THERMAL AND STRUCTURAL STUD WALL DESIGN OPTIMIZATION IN EXCEL USING GENETIC ALGORITHMS

Alexander C. Schreyer
Building Materials and Wood Technology, University of Massachusetts, Amherst

ABSTRACT

This paper presents a method to define the optimal wood stud wall arrangement with respect to structural and thermal performance criteria. Design variables are the stud dimensions w and d and the stud spacing e.

To solve the optimization task (minimize the cost of the wall), an optimization routine based on genetic algorithms (GAs) was implemented into a Windows-based software. This software is capable of performing general GA optimization within an Excel spreadsheet.

Two calibration cases – an unconstrained and a constrained function – were run using the software. It was found to perform well and GA parameters were optimized in a parametric study.

Using the parameters found in the parametric study, the stud wall optimization was performed and compared to a gradient-based optimization method. While the accuracy of the results was within close proximity, the GA algorithm used far more function calls than the "classic method".

INTRODUCTION

In the US, most residential houses are build using light wood framing ("stud framing", "2x4 construction", "stick-built", see Figure 1). This technology originated from the European tradition of half-timbered houses, which was brought to America by the first settlers. It is highly efficient since it uses small-dimension structural members and sheathing, which is fastened to the framing using light-gauge nails. While the structural members provide bending resistance and axial (compression) support, the sheathing provides racking strength, diaphragm action and continuous support against buckling. In the case of the interior-side gypsum wallboard (GWB), it also provides a finishable surface. A major benefit of this type of construction is that in its basic form ("platform framing") it does not need heavy machinery and can be built using a small number of workers.

Figure 1 - Light wood-framing construction site

This paper seeks to define the "ideal" stud size and on-center spacing given a set of realistic parameters together with constraints based on the structural and thermal performance of the wall. The basis of this hypothetical investigation is the difference between structural member and wall dimensions between the US and Europe. In the US it is common to build with 2x… sawn lumber material, which has an actual width of 1.5 in. and depths of 1.5, 3.5, 5.5, 7.25, 9.25 and 11.25 in. Wall studs are typically 2x4s or 2x6s and on-center spacing of these studs in walls is typically 16 in. In Germany, construction lumber is sold in multiples of 20 mm and the typical stud size is 60 mm x 120 mm. While the width is again held constant, the thickness can be increased up to 200 mm. On-center spacing is usually 625 mm.

To reduce cut-offs of panel material, on-center spacing is in both cases set to an even fraction of the panel widths. In the US, the standard panel size is 4 ft x 8 ft and 16 in. is thus 1/3rd of the panel width. In Germany, the standard panel width is 1250 mm and 625 mm is half of that dimension.

Although an adherence to standard dimensions typically proves the most cost-effective route, newer research suggests, that larger panel dimensions can significantly increase racking strength [1]. Since most panels nowadays are made of Oriented Strand Board (OSB), which can be produced in any length, a non-traditional optimization of the on-center spacing might prove cost-effective in those cases. Similarly, the development of new structural wood-based composites may at some point make it economically viable to produce them to order instead of a standard size.

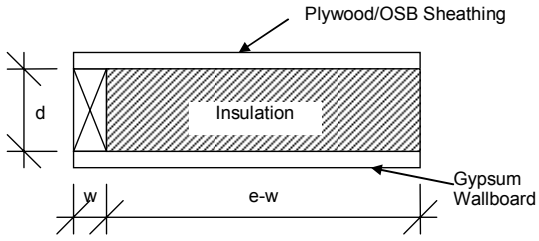
Problem Definition

The optimization problem introduced above was implemented as a minimization of the cost of a 1000 mm section of a stud wall. All parameters, design variables, the objective function and all constraints have been inserted into an Excel spreadsheet for ready processing. The problem was defined as follows:

Design variables:

$$\mathbf{X} = \begin{bmatrix} w & d & e \end{bmatrix}^T$$

where:



