SIMULATION, MATHEMATICAL MODELING AND OPTIMIZATION IN INDUSTRIAL DESIGN: WHEN AND HOW TO APPLY IT?

R. F. Garzia, P. H. Schmidt, and M. R. Garzia
*The Babcock & Wilcox Co.
Barbenton, Ohio 44203
† The University of Akron
Akron, Ohio 44325
USA

ABSTRACT: With the current technology, optimization has become a feasible and profitable avenue for problem solving in industrial design. This in turn brings simulation and modeling into the design process in order to obtain a mathematical description of the process in question. This paper addresses the problem of "When and How to Apply Optimization in Industrial Design?". In addition, it reviews the most common optimization algorithms and how they should be selected. Furthermore, some of the common optimization packages are discussed. Four industrial applications are briefly studied. The first two are in the area of simulation, in particular, the simulation of a foundry and a computer workload simulation. The other two examples deal with optimization. The first of which develops an algorithm for economic tube grouping and the second one discusses the optimization of a boiler's circulation system.

1. INTRODUCTION

In the last ten years, the developments in computer technology have dramatically increased the areas for computer applications. The net effect is at both ends of the computer scale, namely, expensive computer systems (large computer systems) and inexpensive microprocessors. At present, we have at our disposal large computers, using virtual memory concepts, sophisticated operating systems, running at high speed, etc. These computers allow the processing of complex engineering problems in only a few seconds of CPU time. On the other hand, we also have microprocessors that can handle on-line sophisticated control algorithms of adaptive nature.

In addition to the developments in computer technology, the mathematical progress in recent years in the areas of numerical analysis, optimization algorithms, system identification, etc., have opened new avenues of application to the engineering community. These application fields have practically no limits.

Therefore, applications of optimization in industrial design becomes feasible and profitable (Garzia 1977c). This in turn brings simulation and modeling into the design process in order to obtain a mathematical description of the problem suitable for optimization.

In reading recent technical papers, it is possible to see the growing importance of mathematical modeling and analysis to design (Ansari 1975, Naharandi 1976). Additionally, the cost of producing the components is now a factor almost always involved in the design process (Dehne 1972, Naharandi 1976).

For a complete understanding of what "Optimization in Industrial Design" means, as used in this paper, this section will lead to a basic definition. In general, an engineering program operating with a given input and executing predefined calculations and steps produce a set of results. The engineer evaluates these results, and if they are satisfactory, the running is ended, otherwise the necessary changes in the input data are made and the program runs again. Figure 1(a) illustrates the process.

We see that the engineer closes the loop of the iterative process. This is the most important part of the loop. The engineer's evaluation is guided mainly by past experience.
Fig. 1 - Flow Charts of Engineering Programs Without and With Cost Evaluation.

Some conclusions can be drawn from this process. First of all, there is an evaluation of the results. The base of this evaluation could be:

- Reduce the Fabrication Cost - Although the fabrication cost is not calculated by the program, the engineer guided by past experience can manipulate the input values in trying to reduce the fabrication cost of the system obtained. With the present high and still rising cost of materials, the net impact of the past experience is diminished. In any event, in a complex system it is not possible to reach the most economical configuration using this process.
- Improve Performance - Since there is no Fabrication Cost calculated, in general, the changes in cost for performance improvement are unknown.
- Evaluation of other Feasible Alternatives - Again the Fabrication Cost is an important factor that is missed in the results.

By now, it should be apparent that cost estimation is an important component of the engineering calculations. Figure 1(b) illustrates the inclusion of cost estimation. With this arrangement, the engineer has complete control as far as the cost is concerned and can evaluate the most economical configuration of all the ones tested.

To obtain the configuration's minimum cost is almost impossible using the approach described above. To improve the previous process and to be able to obtain the configuration of minimum cost, the process shown in Figure 2 could be used. First, the engineer, during the design of the program or as an input value, needs to define the range of permissible values for some of the problem parameters. Then, during the use of the program, he inputs the remaining parameter values and the program going through the engineering calculations finds the total cost of the system. After the total cost is calculated, the program reviews and changes, if possible, the values of the parameters within the allowable range in such a way that the total cost decreases. This process continues until the total cost obtained is a minimum. This minimum total cost is the optimal cost and the process described, optimization. Therefore, Optimization in Industrial Design can be described as the process of designing a system (mechanical, electrical, etc.) in which a specified performance index is maximized or minimized. In our case, the performance index chosen is total cost, and the process is to minimize.

Let us now try to answer the questions of "when and how to apply it?". In doing so and concentrating on the first one of "when to apply optimization?", let us make the analysis that follows:

When the engineering calculation program is implemented with the estimation program (cost calculations - Fig. 1(b)), we have the capability of performing sensitivity analysis within the range allowed for the problem variables.

