Chapter 1

Introduction

The history of the Marine MDO Project goes back a long way to the inception of MARITECH's FIRST Project. FIRST is a first principles approach for ship building integrated process and product development that integrates ship design, engineering and manufacturing systems. It is concerned with the development of computer-based design tools that couple multidisciplinary design optimization with design, scheduling and analysis to allow for the concurrent development of cost and refinement of build strategies. The Marine MDO project is a subset of this project wherein Virginia Tech's Department of Aerospace and Ocean Engineering was entrusted with the development of commercial software that uses optimization techniques in the design of a container ship.

The idea behind using an MDO model for commercial ship design is to break away from the traditional design-spiral or the "stove-pipe" analysis approach, and reduce the time needed to produce and analyze alternative ship designs. It gives the designer a high degree of flexibility and yet produces a design with the owner's requirement in perspective. Moreover, the MDO model can reap the benefits of modular programming. One of the biggest advantages of using modular programming is the capability of quick and cheap up-gradation of component modules without the need to compile the entire application. Thus, new high fidelity analysis codes can be attached, re-used or upgraded, whenever required, without altering the structure of the design process. Indeed, as the requirements of owners differ, the MDO model offers quick solutions to the design problems by changing a few modules instead of changing the entire design process.

Neu et al [6] chose a container ship as a test case for the MDO model and the "Required Freight Rate" to be a measure of merit that needed to be minimized. A design is defined by

a set of "design variables". The MDO design tool involves the integration of two processes: the analysis process and the optimization process. The analysis process calculates properties and performance evaluation factors of the design using first principles of Naval Architecture. The properties, such as weight, displacement, metacentric height, and the performance evaluation factors such as Required Freight Rate (RFR), resistance, propulsive efficiency, and operational econometrics, are all functions of the design variables. Given a set of design variables, the analysis process calculates the above parameters and uses them to calculate the set of constraints, which the designer imposes on the design process. The optimization process, on the other hand, uses a standard, commercial numerical optimization algorithm to change the values of the design variables with the goal of arriving at the optimum design. At this stage there is a choice between using a genetic or gradient-based algorithms. Considering the higher computation time for genetic searches and the computation-intensive nature of our high fidelity analyses, a gradient-based algorithm became the final choice. The interaction between the two processes as shown in Figure 1.1 is based on the flow of data obtained between the processes. The optimization process sends a set of design variables to the analysis process, which responds by using these design variables to calculate the objective function, i.e. the Required Freight Rate, and the set of constraints imposed on the design problem and sending these values back to the optimization process. This flow of data between the two processes continues until the optimization process cannot improve the design any further, i.e. the convergence is met.

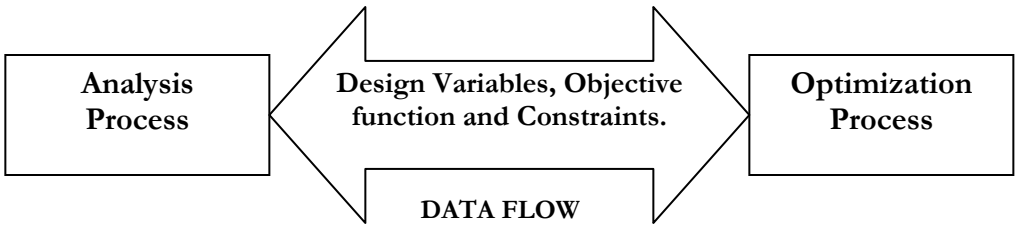


Figure 1.1 Interaction between the analysis and optimization process

The designer is allowed the flexibility of changing the input parameters for both the analysis and optimization processes through a user interface. This interface, which actually

changes the input file to the program, requires the designer to specify an initial set of design variables and constraints. It also provides him the opportunity to make changes to certain cost assumptions, such as the cost of steel per ton, used in the analysis process and make the software more realistic in terms of its estimates. The interface also allows the user to choose the optimization algorithm and change the input parameters for the optimizer. Thus the designer has sufficient scope over initializing the problem as per the owner's requirement.