Minimze:

$$C(\mathbf{X}) = C_{StudPlates} + C_{PlySheath} + C_{GWBSheath} + C_{Insulation}$$

where: C = cost of 1000 mm wall section

Such that the following constraints are satisfied:

Structural (stud) constraint [2]:

$$g_1 = \left(\frac{f_c}{F_c}\right)^2 + \frac{f_b}{F_b(1 - f_c/F_{cEx})} - 1 \le 0$$

where: f_c = axial stress, F_c = compressive strength
f_b = bending stress, F_b = bending strength
$F_{cEx} = \dfrac{0.3E}{\lambda^2}$, $\lambda = l/d$ = slenderness

Structural (panel, 100 mm section) constraint:

$$g_2 = \frac{f_b}{F_b} - 1 \le 0$$

where: f_b = bending stress, F_b = bending strength of panel with faces aligned perpendicular to stud

Deflection (panel, 100 mm section) constraint:

$$g_3 = \delta_{Panel} - \delta_{Allowable} = \frac{5qe}{384 E_{Panel} I_{Panel}} - \frac{e}{400} \le 0$$

where: E_{Panel} = MOE of panel

$$I_{Panel} = \frac{100 \cdot t_{Panel}^{\ 3}}{12}$$

Thermal insulation constraint (min. R-value):

$$g_4 = R_{total} - R_{t\arg et} \le 0$$

where:

$$R_{total} = \frac{1}{f_1/R_1 + f_2/R_2}$$

$$R_1 = 0.13 + d_{GWB}/\lambda_{GWB} + d_{wood}/\lambda_{wood} + d_{ply}/\lambda_{ply} + 0.08$$

$$R_2 = 0.13 + d_{GWB}/\lambda_{GWB} + d_{ins}/\lambda_{ins} + d_{ply}/\lambda_{ply} + 0.08$$

$f_1 = w/e$, $f_2 = (e - w)/e$
1 = section through stud
2 = section through insulation

Nail spacing constraint:

$$g_5 = w_{\min} - w \le 0$$

where: $w_{\min} = 4a + w_{gap}$
a = edge distance of nail
w_{gap} = optional gap between plates

Due to the nonlinear nature of some of the constraints, it was intended to optimize this problem using a genetic algorithm (GA). It is also the intention of the author to create a GA software that could later be used for more complex problems. Although architectural optimization problems are linear in most cases, constraints can prove to be nonlinear or even discontinuous. This poses problems for gradient-based methods and may be solved more accurately using genetic algorithms. An example could be an if-then constraint, where an integer number of windows were inserted into a continuous length wall. The following section briefly introduces the concept of GAs.

Genetic Algorithms (GAs)

Genetic algorithms (GAs) are based on biological principles of evolution and provide an interesting alternative to "classic" gradient-based optimization methods. They are particularly useful for highly nonlinear problems and models,

whose computation time is not a primary concern. Similar to other methods such as Simulated Annealing, they perform better than gradient-based methods in finding a global optimum if a problem is highly nonlinear and features multiple local minima. In general, GAs approach the entire design space randomly and then improve the found design points by applying genetics-based principles and probabilistic selection criteria. A thorough description of genetic algorithms can be found in [3].

Although a large number of modified algorithms are available, a GA typically proceeds in the following order:

1. Start with a finite population of randomly chosen chromosomes ("design points") in the design space. This population constitutes the first generation ("iteration"),
2. Evaluate their fitness ("function value"),
3. Rank the chromosomes by their fitness,
4. Apply genetic operators (mating): reproduction (reproduce chromosomes with a high fitness), cross-over (swap parts of two chromosomes, chosen based on their fitness to create their offspring) and mutation (apply a random perturbation to parts of a chromosome). All of these operators are assigned a probability of occurrence,
5. Assemble the new generation from these chromosomes and evaluate their fitness,
6. Apply genetic mating as before and iterate until convergence is achieved or the process is stopped.