Analysis and evaluation of these results will answer the question of whether it is advantageous to use optimization for the problem in question.
Fig. 2 - Optimization Basic Idea

To clear up this point further, let us assume that we have a single variable problem. Let us define the variable in question \( \nu \) and the range given by:

\[
\nu_1 \leq \nu \leq \nu_2
\]  

Figure 3 shows two different types of total cost behavior.

Fig. 3 - Total Cost Versus Variable \( \nu \)

In the case shown by Fig. 3(a), it is worthwhile to use optimization and the most convenient value of \( \nu \) should be \( \nu_{\text{min}} \). On the other hand, the case shown by Fig. 3(b) does not present any advantages for optimization. Since the variation of \( \nu \) does not produce a significant variation of total cost.

After the decision of implementing optimization has been made, we face the question of what approach to use: simulation, mathematical modeling or both. Let us take a closer look at both of them.
2. SIMULATION

The most common way of describing a physical system in the computer is by means of simulation. This involves the process of representing the problem variables by computer defined variables, which are forced to behave in accordance with the physical system. Depending on the type of system and variables to be represented, we can have discrete or continuous simulations.

In using the approach indicated above, we have available a wide variety of languages. These languages range from the general scientific languages such as FORTRAN, APL, etc., to the high level languages such as ACSL (1975), for continuous systems.

ACSL utilizes commands that are first interpreted producing the update of FORTRAN subroutines (translation process). Then, these subroutines are compiled creating the load modules that will run our model. This simulation language has an extensive set of run-time executive commands.

The major disadvantage in using high level languages for simulation is the CPU time required. The major advantage is the ease with which the model can be created.

Within the framework of optimization in industrial design, simulation has two major disadvantages. These are:

- Excessive use of CPU each time the model is run.
- Modifications of the model require a major revision of the simulation program.

On the other hand, simulation offers a safe avenue through which to obtain the characteristics of the physical system in question.

What is the role of simulation within industrial design? As an answer, let us say that simulation is used for the generation of sensitivity analysis. From this analysis, a mathematical model (closed-form solution) is investigated. In some special cases, from the results of the mathematical model, the optimum design can be obtained in a closed-form (Garzia 1970a).

3. MATHEMATICAL MODELING

The results of the sensitivity analysis obtained using simulation, together with the ranges allowed for the problem variables, will permit the investigation and development of a mathematical model that represents the physical system in question. The development of these mathematical models can be as detailed as necessary.

As an example, in the case of the secondary superheaters of a radiant boiler, a second order curve could be used as a mathematical description of the pressure drop as a function of the steam flow for each steam circuit. Some points can be calculated using simulation and then a second order curve fitted to these. To accomplish this task linear programming could be used. The advantage of using LP is that the errors in the neighborhood of the operation point can be constrained.

As a complement to the previous discussion, the development of the mathematical models involve any kind of techniques that can be used in such a process.

The development of mathematical models have the advantages that follows:

- Small CPU time to run the model.
- Better selection of the optimization algorithm.
- Possibility of reduction of the number of problem variables.
- Better understanding of the problem in question.
- Model integration - All the mathematical models representing the components will be integrated as a mathematical model for the whole system.

During the Development of the mathematical models, some local optimization becomes appropriate, i.e., after all the tube thicknesses are calculated, these should be grouped in such a way that the total material cost is a minimum (see Tube Grouping Application).
4. MATHEMATICAL ALGORITHMS FOR OPTIMIZATION

4.1 The Mathematical Model

The general mathematical problem for which optimization algorithms are readily available can be expressed as: Find values of the numbers $x_1, x_2, \ldots, x_n$ so that $\mathbf{x} = (x_1, \ldots, x_n)$ minimizes the value of the object function $f(\mathbf{x})$ while $\mathbf{x}$ satisfies each of the constraints

\begin{align*}
  g_i(\mathbf{x}) &\leq a_i & i = 1, \ldots, m \\
  h_j(\mathbf{x}) &= b_j & j = 1, \ldots, l \\
  a \leq x_i < b & \quad (a \text{ and/or } b \text{ can be infinite}) \\
  a_i \text{ and } b_j & \quad i = 1, \ldots, m, \quad j = 1, \ldots, l \text{ are fixed constants.}
\end{align*}

The problem can be viewed geometrically as finding a low point on a surface described by $f$ while remaining inside a region determined by the $g$'s and $h$'s.

The single mathematical problem can represent many different types of problems. If $n=1$ then we have single variable optimization, if $n \geq 1$ then multiple variable methods need to be used. If there are no constraint functions then we need to consider methods for unconstrained problems whereas the presence of possibly different types of constraints requires other methods.

The type of object and/or constraint functions; whether linear, quadratic, or nonlinear dictates the type of method which can be applied. The ease with which the function values can be obtained and the degree of accuracy which is required will also play a part in the choice of methods.

