_s = 2 \cdot \frac{1}{4} \cdot \pi \cdot 20^2 = 628.32\ mm^2$$

$$N_s = 628.32 \cdot 434.78 = 273180.33\ N$$

$$x_u = \frac{273180.33}{20 \cdot 200 \cdot 0.75} = 91.06\ mm$$

$$z = 420 - 35 - 8 - 0.5 \cdot 20 - 0.39 \cdot 91.06 = 331.49\ mm$$

$$M_{Rd} = 273180.33 \cdot 331.49 \cdot 10^{-6} = 90.556\ kNm$$

Output model 2		
Height	410	mm
Width	200	mm
Used diameter	25	mm
Number of cables	2	-
Moment capacity	128	kNm
Max shear resistance	282	kN
Spacing shear reinf.	245	mm
Cost	92	€
Calculation time	2	s

Hand calculations:

$$A_s = 2 \cdot \frac{1}{4} \cdot \pi \cdot 25^2 = 981.75\ mm^2$$

$$N_s = 981.75 \cdot 434.78 = 426844.27\ N$$

$$x_u = \frac{426844.27}{20 \cdot 200 \cdot 0.75} = 142.28\ mm$$

$$z = 410 - 35 - 8 - 0.5 \cdot 25 - 0.39 \cdot 142.28 = 299.01\ mm$$

$$M_{Rd} = 426844.27 \cdot 299.01 \cdot 10^{-6} = 127.631\ kNm$$

D - Code computer model 1

```python
import numpy as np
import matplotlib.pyplot as plt
import math
import time


def moment_capacity (beam_height, beam_width, concrete_cover, stirrup_diameter, steel_diameter, number_of_cables):
    As = 0.25 * np.pi * steel_diameter ** 2 * number_of_cables                          # The cross-sectional area of the reinforcement steel
    Ns = As * steel_fyd                                                                 # Force in the reinforcement steel when it is yielding
    Xu = Ns / (alpha * beam_width * concrete_fcd)                                       # Height of the concrete compression zone
    d = beam_height - concrete_cover - stirrup_diameter - 0.5 * steel_diameter          # effective height

    if (Xu / d) > ((3.5 * 10 ** 3) / (3.5 * 10 ** 3 + 7 * steel_fyd)):
        return 0

    rho = As / (beam_height * beam_width)

    if (rho > 0.04) or (rho < (0.26 * concrete_fctm / steel_fyk)):
        return 0

    z = d - beta * Xu                                                                   # Internal lever arm
    return Ns * z

def costs(beam_height, beam_width, beam_length, concrete_cost, steel_diameter, number_of_cables, number_of_stirrups, stirrup_diameter, concrete_cover):
    volume_concrete = beam_height * beam_width * beam_length / 10 ** 9
    weight_steel = 0.25 * np.pi * steel_diameter ** 2 * number_of_cables * beam_length / 10 ** 9 * 7850
    weight_stirrups = (0.25 * np.pi * stirrup_diameter ** 2 * number_of_stirrups * (2 * (beam_height - 2 * concrete_cover) + 2 * (beam_width - 2 * concrete_cover))) / 10 ** 9 * 7850

    return volume_concrete * concrete_cost + weight_steel * 2 + weight_stirrups * 3


def shear_resistance_no_stirrups(beam_width, beam_height, concrete_cover, steel_diameter, number_of_cables):
    effective_height_d = beam_height - concrete_cover - 0.5 * steel_diameter
    k = min(1 + np.sqrt(200 / effective_height_d), 2)
    As = 0.25 * np.pi * steel_diameter ** 2 * number_of_cables
    rho = min(As / (beam_width * effective_height_d), 0.02)

    maximum_allowable_shear_stress = max(0.12 * k * (100 * rho * concrete_fck) ** (1/3), 0.035 * k ** 1.5 * np.sqrt(concrete_fck))
    maximum_allowable_shear_force = maximum_allowable_shear_stress * beam_width * effective_height_d

    return maximum_allowable_shear_force

def determine_spacing_stirrups(beam_height, concrete_cover, steel_diameter, stirrup_diameter, max_shear_force, theta, fywd):
    Asw = 0.5 * np.pi * stirrup_diameter ** 2
    z = 0.9 * (beam_height - concrete_cover - stirrup_diameter - 0.5 * steel_diameter)
    return min(Asw * z * fywd / (np.tan(math.radians(theta)) * max_shear_force), 300)

def max_shear_resistance(beam_width, beam_height, concrete_cover, steel_diameter, stirrup_diameter):
    v1 = 0.6 * (1 - concrete_fck / 250) # eurocode 6.6N
    z = 0.9 * (beam_height - concrete_cover - stirrup_diameter - 0.5 * steel_diameter)
    return beam_width * z * v1 * concrete_fcd / 2 # highest capacity for theta = 45 degrees, cot(theta) + tan(theta) = 2


class load:
    def __init__(self, name, loadtype, location, value):
        self.name = name
        self.loadtype = loadtype   # 'point' or 'distributed'
        self.location = location
        self.value = value


class support:
    def __init__(self, supporttype, location):
        self.supporttype = supporttype # spring suppport not yet supported (only none, moveable, hinge, fixed)
        self.location = location

def create_line(load, support_left, support_right):
    if load.location >= support_right.location or load.location <= support_left.location:
        print('wrong location for', load.name)

    shear_line = np.ones(abs(support_right.location - support_left.location) + 1) # array, values for every point between supports, steps of mm
    moment_line = np.zeros(abs(support_right.location - support_left.location) + 1)

    a = abs(support_left.location - load.location)    # distance left side to load
    b = abs(support_right.location - load.location)   # distance load to right side
    length = support_right.location - support_left.location

    if load.loadtype == 'point':
        if (support_left.supporttype == 'hinge' or support_left.supporttype == 'moveable') and (support_right.supporttype == 'moveable' or support_right.supporttype == 'hinge'):
            shear_force_left_support = load.value * b / (a + b)
            shear_line[0 : load.location] = shear_line[0:load.location] * shear_force_left_support
            shear_line[load.location] = 0
            shear_line[load.location + 1:support_right.location - support_left.location + 1] = shear_line[load.location + 1:support_right.location - support_left.location + 1] * (shear_force_left_support - load.value)

            for i in range(len(moment_line) - 1):
                moment_line[i + 1] = moment_line[i] - (shear_line[i] + shear_line[i + 1]) / 2
            return [shear_line, moment_line]

        # both sides supported by hinges, force determined by hand calculation

        if (support_left.supporttype == 'fixed' and support_right.supporttype == 'hinge') or (support_left.supporttype == 'fixed' and support_right.supporttype == 'moveable'):
            shear_force_left_support = (load.value * b * (3 * length ** 2 - b ** 2)) / (2 * length ** 3) # left support force with forget-me-not
            shear_line[0 : load.location] = shear_line[0:load.location] * shear_force_left_support
            shear_line[load.location : support_right.location - support_left.location] = shear_line[load.location : support_right.location - support_left.location] * (shear_force_left_support - load.value)

            moment_line[0] = load.value * b * (length ** 2 - b ** 2) / (2 * length ** 2)
            for i in range(len(moment_line) - 1):
                moment_line[i + 1] = moment_line[i] - (shear_line[i] + shear_line[i + 1]) / 2
            return [shear_line, moment_line]

        # left side fixed support, right side hinge / moveable support (force determined by 'vergeet-me-nietje')

        if (support_left.supporttype == 'hinge' and support_right.supporttype == 'fixed') or (support_left.supporttype == 'moveable' and support_right.supporttype == 'fixed'):
            shear_force_right_support = (load.value * a * (3 * length ** 2 - a ** 2)) / (2 * length ** 3) # left support force with forget-me-not
            shear_line[0 : load.location] = shear_line[0:load.location] * (shear_force_right_support - load.value)
            shear_line[load.location : support_right.location - support_left.location] = shear_line[load.location : support_right.location - support_left.location] * (shear_force_right_support)
```

```
            moment_line[0] = 0
            for i in range(len(moment_line) - 1):
                moment_line[i + 1] = moment_line[i] + (shear_line[i] + shear_line[i + 1]) / 2
            return [shear_line, moment_line]

        # left side hinge / moveable, right side fixed (force determined by 'vergeet-me-nietje')

        if (support_left.supporttype == 'fixed' and support_right.supporttype == 'fixed'):
            shear_force_left_support = load.value * b ** 2 * (3 * a + b) / (length ** 3)
            shear_line[0:load.location] = shear_line[0:load.location] * shear_force_left_support
            shear_line[load.location:support_right.location] = shear_line[load.location:support_right.location] * (shear_force_left_support - load.value)