The output from the analysis process is updated on a real-time basis and is shown on the user interface so that the designer can witness how the program changes the design variables in trying to achieve the optimum design. On termination of the optimization process the designer may view the iteration history and have the final values of the design variables along with the corresponding properties and performance evaluation factors of the design in a formatted text output file. There is a separate output file for just the optimization process that gives detailed numerical data for each iteration of the optimization process. This output file can be used to analyze the MDO problem from the algorithmic perspective of the optimization process.

The development of the MDO project was a continuous process over a period of three years and the contribution of every team member has been unique. Neu et al[6] developed a prototype tool for the multidisciplinary design optimization of ships. In this work, the team used surrogate parametric analysis modules to examine problem formulation issues prior to the availability of high fidelity modules. Vikram Ganeshan[7] in his *A Model of Multidisciplinary Design Optimization of Container Ships* develops the weight and the cargo module of the project. His methodology involves in treating the number of tiers of container on deck and the speed of the container ship as design variables. Ying Chen[8] in his *Formulation of Multidisciplinary Design Optimization of Container Ships*, has developed the resistance and economics module of the project. Both Chen's and Vikram's modules involve empirical formulation based on existing ship data. Amlan Dasgupta[9] in his *Addition of Features to an Existing MDO model for Container Ships* had introduced a

new design variables, ballast, to enable the optimization process to determine the amount of cargo carried and also added a new structural constraint.

This thesis concentrates on the optimization process of the MDO problem. In this effort, it delves into the intricacies of the optimization algorithm and suggests possible changes within the algorithm to customize it for the MDO problem. It takes the peculiarities of the design problem in the context of the optimization code that causes a premature termination of the process and investigates how the optimization algorithm can be changed to get a final design that can be regarded as a local optimum. The modification also increases the convergence rate of the process.

Chapter 2 gives the formulation of the problem by elaborating on how the MDO project is posed as an optimization problem. It also explains how the project is organized in a modular form and how the various modules interact to build the analysis and the optimization processes. Chapter 3 illustrates a problem in the project and gives a brief review of a relevant optimization algorithm. Chapter 3 is devoted to understanding the mechanics of the optimization process as used in the project. Chapter 4 relates the optimization process to the problem identified in Chapter 3. Chapter 5 proposes the solution to the problem and Chapter 6 derives the proposed solution and validates it for a number of test examples. Chapter 7 is a review of changes made to the analysis process. Chapter 8 gives a review of the software methods used in this project.

Chapter 2

Organization and Formulation

2.1 Project Organization

The two processes in the MDO project, the analysis process and the optimization process, are segregated into COM modules (see Chapter 8). The modules in the analysis process contain surrogate analysis codes that can be functionally divided into two groups: the properties group and the performance group. The optimization process contains two modules, one that contains the objective function and constraints and one that contains the optimization code. A Visual Basic main program module called **optimization manager** orchestrates these COM modules. The optimization manager transfers the input parameters from the user interface into the input COM module, creates all the interface objects and initializes all the COM modules. It also controls the iteration loop and the flow of data between the analysis and optimization process. The structure of the Optimization Manager is shown in Chapter 8. Figure 2.1 illustrates the position of each module in the process flow diagram.