There are difficult decisions facing anyone who would use optimization methods in industrial design applications. One could code a single method and then modify it for various applications. This would be quite cumbersome and as ill-founded as attempting to utilize a single computer system for all applications. At the other extreme, one could attempt to design a special purpose algorithm for each application. This would obviously be very time consuming and indeed could lead to a general incompatibility of code. A middle course is to find a collection of algorithms along with appropriate software to give the user the flexibility required without introducing significant cost in speed or accuracy. The selection of this collection is based on an assessment of both the algorithms to be included and their implementation in the larger package.

4.2 Requirements for Algorithms

Among the requirements of an individual algorithm are clear statements of the assumptions on which the algorithm is based and an indication of the speed and precision of the method. Each algorithm is designed with a specific class of functions or applications in mind and the user should be able to verify whether or not his problem falls into this class or is in some sense "near" to it. Furthermore, optimization algorithms are iterative in the sense that they construct a sequence of points which hopefully converges to a solution of the problem. The rate of convergence and indeed the existence of the limit itself are theoretical questions, but they should be considered in judging the value of algorithms. The user of an algorithm also needs to know what he is expected to provide; derivative values, an initial guess of the solution, convergence parameters, etc. An indication should be given as to the sensitivity of the solution to this data.

4.3 Algorithms

A consideration of a representative number of algorithms is impossible. There are many excellent textbooks which describe and discuss mathematical algorithms for optimization, among them Wilde and Beightler (1967), Zoutendijk (1976), Bazarra and Shetty (1979); the last contains an excellent bibliography. We do consider some of the building blocks of an optimization package and then discuss a pair of alternatives for full constrained minimization.

The heart of any optimization system no matter what the type of constraint or object function is the one dimensional line search. This is so because many algorithms for multivariable optimization select a direction and then carry out a one-dimensional search in this direction. The model for most good line search algorithms is the assumption that the function is locally close to a cubic or quadratic function depending on whether or not derivative information is available. The methods of Gill and Murray (1973b) yields quick and sufficiently accurate results for reasonable starting values and with only the specification of convergence criteria. To use such an algorithm in conjunction with constraints, one needs to include features to identify and maintain feasibility of the search points.

Methods for unconstrained optimization are important in their own right and as components in constrained minimization methods. For good surveys of the unconstrained methods, see Powell (1976) and Gill and
Murray (1971a). A major factor in choosing a method is the availability or expense of calculation of derivatives. Powell's method and the Nelder-Mead simplex method are direct search methods which require no derivatives. The class of quasi-Newton methods, while considerably faster, do require derivatives.

Another consideration is storage, although this is of less importance, in a virtual memory computer environment. Powell's method requires the storage of a full set of search directions. The Nelder-Mead method requires storage of a set of n+1 vectors. Quasi-Newton methods require an approximation to the Hessian as well as the two most recent gradients. Symmetry of the Hessian matrix reduces the storage requirements of the last method but all three methods need approximately the same amount of storage.

Powell's method (Powell 1964), builds up a set of search directions in which sequential line searches are carried out. The directions are modified automatically so as to ultimately become conjugate, a property, which in the case of a quadratic function would cause an exact solution to be found in a finite number of steps. In comparison with other direct search methods this one seems to be quite fast for even nonquadratic functions. The method requires a reasonable starting value and tolerance parameters, but no derivatives are needed. This last feature is most attractive for industrial applications where derivatives may be available only as difference quotients approximations and where those may be had at great expense.

The Nelder-Mead simplex method was originated by Spendley (1962) and modified by Nelder and Mead (1965). It is a geometrical search method which is easy to understand and to code. The method constructs a sequence of "simplices" of diminishing size whose interiors ultimately contain a minimum of the objective function of N variables. An initial simplex is constructed with N+1 vertices. The function values at all vertices are assumed computed and convergence is declared if the standard deviation of the function values from the mean is sufficiently small. If convergence is not obtained, then the simplex is altered by performing one of the operations of "reflection", "contraction", "expansion", or "shrinkage". These operations along with the criteria for their application are described in Nelder and Mead's paper. As with Powell's method, this method requires no derivatives but numerical experiments such as Himmelblau's (1972) suggest that the Nelder-Mead method is less efficient.

The quasi-Newton methods are similar to their namesake in that they search for a common zero of all the partial derivatives of the object function. This procedure is mathematically sophisticated and computationally quite demanding. The classical Newton method requires the user to compute \( n^2 \) second partial derivatives (to form the Hessian matrix) and then to solve a related system of n linear equations in order to find the step to an improved point. The quasi-Newton method involves approximating this matrix of second derivatives and putting the approximation to its inverse at each step. Various update procedures have been proposed but the procedure due to Broyden (1970), Fletcher (1970), Shanno (1970), and Goldfarb (1970) has been accepted as the most stable and accurate. A new search direction is determined at each step and a one-dimensional line search is carried out in this direction.