            moment_line[0] = load.value * a * b ** 2 / length ** 2
            for i in range(len(moment_line) - 1):
                moment_line[i + 1] = moment_line[i] - (shear_line[i] + shear_line[i + 1]) / 2
            return [shear_line, moment_line]

        # point load on a beam on two fixed supports (force determined by 'vergeet-me-nietje')

    if load.loadtype == 'distributed':
        if (support_left.supporttype == 'hinge' or support_left.supporttype == 'moveable') and (support_right.supporttype == 'moveable' or support_right.supporttype == 'hinge'):
            shear_force_left_support = 0.5 * load.value * length
            shear_line[0:support_right.location + 1] = np.linspace(shear_force_left_support, - shear_force_left_support, length + 1)

            moment_line[0] = 0
            for i in range(len(moment_line) - 1):
                moment_line[i + 1] = moment_line[i] - (shear_line[i] + shear_line[i + 1]) / 2
            return [shear_line, moment_line]

        # distributed load on two hinge supports

        if (support_left.supporttype == 'fixed' and support_right.supporttype == 'hinge') or (support_left.supporttype == 'fixed' and support_right.supporttype == 'moveable'):
            shear_force_left_support = 5 / 8 * load.value * length
            shear_force_right_support = - 3 / 8 * load.value * length
            shear_line[0:] = np.linspace(shear_force_left_support, shear_force_right_support, length + 1)

            moment_line[0] = 1 / 8 * load.value * length ** 2
            for i in range(len(moment_line) - 1):
                moment_line[i + 1] = moment_line[i] - (shear_line[i] + shear_line[i + 1]) / 2
            return [shear_line, moment_line]

        # distributed load, left side fixed right side hinge

        if (support_left.supporttype == 'hinge' and support_right.supporttype == 'fixed') or (support_left.supporttype == 'moveable' and support_right.supporttype == 'fixed'):
            shear_force_left_support = 3 / 8 * load.value * length
            shear_force_right_support = - 5 / 8 * load.value * length
            shear_line[0:] = np.linspace(shear_force_left_support, shear_force_right_support, length + 1)

            moment_line[0] = 0
            for i in range(len(moment_line) - 1):
                moment_line[i + 1] = moment_line[i] - (shear_line[i] + shear_line[i + 1]) / 2
            return [shear_line, moment_line]

        # distributed load, left side hinge right side fixed

        if (support_left.supporttype == 'fixed' and support_right.supporttype == 'fixed'):
            shear_force_left_support = 0.5 * load.value * length
            shear_line[0:] = np.linspace(shear_force_left_support, - shear_force_left_support, length + 1)

            moment_line[0] = 1 / 12 * load.value * length ** 2
            for i in range(len(moment_line) - 1):
                moment_line[i + 1] = moment_line[i] - (shear_line[i] + shear_line[i+1]) / 2
            return [shear_line, moment_line]




optimal_design = [1, 1, 1, 1, 1]
lowest_price = 1000000000000000

# list of default values

alpha = 0.75
beta = 0.39

# material properties
# values in N/mm2

steel_fyk = 500         #characteristic yield strength of reinforcement steel B500\
concrete_fck = 30       #characteristic compressive yield strength of concrete C30
concrete_fctm = 2.9     #concrete tensile strength

# factors:

gamma_s = 1.15      # material factor steel
gamma_c = 1.5       # material factor concrete

# design values in N/mm2:

steel_fyd = steel_fyk / gamma_s
concrete_fcd = concrete_fck / gamma_c
steel_fywd = 435        # design yield stress of shear reinforcement

# variables with no proper source yet:

theta = 30              # mm
stirrup_diameter = 8    # mm
concrete_cover = 35     # mm
dg = 30                 # mm (max grain size)
k2 = 5                  # mm (extra distance between bars)

# Costs of concrete and steel

concrete_cost = 130
```

```
# Mainloop

start = time.time()


support_left = support('hinge', 0)
support_right = support('moveable', 3000)
beam_length = support_right.location - support_left.location

loads = []