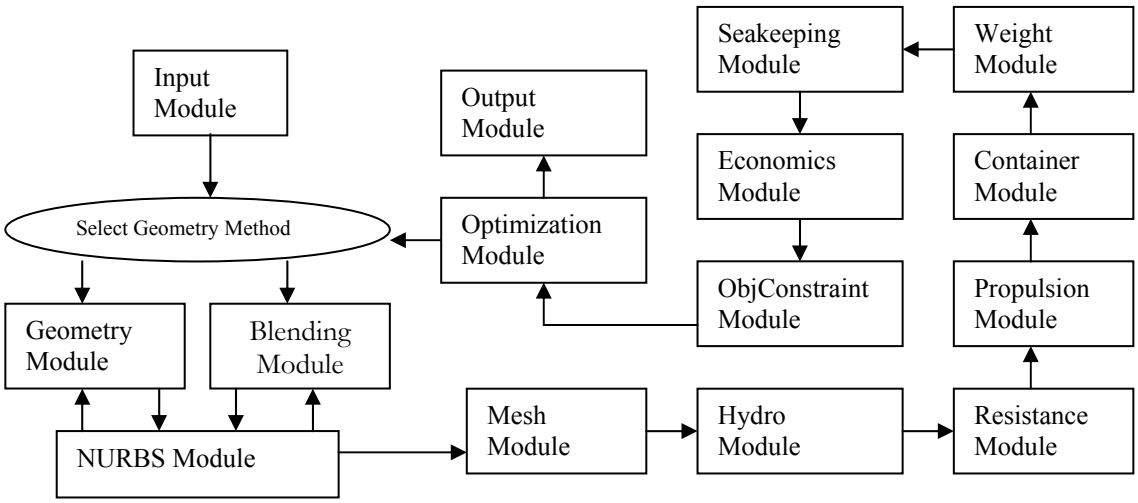


Figure 2.1 Process flow within the MDO project

The **Input Module** is concerned with reading the input parameters from the input subroutine of the optimizer manager through one of its interfaces. It exposes theses input parameters as properties to the rest of the modules and enables them to extract or update these parameters. Once the input data has been transferred to the Input Module, the Optimization Manager chooses the hull optimization method as per the user's choice and activates either the Geometry Module or the Blending Module. The **Geometry** and the **Blending** modules are individual methods that determine how the hull shape is optimized in relation to the rest of the design. In either method, the outline of the hull is generated from Non-Uniform Rational B-Spline (NURBS) surface expressions, which form a net of points. The geometry method takes a few patches at specific locations and moves them in a three-dimensional space. Once a patch is moved to a new location, the rest of the net points are scaled accordingly. This enables the shape of the net to be governed by the absolute location of the patches in a three-dimensional space. The optimizer is free to choose the position of theses patches relative to a reference frame and can thus control the shape of the NURBS net.

The Blending module, on the other hand, is concerned with the barycentric blending of two or more basis hulls. In this module, the net points of two basis hulls, a fine hull and a full hull, is combined in a weighted method to get a resultant hull. If N_1 and N_2 are the net points of the two hulls, the resultant hull is obtained by:

$$N_{res} = C_1 N_1 + C_2 N_2$$

Now the condition for barycentric blending is that $C_1 + C_2 = 1$. Thus the shape of the resultant hull can be controlled by the single weight coefficient C.

The **NURBS Module** is used to store the net points generated by the Geometry or Blending module and pass it on the **Mesh Module** where the actual hull surface is generated based on these net points.

The Mesh Module is the first module in the hydrostatics/properties group as it now gives a physical shape to the design. The hull shape and the physical dimensions are transferred to the **Hydro Module** where the properties of the design are calculated. Among other properties, the Hydro Module calculates the displacement, the metacentric height and the hydrostatic coefficients of the design. The displacement is required for the optimization as well as for the analysis process. For a design to be feasible its displacement must be equal to its weight and the metacentric height above the minimum metacentric height for stability. The hydrostatic coefficients are later used in the resistance and cargo capacity calculations. In the **Resistance Module**, the ship's resistance at the current speed is calculated using the Holtrop/Mennen regression formula. The appendage resistance is ignored and the effect of bulbous bow is taken into consideration. The effective horsepower is output from this module to the Propulsion Module.