The user is required to provide derivative values although this may be carried out by differencing. A significant requirement is that the approximation to the inverse Hessian matrix must remain positive definite so the algorithm must provide a check for this and be able to correct a failure. The method is quite sensitive to the initial guess and to convergence parameters, so care is needed in its implementation. If one can provide derivative values economically and if the object function is reasonably well behaved, e.g. if it is convex, then the steps of the quasi-Newton method will diminish in length geometrically and the limiting point will be a solution to the problem.

For a comparison of effectiveness of several good methods for unconstrained optimization, see Himmelblau (1972).

Constrained problems can occur in many different combinations of object function type, constraint function type, and constraint type; seemingly each type require its own method. For a good survey of the available methods, see e.g. the books by Zoutendijk (1976), Bazaar and Shefety (1979), or the collection of articles in Gill and Murray (1974c). Two popular methods are the general penalty function method and the Generalized Reduced Gradient method.

The penalty function method transforms a constrained optimization problem, e.g.

\[
\min f(x) \quad \text{subject to } g_i(x) \geq 0 \quad i=1,\ldots,k
\]

into a sequence of unconstrained problems whose solutions converge to the solution to the original problem. These unconstrained problems are then solved by an appropriate method. The unconstrained problem can be of the form

\[
\min f(x) + \sum_{i=1}^{k} \phi(g_i(x))
\]
where \( \phi \) is positive and increasing and has the property that \( \phi(t) \to \infty \) as \( t \to 0^+ \) and where \( r > 0 \) is a fixed parameter. Such a function is called a barrier function. Minimization for any positive value of \( r \) has the benefit of guaranteeing feasibility. It can be shown, Mifflin (1972), that under quite reasonable circumstances the solutions to the unconstrained problems converge to the solution to the constrained problem. Unfortunately, this method along with other penalty function methods suffers from the defect that the unconstrained problems behave badly as \( r \to 0 \) so difficulties can occur in the solution of the approximating problems. This is an area of current research in optimization methods.

A penalty function technique using the Nelder-Mead simplex algorithm was used on some problems at Babcock and Wilcox Co. These applications are discussed in section 5.

A good method in the case of linear constraints was proposed by Wolfe (1967). His method involves elimination of the nonbasic variables and the computation of a search direction involving the negative gradient with respect to the basic variables. Abadie and Carpenter (1969) generalize this method to handle nonlinear constraints, also using the notion of a reduced gradient but additionally maintaining feasibility in the presence of the nonlinear constraints. Research proceeds on the application of quasi-Newton type update schemes to methods of this type.

These two methods give an idea of the vast difference in approach which is possible for constrained optimization. There is an ongoing project organized by Colville (1970) of IBM to test programs for constrained optimization. Results of this should be of interest to users of optimization algorithms. No attempt has been made to discuss the questions of convergence or applicability for these methods because results of this type are difficult to handle in general.

4.4 Packages for Optimization

The user of optimization methods will probably look for a package which implements the algorithms which are needed for his applications. The package must have the breadth to cover all the desired applications. In particular, it should handle unconstrained problems and should exhibit the quadratic termination property. Additional features would include the ability to explicitly handle equality constraints and to be adapted to situations where matrices which occur are sparse.

The package should be flexible enough to provide the user the ability to get a quick solution if accuracy is not required or to obtain an accurate solution at increased cost. The user should be allowed to specify parameters such as convergence or differencing step criteria, but there should be a default option under which the package selects good values. The package should detect problems, e.g. a non-positive definite Hessian, and either attempt to correct the condition or terminate with an appropriate message. There should be a flexibility in the calling sequence of subroutines so that the user can call a preassigned "front end" to perform a common optimization procedure or he can organize the logic to suit his special problem.

The algorithms should be written in well-structured code. The package should exhibit modularity so that individual parts are understandable and so that changes and improvements can be easily made. Modularity also aids the user in designing his own routes through the package. Storage demands should be as simple as possible in that the user is required to dimension and allocate only a small number of arrays. Work spaces and data arrays which are internal to the code should be dynamically allocated to allow for various size problems.

The package should be documented at two levels. In-line documentation is important for maintenance or in-house adaptation. External documentation should provide the data for the individual algorithms and a complete description of their implementation in the package.

For lists of available software for constrained minimization, see Wright (1978) and Waren and Lasdon (1979). What promises to be a good code, MINPACK, is currently under development at Argonne National Laboratory. A description of the design and development philosophy behind this package can be found in Brown, et al (1976). A similar discussion for a package available from the British National Physical Laboratory can be found in Gill, et al. (1979d).

5. INDUSTRIAL APPLICATIONS

In this section four industrial applications are discussed. The first two deal with simulation, the language used for this purpose was GPSS (General Purpose Simulation System). The last two examples deal with optimization. All these applications stem from work done at the Babcock & Wilcox Company.