#load_1 = load('load 1', 'point', 1000, 50000)
#loads.append(load_1)

load_2 = load('load 2', 'distributed', 1000, 30)
loads.append(load_2)

#load_3 = load('load 3', 'point', 2000, 50000)
#loads.append(load_3)

for i in range(len(loads)):
    lines = create_line(loads[i], support_left, support_right)

    if i == 0:
        moment_line = lines[1]
        shear_line = lines[0]
    if i > 0:
        for j in range(len(moment_line)):
            moment_line[j] = moment_line[j] + lines[1][j]
            shear_line[j] = shear_line[j] + lines[0][j]

# any restrictions to the dimensions can be specified here

beam_height_max = 0
beam_height_max_free = True

beam_height_min = 0
beam_height_min_free = True

beam_width_max = 0
beam_width_max_free = True

beam_width_min = 0
beam_width_min_free = True

diameter = [12, 16, 20, 25, 32]

if beam_height_max_free:
    beam_height_max = beam_length / 10 * 1.5
    beam_height_max = round(beam_height_max / 10) * 10
if beam_height_min_free:
    beam_height_min = max(beam_length / 15 * 0.5, 200)
    beam_height_min = round(beam_height_min / 10) * 10

height = np.arange(beam_height_min, beam_height_max, 10)

for beam_height in height:
    if beam_width_max_free:
        beam_width_max = max(round((beam_height / 2) / 10) * 10, 220)
    if beam_width_min_free:
        beam_width_min = max(round((beam_height / 3) / 10) * 10, 200)

    width = np.arange(beam_width_min, beam_width_max, 10)

    for beam_width in width:
        self_weight = load('self weight', 'distributed', 1, beam_height * beam_width * 0.000025 * 1.2)
        lines_weight = create_line(self_weight, support_left, support_right)
        moment_line_temp = moment_line.copy()
        shear_line_temp = shear_line.copy()

        for j in range(len(lines_weight[0])):
            moment_line_temp[j] = moment_line_temp[j] + lines_weight[1][j]
            shear_line_temp[j] = shear_line_temp[j] + lines_weight[0][j]

        maximum_moment = min(moment_line_temp)
        maximum_shear = max(abs(shear_line_temp))

        for steel_diameter in diameter:
            max_shear_beam = max_shear_resistance(beam_width, beam_height, concrete_cover, steel_diameter, stirrup_diameter)
            if (max_shear_beam < maximum_shear):
                continue

            max_number = round((beam_width - 2 * concrete_cover - 2 * stirrup_diameter + dg + k2) / (steel_diameter + dg + k2))
            number = np.arange(2, max_number, 1)

            for number_of_cables in number:
                moment_capacity_beam = moment_capacity(beam_height, beam_width, concrete_cover, stirrup_diameter, steel_diameter, number_of_cables)
                if moment_capacity_beam < abs(maximum_moment):
                    continue

                spacing = determine_spacing_stirrups(beam_height, concrete_cover, steel_diameter, stirrup_diameter, maximum_shear, theta, steel_fywd)
                number_of_stirrups = round(beam_length / spacing)

                cost = costs(beam_height, beam_width, beam_length, concrete_cost, steel_diameter, number_of_cables, number_of_stirrups, stirrup_diameter, concrete_cover)
                if cost < lowest_price:
                    optimal_design = [beam_height, beam_width, steel_diameter, number_of_cables, spacing, moment_capacity_beam, max_shear_beam]
                    lines_optimal_design = [moment_line_temp, shear_line_temp]
                    lowest_price = cost

end = time.time()
print('runtime:', end - start)
print('Height:', optimal_design[0])
print('Width:', optimal_design[1])
print('Diameter reinforcement:', optimal_design[2])
print('Number of cables:', optimal_design[3])
print('Spacing shear reinforcement:', optimal_design[4])
print('Cost:', lowest_price)
print('moment capacity:', optimal_design[5])
print('Shear force capacity:', optimal_design[6])
x = np.arange(0, beam_length + 1, 1)

plt.figure(figsize=(7,3))
plt.plot(x, lines_optimal_design[0])
plt.xlabel('Distance from left support [mm]')
plt.ylabel('Bending moment [Nmm]')

plt.figure(figsize=(7,3))
plt.plot(x, lines_optimal_design[1])
plt.xlabel('Distance from left support [mm]')
plt.ylabel('Shear force [N]')