As can be seen above, the primary usefulness of the GA is that it starts by sampling the entire design space, possibly enabling it to pick points close to a global optimum. It then proceeds to apply changes to the ranked individual design points, which leads to an improvement of the population fitness from one generation to another. To ensure that it doesn't converge on an inferior point, mutation is randomly applied, which perturbates design points and allows for the evaluation and incorporation of remote points.

The main advantages of GAs are:

- The nature of the optimization model does not need to be known. This makes GAs very interesting for complex problems or for users inexperienced in gradient-based optimization techniques.
- The optimization model and its constraints do not have to be continuous or even real values. No simplification of a problem is necessary to accommodate it to a particular algorithm (e.g. linearization).
- They are readily available and easily implemented.

The main disadvantages are:

- A large number of parameters need to be set. This is simplified by information from literature, but problem-specific adjustments might need to be made.
- Due to the comparatively very large number of function calls, GAs require significant computational resources. This makes them unattractive for optimization problems with computationally demanding analyses.

MODEL AND SOFTWARE DEVELOPMENT

Several software implementations of GAs are currently available as open-source on various websites. An example can be found in [4]. While most of them are available as C++ code, it was intended to write the GA Optimization for Excel software in Delphi, an RAD (Rapid Application Development) Object-Pascal environment for the Windows platform. Although C++, FORTRAN or any other code can be compiled into a DLL and then used by any software development environment, an existing GA implementation in Delphi was chosen for this project.

The GA code written by Soft Tech Design, Inc. [5] is a very well programmed Pascal implementation into an object model. It features:

- 2 chromosome types: string and float
- 3 GA objects: string (for permutations), float (for real calculations) and sequence list (e.g. for the traveling salesman problem)
- 3 cross-over types: one point, two point and uniform ($0.5 < p < 0.8$)
- Elitism (the two chromosomes with the highest fitness automatically advance to the next generation)
- Chance for random selection (not based on fitness)
- Mutation of lowest fit chromosomes
- Option for a number of preliminary runs that select the fittest chromosomes from the preliminary generations and then use them as starting generation for the "real" runs.

Since the given implementation as an Excel optimization program solely works with real values, only the float GA object was implemented here.

In addition to the above, several functions were added by the author in developing the GA Optimization for Excel software:

- **Limits on the design variables:** All design variables can have lower and upper limits. If during the GA process, a gene reaches or overshoots any of these limits its value is set to the limit itself. This effectively provides side constraints for all variables.
- **Termination criterion:** After genetic mating, an average deviation of the individual chromosomes from the population average is calculated. This is then compared to a predefined convergence tolerance value and further computations are stopped if the criterion is fulfilled as follows:
$$\frac{\sum \left| F_{avg} - F_i \right|}{n_{chromosomes}} \leq \varepsilon_{tol}$$
- **Minimization and target value option:** Since GAs look for the individual chromosome (design point)

with the highest fitness, they effectively are maximization routines. To also allow for minimization, a simple modification was added where:

$$Minimize\ F = Maximize\ (-F)$$

Consequently, driving the result to a target value requires:

$$For\ F = Target\ we\ Minimize\ |F - Target|$$

Both options were added to the maximization routine.

- **Constraints:** The option for constraints on the optimization model was added by including an "absolute penalty method". This was done in the subroutine (object method) that calculates the fitness of the individual chromosome.
  By applying an absolute penalty, the fitness is set to a very low value (e.g. -10,000) when the chromosome (design point) violates a constraint. This results in the chromosome being ranked at a very low fitness and thus being ultimately "outgrown" by fitter individuals.
  The benefit of this method is that it does not present a potential variable overflow as can occur with scaled methods (e.g. exterior penalty function) since the penalty value and thus the adjusted fitness value is a constant number. On the other hand, the disadvantage is that this method does not provide a scaled fitness where some only slightly violating chromosomes could be ranked more favorably than others.
- **Excel connectivity:** Design values (gene values) are given to Excel and function values (fitness values) and constraint values are read from Excel through the Windows COM (Component Object Model) interface. This allows for an easy data transfer, which is direct (it does not need text files), immediate and object-based, since the Excel application object is created and called directly from within the Delphi application.
- **GUI (user interface):** A Windows user interface was created that allows the user to use the GA model without prior knowledge. As can be seen in Figure 2 and Figure 3, an Excel file, which contains the calculation model, can be selected and cell references for the function value, all design variables and all constraints can be specified. On another tab, the user can modify the given GA parameters and then on a third tab, the user can run the GA algorithm and capture its output. Optionally, the user can save and restore all GA and model parameters in a text file and restore them from there later.