5.1 Foundry Simulation

The foundry simulation program was created for two main purposes: 1) scheduling and 2) to study the effects of man/machine availability on production. The program is subdivided into three parts. The main subprogram is concerned with the manufacturing of the different products, and consists of the
different steps necessary for their completion. For example:
molding + coring + pouring + etc.

The second subprogram is concerned with the pouring line. In essence, this simulates the melting of metal in the furnace, heat, and its consecutive pouring into the ladle and finally pouring from the ladle into the mold. The last subprogram consists of the coring line. In the foundry cores are started two days in advance of its respective mold. This is done to assure that the cores will be ready on time. In the simulation, therefore, whenever a mold is created by the GPSS generator, a core is started, and the respective mold is delayed 48 hours (GPSS clock time) before continuing with its processing. A portion of the GPSS blocks for the main subprogram is shown in Figure 4.

![Flow Chart]

Fig. 4 - Part of the GPSS Program Flow Chart

The unit of time used in the simulation was one minute. This selection was necessary in order to appropriately represent some of the processes in mold making. The simulation was also made to take into account the three different shifts of the day. Thus, it was possible to change the number of machines/crews working in each shift. This was specially important since some operations occur only during one or two shifts and not all day long.

The data built into the program consists of two different types: 1) time required for the completion of each step in the production line; 2) number of pieces of each commodity, i.e. flasks, furnaces, crews.
From the information obtained in 2), it is possible to see using the simulation if, say, adding an extra furnace for melting will actually increase production, or just cause a bottleneck further down the production line.

The foundry that was simulated makes hundreds of different products. In order to simplify the simulation, these products were broken down into 35 different classes. The criteria for this breakdown was that of grouping together those products which have approximately the same characteristics, that is, the same time required for the completion of different steps in the production line and the same commodity utilization. This simplification makes it necessary to translate the actual input data into the 35 representatives.

After some sensitivity analysis and comparison with past foundry loads, it was determined that the simulation could be used successfully to simulate periods of time of one month or more. When the simulation starts, the foundry is empty. In order to make a realistic simulation, it is therefore necessary first to achieve steady state conditions for the simulation. At this point, normal workload will have been achieved at each step of the production line. The current state of the foundry (steady state) is then stored, and any necessary simulations can then be run starting from this point. This process serves a dual purpose. First as previously mentioned, it builds up the queue lengths to their normal level. Secondly, it allows the queues to be built up for the particular type of load that the foundry might be under. This second point, allows one to see if under a particular load, a different type of load can be introduced and completed in a given timespan. For the particular simulation under study (with randomly generated queues), it was determined that a month's time (GPSS clock time) had to be simulated in order to properly build up the queues. On the IBM 370/155, the simulation of a month's time took approximately 30 CPU minutes with the information generated using up 200 tracks of storage space.

5.2 Computer Workload Simulation

This example deals with the simulation of the workload level of the IBM 370/158 under MVS. For the purpose of this study, three GPSS transaction generators were assumed. As seen in Figure 5, each generator corresponds to one of the three different types of transactions considered in our system, namely Batch, TSO and Data Base Transactions. Each of these types of transactions is assigned its own domain within the computer.

![Fig. 5 - GPSS Flow Chart of Computer Workload](image-url)
Once a transaction is generated, characteristics are assigned which differentiate it from the other two types. After the assignment, Data Base and TSO transactions go directly to the domain assignment. Batch transactions are first given a priority class, and then proceed to the domain assignment. In the domain, the GPSS clock time is placed into one of the transaction parameters and later used for CPU and service rate calculations. Also, at this point, a unique number is assigned to the transaction for later recognition.

The transaction is now split into two. The copy containing less parameters will be actively involved in calculating the service rate, CPU given to the transaction, swapping condition, and whether or not the transaction is finished. Using a duplicate, but with less parameters, for these calculations results in a savings of program running time. The original transaction will go to the CPU block. For the purpose of this simulation, there are only two ways in which a transaction can be swapped out of real memory. Both cases are controlled by the copy. The first of these is when the workload level of the swapped-in transaction exceeds that of a swapped-out transaction. In this case, the transaction executing will be swapped out, and the one with lower workload level swapped in. The second one is when the amount of time the transaction was in the swapped-in condition, duration time, exceeds a certain preassigned value. In this case, if there is a transaction waiting to be swapped in, the executing transaction is swapped out and the other swapped in. This preassigned duration time is domain dependent.

Once the processing for a particular transaction is complete, the copy will cause the original transaction to leave the CPU. The original and the copy will then rejoin and leave the system.

As mentioned in Garzia (1978a), in the performance evaluation of a complex computer system, it is necessary to use both simulation and mathematical modeling. The simulation model described above is used to simulate the transaction handling.

Queueing theory, however, is used to model the behavior within the CPU (Garzia 1978d&e). Unlike the previous example, the amount of CPU and storage used by this simulation is negligible on the IBM.