```

E - Code computer model 2

```python
from scipy.optimize import minimize
import numpy as np
import math
import time

def moment_capacity (beam_height, beam_width, concrete_cover, stirrup_diameter, steel_diameter, number_of_cables):
    As = 0.25 * np.pi * steel_diameter ** 2 * number_of_cables                  # The cross-sectional area of the reinforcement steel
    Ns = As * steel_fyd                                                         # Force in the reinforcement steel when it is yielding
    Xu = Ns / (alpha * beam_width * concrete_fcd)                               # Height of the concrete compression zone
    d = beam_height - concrete_cover - stirrup_diameter - 0.5 * steel_diameter  # effective height

    if (Xu / d) > ((3.5 * 10 ** 3) / (3.5 * 10 ** 3 + 7 * steel_fyd)):
        return 0

    rho = As / (beam_height * beam_width)

    if (rho > 0.04) or (rho < (0.26 * concrete_fctm / steel_fyk)):
        return 0

    z = d - beta * Xu                                                           # Internal lever arm
    return Ns * z

def costs(beam_height, beam_width, beam_length, concrete_cost, steel_diameter, number_of_cables, number_of_stirrups, stirrup_diameter, concrete_cover):
    volume_concrete = beam_height * beam_width * beam_length / 10 **9
    weight_steel = 0.25 * np.pi * steel_diameter ** 2 * number_of_cables * beam_length / 10 ** 9 * 7850
    weight_stirrups = (0.25 * np.pi * stirrup_diameter ** 2 * number_of_stirrups * (2 * (beam_height - 2 * concrete_cover) + 2 * (beam_width - 2 * concrete_cover))) / 10 ** 9 * 7850

    return volume_concrete * concrete_cost + weight_steel * 2 + weight_stirrups * 3

def shear_resistance_no_stirrups(beam_width, beam_height, concrete_cover, steel_diameter, number_of_cables):
    effective_height_d = beam_height - concrete_cover - 0.5 * steel_diameter
    k = min(1 + np.sqrt(200 / effective_height_d), 2)
    As = 0.25 * np.pi * steel_diameter ** 2 * number_of_cables
    rho = min(As / (beam_width * effective_height_d), 0.02)

    maximum_allowable_shear_stress = max(0.12 * k * (100 * rho * concrete_fck) ** (1/3), 0.035 * k ** 1.5 * np.sqrt(concrete_fck))
    maximum_allowable_shear_force = maximum_allowable_shear_stress * beam_width * effective_height_d

    return maximum_allowable_shear_force

def determine_spacing_stirrups(beam_height, concrete_cover, steel_diameter, stirrup_diameter, max_shear_force, theta, fywd):
    Asw = 0.5 * np.pi * stirrup_diameter ** 2
    z = 0.9 * (beam_height - concrete_cover - stirrup_diameter - 0.5 * steel_diameter)
    return min(Asw * z * fywd / (np.tan(math.radians(theta)) * max_shear_force), 300)

def max_shear_resistance(beam_width, beam_height, concrete_cover, steel_diameter, stirrup_diameter):
    v1 = 0.6 * (1 - concrete_fck / 250) # eurocode 6.6N
    z = 0.9 * (beam_height - concrete_cover - stirrup_diameter - 0.5 * steel_diameter)
    return beam_width * z * v1 * concrete_fcd / 2 # highest capacity for theta = 45 degrees, cot(theta) + tan(theta) = 2


class load:
    def __init__(self, name, loadtype, location, value):
        self.name = name
        self.loadtype = loadtype   # 'point' or 'distributed'
        self.location = location
        self.value = value


class support:
    def __init__(self, supporttype, location):
        self.supporttype = supporttype # spring suppport not yet supported (only none, moveable, hinge, fixed)
        self.location = location

def create_line(load, support_left, support_right):
    if load.location >= support_right.location or load.location <= support_left.location:
        print('wrong location for', load.name)

    shear_line = np.ones(abs(support_right.location - support_left.location) + 1) # array, values for every point between supports, steps of mm
    moment_line = np.zeros(abs(support_right.location - support_left.location) + 1)

    a = abs(support_left.location - load.location)    # distance left side to load
    b = abs(support_right.location - load.location)   # distance load to right side
    length = support_right.location - support_left.location

    if load.loadtype == 'point':
        if (support_left.supporttype == 'hinge' or support_left.supporttype == 'moveable') and (support_right.supporttype == 'moveable' or support_right.supporttype == 'hinge'):
            shear_force_left_support = load.value * b / (a + b)
            shear_line[0 : load.location] = shear_line[0:load.location] * shear_force_left_support
            shear_line[load.location] = 0
            shear_line[load.location + 1:support_right.location - support_left.location + 1] = shear_line[load.location + 1:support_right.location - support_left.location + 1] * (shear_force_left_support - load.value)

            for i in range(len(moment_line) - 1):
                moment_line[i + 1] = moment_line[i] - (shear_line[i] + shear_line[i + 1]) / 2
            return [shear_line, moment_line]

        # both sides supported by hinges, force determined by hand calculation

        if (support_left.supporttype == 'fixed' and support_right.supporttype == 'hinge') or (support_left.supporttype == 'fixed' and support_right.supporttype == 'moveable'):
            shear_force_left_support = (load.value * b * (3 * length ** 2 - b ** 2)) / (2 * length ** 3) # left support force with forget-me-not
            shear_line[0 : load.location] = shear_line[0:load.location] * shear_force_left_support
            shear_line[load.location : support_right.location - support_left.location] = shear_line[load.location : support_right.location - support_left.location] * (shear_force_left_support - load.value)

            moment_line[0] = load.value * b * (length ** 2 - b ** 2) / (2 * length ** 2)
            for i in range(len(moment_line) - 1):
                moment_line[i + 1] = moment_line[i] - (shear_line[i] + shear_line[i + 1]) / 2
            return [shear_line, moment_line]

        # left side fixed support, right side hinge / moveable support (force determined by 'vergeet-me-nietje')

        if (support_left.supporttype == 'hinge' and support_right.supporttype == 'fixed') or (support_left.supporttype == 'moveable' and support_right.supporttype == 'fixed'):
            shear_force_right_support = (load.value * a * (3 * length ** 2 - a ** 2)) / (2 * length ** 3) # left support force with forget-me-not
            shear_line[0 : load.location] = shear_line[0:load.location] * (shear_force_right_support - load.value)
            shear_line[load.location : support_right.location - support_left.location] = shear_line[load.location : support_right.location - support_left.location] * (shear_force_right_support)

            moment_line[0] = 0
            for i in range(len(moment_line) - 1):
                moment_line[i + 1] = moment_line[i] + (shear_line[i] + shear_line[i + 1]) / 2
            return [shear_line, moment_line]