Figure 2 and Figure 3 are images of the user interface. The software can be downloaded for free from the author's website: http://www.alexschreyer.de/projects/xloptim/

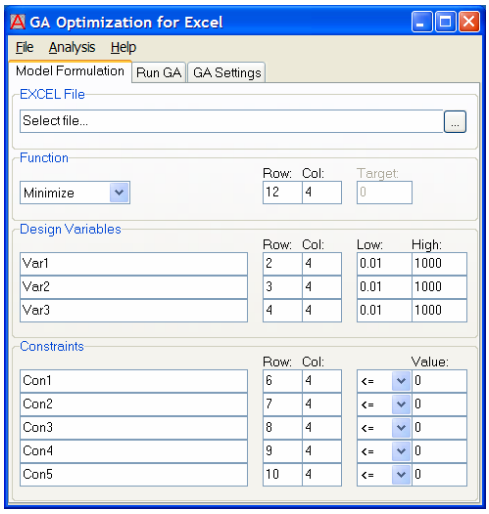


Figure 2 - GA Optimization for Excel user interface

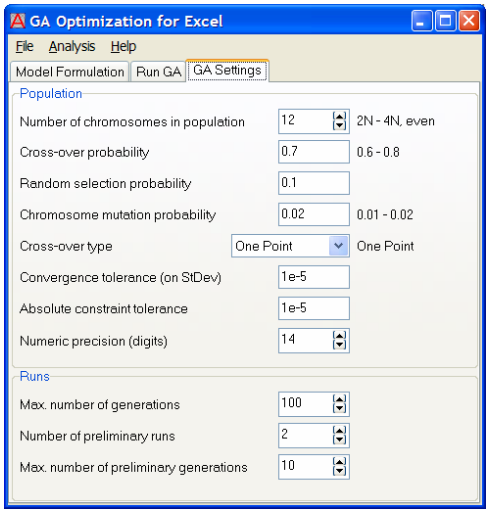


Figure 3 - GA settings tab

GA MODEL VERIFICATION – UNCONSTRAINED

To test the optimization routine and debug the software, a function was chosen that featured a global maximum and several local maxima. It was taken from literature [6] and is defined as:

$$F(X_1, X_2) = \cos^2(n \cdot \pi \cdot r) \cdot e^{(-r^2/\sigma^2)}$$

where:
$$r^2 = (0.5 - X_1)^2 + (0.5 - X_2)^2$$
$$n = 9$$
$$\sigma^2 = 0.15$$

The function was implemented into the maximization routine as:

$$\textit{Maximize} \quad F(X_1, X_2)$$

$$\text{s.t.} \qquad 0 \le X_1, X_2 \le 1 \quad \text{(side constraints)}$$

This function is ideally suited to test a global maximization routine since it features several high and steep "ripples" in the vicinity of the optimal solution (which is at $X_1 = 0.5$ and $X_2 = 0.5$ with a function value of $F = 1.0$). It can be expected that if a gradient-based optimization routine were started from an initial point not on the hill closest to the optimum, it would likely fail to reach the optimum since it could not escape from the surrounding ripples. Figure 4 illustrates this function in the X_1, X_2-space.