5.3 Tube Grouping

An important factor to consider in the design of superheaters (steam generator components) is the tube wall thicknesses. The minimum wall thickness necessary depends on the design pressure and temperature under which the particular tube will operate. When purchasing tubes, the ordered thickness is the minimum allowed thickness described above. The reason for this being that thicker tubes cost more than thinner ones. It is, however, usually true that as far as performance is concerned, a thicker tube can be substituted for a tube with minimum allowed wall thickness. It is this particular property which is exploited in the following economic tube grouping algorithm. This discussion is described fully in (Schmidt 1980).

For a particular thickness, tube cost depends primarily on the following: A base cost factor, a quantity factor, and the total length of tubes. The quantity factor decreases as the order quantity increases. The base cost factor, as previously mentioned, increases as the tube thickness increases. The minimum quantity factor will be obviously achieved if all tube thicknesses in the group of tubes under consideration are set to the largest minimum wall thickness allowed in the group. This, however, will increase the base cost factors for those tubes with a smaller minimum wall thickness. On the other hand, the minimum base cost factor will be achieved if each tube is ordered with its minimum allowable wall thickness. In this case, the quantity factor will not be optimal since the number of tubes in each thickness category will be smaller. This is the trade-off to be optimized.

To define a solution to this problem, it is assumed that all tubes have the same outside diameter, and that the given group of tubes consists of n subgroups. Each subgroup consisting of an allowable minimum wall thickness, a tube length, and the number of tubes in this subgroup. These subgroups are ordered according to increasing thickness, and subgroups with the same thickness are ordered according to decreasing total length. The subgroups are therefore completely defined by the following three variables:

\[
t_i - i^{th} \text{ minimum wall thickness} 
\]

\[
\lambda_i - i^{th} \text{ length} 
\]

\[
q_i - i^{th} \text{ number of tubes} 
\]

In our case, the quantity factor is obtained from the total weight of the tubes to be ordered. Thus with a particular subgroup, say subgroup i, there is associated a base cost per hundred feet of tube with thickness \( t_i \), \( c(t_i) \), and a quantity factor for the subgroup with total weight \( w_i \), \( p(w_i) \). With these definitions, it is now possible to state the general optimization problem:

\[
\text{minimize } C(t_1, t_2, ..., t_n) = \sum_{i=1}^{n} c(t_i) p(w_i) (L_i/100) 
\]
subject to

\[ I \; t \geq T \]  

(9)

where

\[ C \; \text{- total cost} \]

\[ I \; \text{- identity matrix} \]

\[ t \; \text{- vector of thicknesses} \; (t_1, t_2, \ldots, t_n) \]

\[ T \; \text{- vector of minimum allowed thicknesses} \; (T_1, T_2, \ldots, T_n), \]

\[ T_i \; \text{is the minimum thickness for the } i^{th} \text{ subgroup} \]

\[ L_i \; \text{- the total length of the } i^{th} \text{ subgroup} \]

The weight \( w_i \) is obtained by multiplying the density \( d(t_i) \) (depending on \( t_i \)) by the total length \( L_i \), so that we may write \( w_i = d(t_i) \).

The algorithm developed attacks a simplified version of this general problem. This simpler problem is obtained when the following two assumptions are made:

1. The thickness is allowed to take only the discrete values given for minimum thicknesses.
2. If two different subgroups are assigned thickness \( T \), then all subgroups in between use thickness \( T \).

With these assumptions, it is now possible to describe a simpler and computationally efficient algorithm. As seen in Schmidt (1980), the key ideas to this algorithm are those of contiguous sub-group and grouping which are defined as follows:

A contiguous subgroup, \( s_{ij} \), is defined as any set of consecutive subgroups starting with subgroup \( i \) and ending with subgroup \( j \). Its cost is given by

\[ c_{ij} = c(T_j) \cdot p(T_j) \left( \frac{\sum_{k=1}^{j} L_k}{100} \right) \]  

(10)

A grouping is any collection of contiguous subgroups \( [s_{i_k k+1-1}] \) (starting with \( i_1 \)), its cost given by

\[ c = \sum_{k=1}^{j} c_{i_k k+1-1} \]  

(11)

The costs of the contiguous subgroups can be put in matrix form where the \((i,j)\) entry is the cost of the contiguous subgroup \( s_{ij} \). From the definition \( i < j \) and so we obtain the triangular matrix

\[
\begin{pmatrix}
c_{1,1} & c_{1,2} & \cdots & c_{1,n} \\
0 & c_{2,2} & \cdots & c_{2,n} \\
0 & 0 & \cdots & c_{n,n}
\end{pmatrix}
\]

To form a grouping:

1. Set \( k = 0 \), cost = 0
2. Choose an entry in the \((k+1)\)st row, say \( c_{k+1,j} \)
3. Set cost = cost + \( c_{k+1,j} \)
4. If \( j < n \), set \( k = j \) and go to 2, else go to 5.
5. Write out cost.