Once the properties of the design have been obtained, the analysis process now looks for performance measures. The **Propulsion Module** is the first module in the performance group and assumes a propulsive efficiency of 0.65 and calculates the shaft horsepower and the rate of fuel consumption. As a more important measure of performance of the design, its cargo carrying capacity is calculated in the **Container Module**. This capacity is calculated in terms of TEU's -- Twenty-foot Equivalent Unit, (6.1m X 2.44m X 2.44m) for cargo above deck and below deck. The space available for carrying cargo below deck is discretized into lengthwise, beamwise and depthwise directions. The product of the number of containers that can be stored in each direction, multiplied by a stowage factor, which is a function of the block coefficient, gives the total number of containers that can be stowed. The space available for cargo above deck is also discretized into lengthwise and beamwise directions and the product of the number of containers that can be stored in each direction when multiplied by a stowage factor gives the number of containers that can be stored above deck. The stowage factors are calculated by curve fitting to a database of twelve ships. Since these values of the number of TEU's are discrete integer values, which would

cause numerical ill-conditioning, the cargo carrying capacity is made a continuous function by using a linear response surface fit.

The **Weight Module** calculates the estimates of the lightship weight and the center of gravity through parametric relationships based on formulas by Benford, Taggert and Schneekluth []. To include the effect of change of design, the coefficients of theses formulas were calculated from regression analysis of twelve modern container ships. The lightship weight of the ship is broken down into hull steel, outfit and machinery weight. Fuel weight is based on the specified ship range, its speed and the calculated fuel rate. Miscellaneous weights including crew and provisions, fresh water and lube oil based on formulas by Erichsen, are also added.

Using the relations of Benford and Erichsen [], the required freight rate (RFR) is determined in the **Economics Module**. To calculate the RFR, the total cost was split into the building cost and the annual operating cost. The building cost is contains the costs for labor and materials. The annual building cost is obtained from multiplying the building cost by a capital recovery factor to distribute it into an annual cost over the life of the ship. The capital recovery factor is given by:

$$\text{Capital Recovery Factor} = \frac{i(1+i)^n}{(1+i)^n - 1}$$

where i is the interest rate and n is the life of the ship in years. The annual operating cost depends on the number of round trips made annually which in turn depends on a number of factors like time spent at sea, cruising speed and time spent in port. The time spent in port again depends on the loading and unloading time, which depends on the number of cranes being used. The number of cranes is calculated based on the formula that one crane can be accommodated every 135 feet over 75 percent of the ships length. To include this in the RFR formula, a continuous form of the formula was obtained through a least square fit. These factors are related to the operating costs through wages, cost of stores and supplies, insurance, maintenance, repair port expenses etc. Finally the required freight rate is expressed as the total annual cost per ton of cargo carried per mile.

The analysis process has another module called the **Seakeeping Module**, which calculates the rolling period of the ship basing on its principle dimensions, and properties obtained from the Hydro module. With this, the analysis process comes to an end and the optimization process begins. With the values of the properties of the design and the performance evaluation factors, the **ObjConstraint Module** calculates the constraint values. The value of the objective function is input into this module from the Economics Module.

The **Optimization Module** contains the optimization algorithm code. A commercial optimization tool called DESIGN OPTIMIZATION TOOLS (DOT) from Vanderplaats Research and Development Inc. is used for the optimization process. This commercial code gives a choice of three different gradient-based algorithms for the optimization process. The input parameters for the algorithms are directly taken from the Input Module. The user has a choice of providing the algorithm with gradients calculated external to DOT or to depend on DOT to calculate the gradients. DOT is written in FORTRAN 77 and consists of a set of subroutines. Initially a C++ wrapper was constructed for it to be compatible with the COM environment but within Compaq's Visual Fortran, which allows mixed language programming, the C++ wrapper was no longer necessary. DOT is called as a function in the Optimization Module and communicates with the ObjConstraint Module to get the values of the objective function and constraints. The Optimization Module is directly linked to the Optimization Manager Module, which controls the iteration loop.

2.2 Problem Formulation

2.2.1 Design variables:

We can now define the MDO project as an optimization problem. To do that, we need to identify the design variables. The design of any ship must specify its principle dimensions – length, breadth, and depth. Thus these are obvious design variables. Since the values of these principle dimensions change with the length of the ship, their mid-ship values are taken to be the design variables. Draft was chosen to be a design variable so that the

optimizer could control the displace of the ship to satisfy the weight-displacement equality constraint (section 2.2.3). The speed is chosen as a design variable since the operating costs directly depend on it. By controlling speed, the optimizer can directly control the operating cost and thereby the RFR.