```

```
        # left side hinge / moveable, right side fixed (force determined by 'vergeet-me-nietje')

        if (support_left.supporttype == 'fixed' and support_right.supporttype == 'fixed'):
            shear_force_left_support = load.value * b ** 2 * (3 * a + b) / (length ** 3)
            shear_line[0:load.location] = shear_line[0:load.location] * shear_force_left_support
            shear_line[load.location:support_right.location] = shear_line[load.location:support_right.location] * (shear_force_left_support - load.value)

            moment_line[0] = load.value * a * b ** 2 / length ** 2
            for i in range(len(moment_line) - 1):
                moment_line[i + 1] = moment_line[i] - (shear_line[i] + shear_line[i + 1]) / 2
            return [shear_line, moment_line]

        # point load on a beam on two fixed supports (force determined by 'vergeet-me-nietje')

    if load.loadtype == 'distributed':
        if (support_left.supporttype == 'hinge' or support_left.supporttype == 'moveable') and (support_right.supporttype == 'moveable' or support_right.supporttype == 'hinge'):
            shear_force_left_support = 0.5 * load.value * length
            shear_line[0:support_right.location + 1] = np.linspace(shear_force_left_support, - shear_force_left_support, length + 1)

            moment_line[0] = 0
            for i in range(len(moment_line) - 1):
                moment_line[i + 1] = moment_line[i] - (shear_line[i] + shear_line[i + 1]) / 2
            return [shear_line, moment_line]

        # distributed load on two hinge supports

        if (support_left.supporttype == 'fixed' and support_right.supporttype == 'hinge') or (support_left.supporttype == 'fixed' and support_right.supporttype == 'moveable'):
            shear_force_left_support = 5 / 8 * load.value * length
            shear_force_right_support = - 3 / 8 * load.value * length
            shear_line[0:] = np.linspace(shear_force_left_support, shear_force_right_support, length + 1)

            moment_line[0] = 1 / 8 * load.value * length ** 2
            for i in range(len(moment_line) - 1):
                moment_line[i + 1] = moment_line[i] - (shear_line[i] + shear_line[i + 1]) / 2
            return [shear_line, moment_line]

        # distributed load, left side fixed right side hinge

        if (support_left.supporttype == 'hinge' and support_right.supporttype == 'fixed') or (support_left.supporttype == 'moveable' and support_right.supporttype == 'fixed'):
            shear_force_left_support = 3 / 8 * load.value * length
            shear_force_right_support = - 5 / 8 * load.value * length
            shear_line[0:] = np.linspace(shear_force_left_support, shear_force_right_support, length + 1)

            moment_line[0] = 0
            for i in range(len(moment_line) - 1):
                moment_line[i + 1] = moment_line[i] - (shear_line[i] + shear_line[i + 1]) / 2
            return [shear_line, moment_line]

        # distributed load, left side hinge right side fixed

        if (support_left.supporttype == 'fixed' and support_right.supporttype == 'fixed'):
            shear_force_left_support = 0.5 * load.value * length
            shear_line[0:] = np.linspace(shear_force_left_support, - shear_force_left_support, length + 1)

            moment_line[0] = 1 / 12 * load.value * length ** 2
            for i in range(len(moment_line) - 1):
                moment_line[i + 1] = moment_line[i] - (shear_line[i] + shear_line[i+1]) / 2
            return [shear_line, moment_line]

def mainfunction(array):
    self_weight = load('self weight', 'distributed', 1, array[0] * array[1] * 0.000025 * 1.2)
    lines_weight = create_line(self_weight, support_left, support_right)
    moment_line_temp = moment_line.copy()
    shear_line_temp = shear_line.copy()

    for j in range(len(lines_weight[0])):
        moment_line_temp[j] = moment_line_temp[j] + lines_weight[1][j]
        shear_line_temp[j] = shear_line_temp[j] + lines_weight[0][j]

    maximum_shear = max(abs(shear_line_temp))

    spacing = determine_spacing_stirrups(array[0], concrete_cover, array[2], stirrup_diameter, maximum_shear, theta, steel_fywd)
    number_of_stirrups = round(beam_length / spacing)

    cost = costs(array[0], array[1], beam_length, concrete_cost, array[2], array[3], number_of_stirrups, stirrup_diameter, concrete_cover)

    return cost


def constraint_1(array):
    self_weight = load('self weight', 'distributed', 1, array[0] * array[1] * 0.000025 * 1.2)
    lines_weight = create_line(self_weight, support_left, support_right)
    moment_line_temp = moment_line.copy()
    shear_line_temp = shear_line.copy()

    for j in range(len(lines_weight[0])):
        moment_line_temp[j] = moment_line_temp[j] + lines_weight[1][j]
        shear_line_temp[j] = shear_line_temp[j] + lines_weight[0][j]

    maximum_moment = min(moment_line_temp)
    moment_capacity_beam = moment_capacity(array[0], array[1], concrete_cover, stirrup_diameter, array[2], array[3])