The optimal grouping, subject to the above mentioned assumptions, is found by considering a staged process. The first stage minimum is set equal to \( c_{nn} \), and the minimum at the \( i^{th} \) stage is equal to the minimum cost over all the groupings of the last \( i \) rows of the matrix, denoted \( \text{min}_i \). The optimal value is \( \text{min}_n \). The sequence \( \text{min}_i \) is constructed as follows:
\[
\min_i = \text{minimum} \left[ c_{n-i+1,n-i+1} + \min_i ; c_{n-i+1,n-i+2} + \min_{i+1} ; \ldots ; c_{n-n-1,n-n-1} + \min_{n-1} ; c_{n-n-1,n} \right]
\]

(12)

This cost will actually give a minimum of the last \( i+1 \) rows since the grouping is formed by selecting an entry \( c_{n-i+1,n-j+1} \) and then adding to this costs from rows \( n-j \) on. These costs will all be non less than \( \min_j \) with \( \min_j \) actually attainable at some grouping of the last \( j \) rows. Thus, the minimum will be the smallest of the sums in (12). The efficiency of the algorithm is increased by noting that \( \min_i \) is a monotonically increasing sequence and so if \( c_{i,j} = \min[c_{i,1}, \ldots, c_{i,n}] \) then

\[
c_{1,i} + \min_{n-i+1} > c_{1,j} + \min_{n-j+1} \quad \text{if } j > i
\]

(13)

Thus, all rows with numbers less than or equal to the column number of the minimal entry in the first row do not have to be checked.

The procedure is in essence a sort algorithm, that is, the solution obtained is not necessarily the optimum. The reason for this is the second simplifying assumption, previously mentioned, which allows us to use the idea of contiguous subgroup. The Nelder-Mead simplex algorithm (see Sec. 4.3) was originally used to solve the problem. In this case, the second assumption was not necessary and the solution would have been optimal. This approach, however, was not productive due to the fact that it converged very slowly and at times even gave the wrong solution. It was, therefore, determined that the sort algorithm was superior in the sense that it gave a good solution, even though not optimal, and at the same time had a very fast convergence rate. This exemplifies the fact that in practical applications, classical optimization techniques are not always the answer.

5.4 Boiler Circulation System Optimization

In this study, the natural circulation system of a radiant boiler was optimized. This system consists of drum, downcomers, supplies, risers, and panels. Figure 6 obtained from Garzia (1976) describes the system under study.

![Radiant Boiler Circulation System](image)

**Fig. 6 - Radiant Boiler Circulation System**

It is assumed that the drum and panels are set by the design and arrangement of the boiler, therefore, the part of the system to be optimized is the downcomers, supplies and risers. Due to the complexity of the system, certain technical assumption had to be imposed. These will not be discussed here but details can be found in Garzia (1976). After the necessary assumptions are made, the next step towards the optimization of a complex system is the development of the mathematical models that describe its components. The kinds of techniques that can be used in such a process range from regression analysis to a complete and detailed application of the physical laws of the system in order to get the nonlinear integro-differential equations for its descriptions. Even though our system is complex, its components are not. The mathematical models thus developed, provide the lengths and bends of all pipes in the circulation system of radiant boilers, the flow distribution per downcomer, and the circuit balancing. It is beyond the scope of this discussion to derive and fully explain the equations and variables necessary for the optimization; these results can be obtained from the references. For our purpose, it suffices to state the problem:
\[
\begin{align*}
\text{minimize } & \left[ \sum_{j=1}^{n_d} c_{fd} \left( L_j, b_j, ID_d, \text{Thk}, \nu_{sdj}, \nu_{fdj}, \text{inj}, \text{Lag}_j, su_j \right) \\
& + \sum_{j=1}^{n_{CL}} \left( c_{fs} \left( L_{is}, b_{is}, \nu_{fsj}, \nu_{sis}, \text{injs}, \text{Lag}_{is}, \text{OD}_s, n_{si} \right) \\
& + c_{fr} \left( L_{ir}, b_{ir}, \nu_{fir}, \nu_{sir}, n_{ri} \right) \right) \right]
\end{align*}
\]