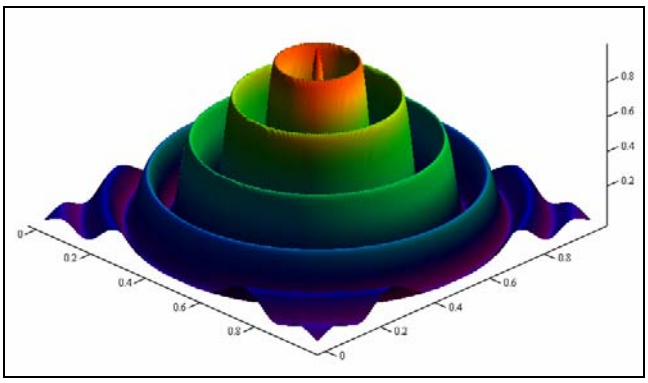


Figure 4 - Function plot

To comparatively test this function using a gradient-based optimization routine, it was implemented into an Excel spreadsheet and maximized using the Excel Solver add-on. The Solver features a modified Generalized Reduced Gradient (GRG2) algorithm.

After several runs, it became obvious that any starting point within the design space led to convergence on a local maximum (one of the ripples) instead of the global maximum (the peak). This is clearly explained by the gradient-search methodology and the inability to escape local minima once they are encountered. It should be noted, though, that when the extreme points of the design space (0,0 – 0,1 – 1,1 – 1,0 or their close vicinity) were chosen as starting points, Solver managed to converge quickly on the optimal solution (0.5,0.5). This can be explained by the gradient at those locations, which points directly towards the maximum (see Figure 4).

Using the GA Optimization for Excel software, the same spreadsheet was used to test the effect of the GA parameters on convergence behavior. In theory, the GA optimization should be able to find the global maximum. Nevertheless, the nature of the function does not provide much space around the maximum and since the GA is based on an initial set of random points, the chance of starting with a point within this region is quite small. The region around the peak can be quantified as:

$$A = \frac{\pi}{4n^2}$$

To test the performance of the GA optimization routine, all settings parameters (see Figure 3) were varied independently from an initial set of well performing parameters (8, 0.8, 0.1, 0.01, One Point, 1e-5, 1e-5, 12, 100, 2, 10 – read in top to bottom order with respect to Figure 3). The performance was then evaluated by the following error term and averaged over five runs:

$$\varepsilon = \left| \frac{F(X_1, X_2) - 1}{1} \right|$$

The following conclusions were drawn from the parametric study:

- Increasing the **number of chromosomes** per population directly increased the accuracy of the solution. This was most pronounced when the number changed from 8 to 16 to 50. Changing it to 100 did not yield a significant improvement. Evidently, a higher number of chromosomes provides a higher chance of starting closer to the optimum. Since the number of chromosomes generally determines the number of function calls, it should be kept to a minimum.
- Increasing the **cross-over probability** from 0.6 to 0.8 to 0.9 also yielded an improved performance. A cross-over probability of 1, however, decreased the accuracy significantly. Also, changing the **cross-over method** from One Point to Two Points to Uniform did not change the error much. The best performance was here observed with the "classic" one-point cross-over.
- The two "insurance" parameters, **probability of random selection** and **mutation probability** were both found to be well performing at the 0.1 level. Although this is quite high compared to the recommended values for the mutation probability (0.01 – 0.02), the nature of the test function and the small target (peak) area explain the necessity of having these parameters at high levels. When either parameter was chosen at too high levels (0.5 for example), it was observed that they led to an instable convergence and necessitated a significantly higher number of generations before convergence.
- The effect of running **preliminary generations** and selecting the best performing individuals for the starting generation of the actual runs was found to be significant. Comparing only the number of preliminary runs, a number of 2 showed the lowest and 8 the best performance – with 8 preliminary runs showing a similar performance as no

preliminary runs at all. In that case, though, the numbers of necessary generations were 20.6 and 11.4, respectively. As could be expected, the preliminary runs provide a better starting pool and thus reduce the number of actual generations before convergence is achieved. When the number of generations per preliminary run was set to a higher value (50), performance was increased. Similar to the total number of generations, however, the preliminary run generations also contribute to the total number of function calls and should be kept to a minimum.