    return moment_capacity_beam - abs(maximum_moment)

def constraint_2(array):

    self_weight = load('self weight', 'distributed', 1, array[0] * array[1] * 0.000025 * 1.2)
    lines_weight = create_line(self_weight, support_left, support_right)
    shear_line_temp = shear_line.copy()

    for j in range(len(lines_weight[0])):
        shear_line_temp[j] = shear_line_temp[j] + lines_weight[0][j]

    maximum_shear = max(abs(shear_line_temp))
    max_shear_beam = max_shear_resistance(array[1], array[0], concrete_cover, array[2], stirrup_diameter)

    return max_shear_beam - maximum_shear

def constraint_3(array):
    return array[1] - (array[3] * (dg + k2 + array[2])) - 2 * concrete_cover + dg + k2


def round_output_to_good_values(optimal_array):
    array = [0,0,0,0]
    array[0] = math.ceil(optimal_array[0]/10.0 - 0.000001) * 10
    array[1] = math.ceil(optimal_array[1]/10.0 - 0.000001) * 10
    optimal_area = optimal_array[3] * 0.25 * np.pi * optimal_array[2] ** 2
    best_fit = [10, 10, 10000000, 1]

    for extra_width in [0, 10, 20, 30, 40, 50]:
        if best_fit[1] != 10:
            continue

        for i in [12, 16, 20, 25, 32]:
            for j in np.arange(2, 9, 1):
                if (j * (dg + k2 + i)) + 2 * concrete_cover - dg + k2 > (optimal_array[1] + extra_width):
                    continue
                area = j * 0.25 * np.pi * i ** 2
                if (area >= optimal_area - 0.00001) and (area < best_fit[2]):
                    best_fit = [i, j, area, extra_width]


    array[2] = best_fit[0]
    array[3] = best_fit[1]
    array[1] = array[1] + best_fit[3]
    return array


# constants

alpha = 0.75
beta = 0.39

# material properties
# values in N/mm2

steel_fyk = 500         #characteristic yield strength of reinforcement steel B500\
concrete_fck = 30       #characteristic compressive yield strength of concrete C30
concrete_fctm = 2.9     #concrete tensile strength

# factors:

gamma_s = 1.15      # material factor steel
gamma_c = 1.5       # material factor concrete

# design values in N/mm2:

steel_fyd = steel_fyk / gamma_s
concrete_fcd = concrete_fck / gamma_c
steel_fywd = 435        # design yield stress of shear reinforcement

# variables with no proper source yet:

theta = 30              # mm
stirrup_diameter = 8    # mm
concrete_cover = 35     # mm
dg = 30                 # mm (maximum grain size)
k2 = 5                  # mm

# Costs of concrete and steel

concrete_cost = 130
```

```
# specify loads and supports here


support_left = support('hinge', 0)
support_right = support('moveable', 7000)
beam_length = support_right.location - support_left.location

loads = []

#load_1 = load('load 1', 'point', 1000, 50000)
#loads.append(load_1)

load_2 = load('load 2', 'distributed', 1000, 5)
loads.append(load_2)

#load_3 = load('load 3', 'point', 2000, 50000)
#loads.append(load_3)

for i in range(len(loads)):
    lines = create_line(loads[i], support_left, support_right)

    if i == 0:
        moment_line = lines[1]
        shear_line = lines[0]
    if i > 0:
        for j in range(len(moment_line)):
            moment_line[j] = moment_line[j] + lines[1][j]
            shear_line[j] = shear_line[j] + lines[0][j]

bound_height_low = max(beam_length / 15 * 0.5, 200)
bound_height_high = max(beam_length / 10 * 1.5, 220)
bound_width_low = max(200, bound_height_low / 3)
bound_width_high = max(bound_height_high / 2, 220)

start = time.time()

constraints = ({'type':'ineq', 'fun':constraint_1}, {'type':'ineq', 'fun':constraint_2}, {'type':'ineq', 'fun':constraint_3})

optimal = minimize(mainfunction, x0 = [max(beam_length / 15, 200), max(beam_length / 50, 200), 16, 3], bounds=[(bound_height_low, bound_height_high), (bound_width_low, bound_width_high), (12, 32), (2, 10)], constraints= constraints)
optimal_array = optimal.x
print(optimal)
new_optimal_array = round_output_to_good_values(optimal_array)


end = time.time()
print('calculation time:', end - start)




self_weight = load('self weight', 'distributed', 1, new_optimal_array[0] * new_optimal_array[1] * 0.000025 * 1.2)
lines_weight = create_line(self_weight, support_left, support_right)
moment_line_temp = moment_line.copy()
shear_line_temp = shear_line.copy()

for j in range(len(lines_weight[0])):
    moment_line_temp[j] = moment_line_temp[j] + lines_weight[1][j]
    shear_line_temp[j] = shear_line_temp[j] + lines_weight[0][j]

maximum_shear = max(abs(shear_line_temp))
spacing = determine_spacing_stirrups(new_optimal_array[0], concrete_cover, new_optimal_array[2], stirrup_diameter, maximum_shear, theta, steel_fywd)
number_of_stirrups = round(beam_length / spacing)

cost = costs(new_optimal_array[0], new_optimal_array[1], beam_length, concrete_cost, new_optimal_array[2], new_optimal_array[3], number_of_stirrups, stirrup_diameter, concrete_cover)

print('beam height:',new_optimal_array[0])
print('beam width:',new_optimal_array[1])
print('diameter:',new_optimal_array[2])
print('number of cables:',new_optimal_array[3])
print('spacing:', spacing)
print('costs:', cost)