subject to

\[
\begin{align*}
L_j &= \hat{Q}_L(x_{dj}, y_{dj}, z_{dj}, x_{doj}, y_{doj}, \nu_{sdj}, ID_d, d_1, d_2, d_3, d_4, f, g, H) \\
b_j &= \hat{Q}_b(x_{dj}, y_{dj}, z_{dj}, x_{doj}, y_{doj}, \nu_{sdj}, ID_d, d_1, d_2, d_3, d_4, f, g) \\
w_{sdj} &= \hat{Q}_{ws}(L_j, b_j) \\
w_{fdj} &= \hat{Q}_{wf}(L_j, b_j, w_{sdj}) \\
\text{inj} &= \hat{Q}_{in}(L_j, ID_d, \text{Thk}) \\
\text{Lag}_j &= \hat{Q}_{lag}(L_j, ID_d, \text{Thk}) \\
suj &= \hat{Q}_{su}(b_j) \\
\text{L}_{is} &= \chi_L(x_{dj}, y_{dj}, z_{dj}, x_{si}, y_{si}, z_{si}, d_1, d_2, d_3, d_4, f, g, n_{si}) \\
b_{is} &= \chi_b(x_{dj}, y_{dj}, z_{dj}, x_{si}, y_{si}, z_{si}, d_1, d_2, d_3, d_4, f, g, n_{si}) \\
w_{sis} &= \chi_{ws}(L_{is}, b_{is}, ID_d, \text{OD}_s) \\
v_{sis} &= \chi_{ws}(L_{is}, b_{is}, w_{fsj}, \text{OD}_s) \\
\text{inis} &= \chi_{in}(L_{is}, \text{OD}_s) \\
\text{Lag}_{is} &= \chi_{lag}(L_{is}, \text{OD}_s) \\
\text{L}_{ir} &= \phi_L(x_{ir}, y_{ir}, z_{ir}, x_{di}, y_{di}, z_{di}, L_d, d_i, f_i) \\
b_{ir} &= \phi_b(x_{ir}, y_{ir}, z_{ir}, x_{di}, y_{di}, z_{di}, L_d, d_i, f_i) \\
w_{fir} &= \phi_{wf}(L_{ir}, b_{ir}, \text{OD}_s) \\
w_{sir} &= \phi_{ws}(L_{ir}, b_{ir}, \text{OD}_s, w_{fir})
\end{align*}
\]

where

\[
\begin{align*}
& j = 1, \ldots, n_d; \quad i = 1, \ldots, n_c \\
c_{fd} - \text{cost due to downcomers} \\
c_{fs} - \text{cost due to supplies} \\
c_{fr} - \text{cost due to risers} \\
n_d - \text{number of downcomers} \\
n_c - \text{number of panels} \\
n_{si} - \text{number of supplies in panel } i \\
n_{ri} - \text{number of risers in panel } i \\
L_d - \text{length of downcomer } i \\
b_d - \text{number of bends of downcomer } i \\
ID_d - \text{downcomer inside diameter} \\
\text{Thk} - \text{downcomer thickness}
\end{align*}
\]
\( w_{sd1} \) - number of shopwelds on downcomer \( i \)
\( w_{fdi} \) - number of field welds on downcomer \( i \)
\( i_{ni} \) - insulation material of downcomer \( i \)
\( \text{L}_i \) - lagging material of downcomer \( i \)
\( n_{si} \) - number of supports of downcomer \( i \)
\( L_{fs} \) - average length of supply \( i \)
\( n_{fs} \) - average number of length in supply \( i \)
\( w_{fsi} \) - number of field welds in supply \( i \)
\( w_{sis} \) - number of shopwelds in supply \( i \)
\( i_{sis} \) - insulation material for supply \( i \)
\( \text{L}_{sis} \) - lagging material for supply \( i \)
\( \text{OB}_s \) - outside diameter of supply
\( L_{r} \) - average length of riser \( i \)
\( b_{ir} \) - average number of bends on riser \( i \)

It is not important to define here the variables inside the constraint functions. The important thing to observe is that we have a nonlinear objective function with nonlinear constraints. Furthermore, there are discrete variables as well as continuous ones. The discrete variable is \( n_d \). To overcome this problem, an optimization is run for each possible number of downcomers and then the minimum cost is selected from these. The routine used for this optimization uses the Nelder-Mead Simplex Algorithm. The constraints were handled by a penalty function. This algorithm basically surrounds the optimum point with a polyhedron, and then successfully selects smaller polyhedrons making sure the optimum point lies inside these smaller ones. Obviously, the initial polyhedron selected has to be large enough to make sure it encloses the optimum point. The savings observed from this study were up to 15% of the total cost of the circulation system.

6. CONCLUSIONS

The general approach to the application of optimization in industrial design has been presented. Sensitivity Analysis has been emphasized as the most common analysis to detect the benefits of applying optimization techniques to a particular case study. Furthermore, this analysis can be used in the development of compact mathematical models.

When the shape of the cost function (or any other objective function, i.e., performance) and the constraints are known, the selection of the most convenient algorithm can be accomplished.

In order to select the optimization algorithm, the following needs to be taken under consideration:

- Type of the objective function (linear, quadratic, or nonlinear).
- Type of constraint functions (linear, quadratic, or nonlinear).
- Case with which function values can be obtained.
- Degree of accuracy required.

The analysis performed in the development of the mathematical models will make a strong contribution to the component's description. This in turn will allow the application of sophisticated control systems with on-line optimization algorithms.

The dual combination of computer technology and modern mathematics will change the relationship of time spent on analysis and design. Thereby, leading to better use of the available resources.
7. REFERENCES