- Changing the **convergence tolerance** from 10^{-5} to 10^{-12} did not yield an improvement but rather increased the number of generations. Comparing **numeric precision**, it was found that the performance actually was highest at the 3-digits and approximately similar at the 6-digits and 12-digits level.

This parametric study resulted in a set of individually well performing parameters (32, 0.9, 0.1, 0.1, One Point, 1e-5, 1e-5, 12, 100, 8, 20). These, however, tended to yield a large number of function calls, so they were modified to the following parameters: 16, 0.9, 0.1, 0.1, One Point, 1e-5, 1e-5, 6, 100, 4, 10, which resulted in small error values ($\varepsilon = 2.24\%$ after 14.6 generations and 873.6 function calls; all averaged over five runs). Figure 5 illustrates a sample run with these parameters.

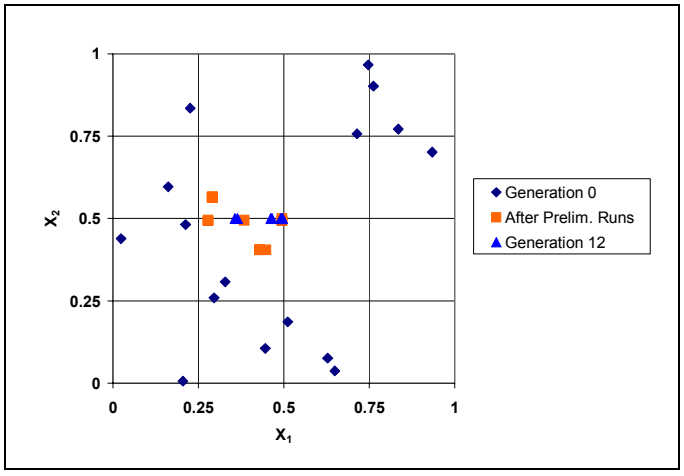


Figure 5 – Example of an unconstrained parametric study run

GA MODEL VERIFICATION – CONSTRAINED

Similar to the unconstrained model verification, a function was chosen that featured two design variables and several constraints (two in this case). The minimization case was again taken from literature [7] and is of the following form:

Minimize $F(X_1, X_2) = X_1 + X_2$

s.t. $g_1(X_1, X_2) = -2 + X_1 - 2X_2 \le 0$

$g_2(X_1, X_2) = 8 - 6X_1 + X_1^2 - X_2 \le 0$

$-10 \le X_1, X_2 \le 10$ (side constraints)

Although this function was not as severe by nature as the one chosen above, it allowed testing the parameters that were related to constraints. Figure 6 illustrates the constraint functions g_1 and g_2.

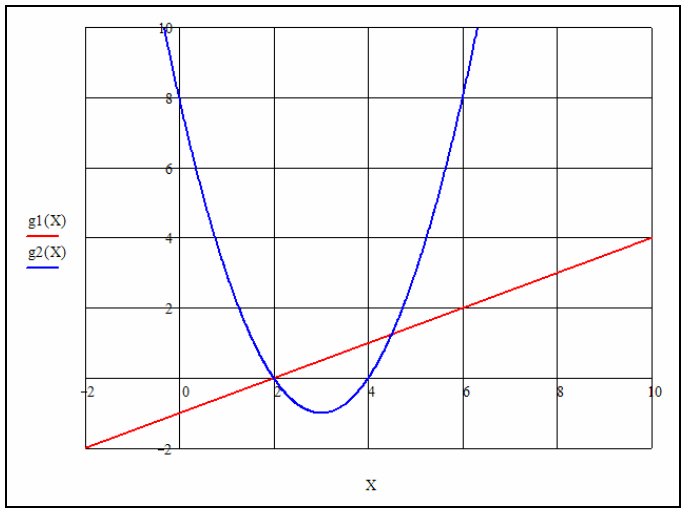


Figure 6 - Constraint functions

A limited parametric study was performed on the minimization problem. Initially, the best performing parameters from the unconstrained parametric study were chosen and then individually modified. The application of constraint penalties followed the previously described absolute penalty method.