```

F – Results of the tests

L =	3	4	5	6	7	8	9	10	11	12	13	14	15
q = 5 kN/m													
Model 1													
time	0,374	1,157	3,258	6,489	11,043	17,130	24,477	38,724	48,578	60,291	77,417	97,483	119,842
height	200	200	280	240	320	290	360	440	370	440	520	620	500
width	200	200	200	200	200	200	200	200	210	210	210	210	220
diameter	12	12	12	16	16	20	20	20	25	25	25	25	32
number of bars	2	2	2	2	2	2	2	2	2	2	2	2	2
spacing	300	300	300	300	300	300	300	300	300	300	300	300	300
cost	32,41	43,01	67,84	89,53	123,13	161,61	202,85	252,11	319,20	377,40	444,99	529,53	661,96
Model 2													
time	0,910	0,861	1,024	2,765	12,783	4,720	5,287	4,009	4,952	5,785	11,780	12,563	7,740
height	200	200	210	250	280	320	360	390	430	470	500	540	580
width	200	200	200	200	200	200	200	200	200	200	200	200	200
diameter	12	12	12	16	16	16	20	25	25	25	32	32	32
number of bars	2	2	3	2	3	3	2	2	2	2	2	2	2
spacing	300	300	300	300	300	300	300	300	300	300	300	300	300
cost	32,41	43,01	64,80	91,56	135,76	166,61	202,85	290,69	335,45	381,79	554,30	616,87	680,76
cost continuous var.	32,41	43,01	62,82	88,85	120,06	157,47	199,6	247,48	302,48	362,634	429,23	503,93	584,22
q = 10 kN/m													
Model 1													
time	0,363	1,109	2,905	5,885	9,646	15,801	22,119	30,783	41,94	53,48	69,016	88,561	109,2
height	200	210	290	290	370	470	410	500	600	490	560	640	780
width	200	200	200	200	200	200	210	210	210	220	220	220	260
diameter	12	16	16	20	20	20	25	25	25	32	32	32	32
number of bars	2	2	2	2	2	2	2	2	2	2	2	2	2
spacing	300	300	300	300	300	300	300	300	300	300	300	300	300
cost	32,41	55,4	83,35	121	159,81	210,56	273,55	335,17	408,42	525,19	601,66	689,92	880,8
Model 2													
time	0,727	1,787	1,766	4,93	3,572	5,5	5,872	3,408	5,501	6,338	7,065	13,787	8,518
height	200	230	280	330	370	420	470	520	570	620	670	720	770
width	200	200	200	200	200	200	200	200	200	200	200	200	200
diameter	12	12	16	16	20	25	25	25	32	32	32	32	32
number of bars	2	3	3	3	2	2	2	2	2	2	2	2	2
spacing	300	300	300	300	300	300	300	300	300	300	300	300	300
cost	32,41	54,15	97,43	126,77	159,81	241,35	286,34	334,65	495,99	560,88	629,07	702,42	777,35
cost continuous var.	32,41	53,02	81,69	116,11	157,08	205,94	260,77	322,81	393,75	470,98	556,09	651,14	752,81

L =	3	4	5	6	7	8	9	10	11	12	13	14	15
q = 20 kN/m													
Model 1													
time	0,314	1,182	3,157	6,493	10,774	16,885	25,655	33,888	45,131	59,424	76,394	96,547	124,194
height	220	330	350	470	440	550	700	570	670	810	680	780	910
width	200	200	200	200	210	210	230	220	220	270	290	290	300
diameter	16	16	20	20	25	25	25	32	32	32	32	32	32
number of bars	2	2	2	2	2	2	2	2	2	2	3	3	3
spacing	300	300	300	300	300	300	300	293,2	300	300	251,3	262,1	274,4
cost	42,73	71,58	111,32	157,6	219,75	285,06	383,2	467,88	554,27	733,23	927,89	1058,7	1239,88
Model 2													
time	2,091	2,607	2,473	3,456	4,388	3,56	3,987	4,841	5,374	5,269	5,422	5,278	5,906
height	240	300	370	430	500	560	630	700	760	830	900	960	1030
width	200	200	200	200	200	200	200	200	200	200	250	250	250
diameter	12	16	16	25	25	25	32	32	32	32	32	32	32
number of bars	3	3	3	2	2	2	2	2	2	2	3	3	3
spacing	300	300	300	300	300	300	300	300	300	300	300	300	300
cost	41,8	80,16	112,76	182,76	229,39	279,42	423,71	493,91	566,98	646,28	975,51	1086,18	1205,27
cost continuous var.	40,911	70,21	108,83	155,05	209,94	275,18	348,2	430,54	524,33	626,09	737,82	862,14	994,63
q = 30 kN/m													
Model 1													
time	0,328	1,215	3,256	6,289	10,593	16,625	23,628	34,872	47,081	61,99	78,97	98,14	123,327
height	280	330	470	460	600	530	640	780	670	790	930	850	980
width	200	200	200	210	210	220	220	260	290	290	310	360	360
diameter	16	20	20	25	25	32	32	32	32	32	32	32	32
number of bars	2	2	2	2	2	2	2	2	3	3	3	4	4
spacing	300	295,2	300	279,4	300	239,63	257,17	272,42	211,37	225,24	236,35	196,61	206,27
cost	48,83	86,7	131,75	194,18	259,04	370,95	451,68	595,01	795,48	929,91	1122,91	1443,87	1652,95
Model 2													
time	3,2	4,145	3,746	2,424	2,562	3,301	6,033	7,415	7,66	9,124	7,422	11,276	4,382
height	280	360	440	510	590	670	750	830	910	990	1070	1240	1440
width	200	200	200	200	200	200	200	200	250	250	250	250	250
diameter	16	16	25	25	32	32	32	32	32	32	32	32	32
number of bars	2	3	2	2	2	2	2	2	3	3	3	3	3
spacing	300	300	300	300	300	300	300	300	286,43	282,64	278,84	292,65	300
cost	48,83	88,25	154,39	199	319,55	388,05	460,31	537,87	833,77	950,04	1075,82	1247,94	1453,68
cost continuous var.	48,36	83,52	129,72	185,05	250,67	328,54	415,59	513,59	625,02	748,21	883,36	1029,14	1201,14