The following conclusions could be drawn from this study:

- Due to the nature of the elimination system implemented by the absolute penalty method, a strong influence of the **number of preliminary runs** on the final accuracy was found. When no preliminary runs were specified, the error of the final result was 103.3%. This reduced to 21.5% and 3.95% when 4 and 8 preliminary runs, respectively, were executed. This clearly shows that the larger number of preliminary runs eliminates many individuals with constraint violations and provides a population for the first generation that is well within the feasible design space.
- When no preliminary runs were specified, the **number of chromosomes** per generation became highly important. Since the initial generation provides the base for all subsequent ones, a first generation of all "constraint violators" will provide a full generation with the same function value (i.e. the penalty value). When **bounds** are not too wide

and the number of chromosomes is not too small, this should not become an issue. One case was even observed where all but one individual of the initial generation had violated constraints, yet the final error value (after 39 generations) was only 2%.

- Since the GA essentially works with random numbers (of a certain precision), it is rather impossible to hit a target value (e.g. 0) with a high precision and without rounding. As a result, a **constraint tolerance** was implemented into the software that makes runs possible, where we want to use equality constraints. Several runs with constraint tolerances of 10^{-5} and 10^{-12} together with the given minimization problem (which features very large bounds) showed a high decrease in accuracy. It is thus recommended to only use the tolerance value where equality constraints are employed.

Combining the results from both parametric studies, the best performing settings were: 16, 0.9, 0.1, 0.1, One Point, 1e-5, 0, 6, 100, 8, 10 (read in top to bottom order with respect to Figure 3).

APPLICATION OF GA FOR DESIGN OF A STUD WALL

The optimization of the stud wall dimensions was performed using the values given in Table 1. Properties given in Table 1 were taken from pertinent literature and represent typical values. The wall was assumed to be located on the first floor under two upper stories and assumed to be an exterior wall. Cost values were averaged and include labor.

Height of wall =	2438	mm
Line load on wall =	20.04	N/mm
Wind pressure on wall =	0.00065	N/mm^2
Wood compression strength =	12.9	N/mm^2
Wood bending strength =	14.8	N/mm^2
Wood MOE =	11000	N/mm^2
Panel bending strength =	15	N/mm^2
Panel MOE =	6000	N/mm^2
Nail d =	3.7	mm
Edge distance (x*d) =	2.5	
Gap between sheathing =	0	mm
Cost studs & plates =	1.54E-06	$\$/mm^3$
Cost plywood =	1.53E-05	$\$/mm^2$
Cost GWB =	1.24E-05	$\$/mm^2$
Cost insulation =	5.47E-08	$\$/mm^3$
Insulation heat transmission =	0.05	W/(m*K)
Stud heat transmission =	0.13	W/(m*K)
GWB heat transmission =	0.25	W/(m*K)
OSB/ply. heat transmission =	0.15	W/(m*K)
Min R-Value for wall =	3.0	$(m^2*K)/W$

Table 1 - Parameters for stud wall

In a first set of runs, the parameters found in the previous chapter were kept and the stud wall model was run with the constraints given. Five runs were performed and averaged to determine the differences. As can be seen in Table 2, all values of interest averaged within close range. The total cost averaged at \$ 109.23 per 1 meter wall with a standard deviation of only \$2.03. The largest differences were in the depth of the stud and the stud spacing, i.e. the insulated space.

	Average	**St. Dev.**
w =	38.5	1.78
d =	151.9	5.10
e =	988.2	18.47
Actual R =	3.1	0.09
Total cost =	\$109.23	2.03
Criterion efficiency:		
Stud combined strength	36.90%	0.05
Panel bending strength	8.75%	0.00
Panel deflection	94.59%	0.05
Difference to min. geometry	4.11%	0.05
Thermal insulation target	103.57%	0.03
Number of generations	59.8	30.62

Table 2 - Runs with 16 chromosomes

It was found that the optimization routine always reduced the stud width to the minimum (w_{min} = 37 mm). This is only prudent from a structural perspective. Also, since the thermal conductivity through the stud is quite a bit higher than through the insulated space, a reduction increases the overall thermal resistance.

The depth d was also within close tolerances since it determines the thickness of the insulation. Its variability was higher than w mainly because d and e provided flexibility (when one increased, the other one decreased).

It is interesting to note that at the R = 3.0 insulation level, the stud strength constraint was not active and stud strength was only efficient at ~40%. With this high insulation value, the thermal insulation target constraint was fully active. Since panel deflection determined the maximum e, this constraint was also almost active.

	Average	**St. Dev.**
w =	37.5	0.51
d =	148.5	2.46
e =	982.6	8.63
Actual R =	3.0	0.05
Total cost =	\$107.82	0.90
Criterion efficiency:		
Stud combined strength	40.40%	0.03
Panel bending strength	8.65%	0.00

Panel deflection	92.95%	0.02
Difference to min. geometry	1.26%	0.01
Thermal insulation target	101.53%	0.02
Number of generations	87.2	28.62

Table 3 - Runs with 32 chromosomes

A comparative set of runs was performed with 32 chromosomes and all other GA parameters as before. This improved the accuracy of all results, but in many cases, convergence was not achieved before the 100 generation limit. Average results from five of these runs are shown in Table 3.

VERIFICATION OF RESULTS

Due to the closed-form nature of this problem, it was possible to comparatively solve the optimization using Excel's Solver add-on. Table 4 shows the results of that calculation.

	Result
w =	37.0
d =	145.8
e =	1007.0
Actual R =	3.0
Total cost =	$106.34
Criterion efficiency:	
Stud combined strength	45.12%
Panel bending strength	9.08%
Panel deflection	100.00%
Difference to min. geometry	0.00%
Thermal insulation target	100.00%

Table 4 - Solver results

While the GA calculations took 4480 generations at the maximum to reach the solution shown in Table 3 (32 chromosomes * 100 generations + 4 preliminary runs * 32 chrom. * 10 preliminary generations), only 10 iterations were necessary for Solver to arrive at its solution. Although this clearly shows the efficiency of a gradient-based solver over a GA solver for this problem, the applicability of the GA software has been proven.

SUMMARY

This project successfully created a GA optimization software that optimizes problems, which reside on Excel spreadsheets. Parameter tests have been performed and the model was calibrated to a high degree of efficiency. Ultimately, it was applied to a stud-wall design, which minimized the cross-sectional properties and maximized the thermal resistance while reducing the cost of the wall as much as possible.

REFERENCES

[1] Durham, J., Lam, F., Prion, H.G.L. (2001) "Seismic Resistance of Wood Shear Walls with Large OSB Panels". Journal of Structural Engineering, ASCE, 127 (12)

[2] AF&PA (2001). *National Design Standard for Wood Construction*. American Forests & Paper Association, Washington, DC

[3] Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley

[4] Anonymous (2005). "GAlib, A C++ Library of Genetic Algorithm Components". At: http://lancet.mit.edu/ga/

[5] Soft Tech (2002). "GA Class Library". At: http://www.softtechdesign.com/products/GA_Delphi/GeneticAlgorithm.htm

[6] Charbonneau, P., Knapp, B. (1995). "Users guide to PIKAIA 1.0". NCAR Technical note TN-418-IA

[7] Vanderplaats, G.N. (2001). *Numerical Optimization Techniques For Engineering Design*. 3rd Edition, VR&D, Colorado Springs, CO