Another quantity that has a direct influence on the RFR is quantity of cargo carried. Since the number of tiers below deck is dependent on the hull geometry, which is again dependent on the principle dimensions of the ship it cannot not be taken as an independent design variable. On the contrary the number of tiers above deck can be taken not only as an independent design variable but also as continuous variable. The fractional part of the number of tiers on deck can be represented as an incomplete topmost tier. The idea of adding more tiers on deck would be more comforting if the optimizer had the chance to counter the decrease of stability due to the same. To do this ballast is added as a design variable.

By controlling the shape of the hull through the hull coefficient, the optimizer can control resistance of the ship and hence the power requirement of the ship. Since cost of the propulsion plant and the fuel are appreciable parts of the RFR, it makes good sense to have the hull coefficient as the design variable. So the final list of eight design variables is given in Table 2.1:

Design Variables	**Units**
Length	meters
Beam	meters
Depth	meters
Draft	meters
Speed	m/s
Tiers on Deck	--
Ballast	tons
Hull Coefficient	--

Table 2.1 The design variables

2.2.2 Objective:

The required freight rate is the minimum freight rate (RFR) for the owner to break even on his investment. Any freight rate greater than RFR would lead to profits. Thus the best measure of merit of a design is the RFR. The objective of the MDO project is,

$$\text{Minimize } \mathbb{F}(\mathbf{X}) = \frac{RFR}{RFR_i} \quad \text{where } RFR_i \text{ is the initial RFR.}$$

2.2.3 Subject to:

Constraints can be divided into equality and inequality constraints.

a) **<u>Equality constraint</u>**:

Weight = Displacement

Any ship that floats has to satisfy this condition. To numerically represent this condition, the constraint is split into two inequality constraints.

In the normalized form it is represented by: $\frac{\text{Weight}}{\text{Displacement}} - 1.0 \leq 0.0$

and, $1.0 - \frac{\text{Weight}}{\text{Displacement}} \leq 0.0$

b) **<u>Inequality constraints</u>**:

1)

Metacentric height $\geq$ *Minimum Metacentric height*

The metacentric height of a ship has to be greater than a minimum metacentric height to ensure the ship's transverse stability. The minimum metacentric height is calculated using the wind heel criteria set by the USCG.

In the normalized form it is represented by: $1.0 - \frac{\text{GM}}{\text{GM}_{\min}} \leq 0$

2) *Freeboard* $\geq$ *Minimum Freeboard*

The freeboard of the ship has to be greater than a minimum quantity to prevent the flooding of the deck. The minimum freeboard is also calculated as per USCG standards.

In the normalized form it is represented by $1.0 - \frac{FB}{FB_{min}} \leq 0$

3) *Draft/Depth* $\geq$ *minimum Draft/Depth*

This constraint ensures proper immersion of the propeller to prevent cavitation and to build thrust. The user can change the minimum value.

4) *Length/Depth* $\geq$ *minimum Length/Depth*

This is a requirement to stay within the range of applicability of Benford's[3] hull steel weight estimates.

5) *Transverse Rolling Period* $\geq$ *minimum Transverse Rolling Period*

This constraint is also stability-based constraint as it prevents the GM from getting too large. It is directly proportional to the GM and is undefined for negative values of GM. If GM is lesser than zero, it is calculated as an even function of the GM.

7) Container ship upper bounds:

Maximum displacement ≤ 70000

Maximum Shaft Horse Power ≤ 60000

Minimum Container Number (TEU) ≥ 2000

Maximum Container Number ≤ 20000

These upper limits are all user-supplied and can be changed in the user interface.

2.3 Scaling and Normalizing

DOT automatically scales the design variables according to its value. This is to prevent numerical ill-conditioning of the design hyperspace. Besides, the ObjConstraint module also scales the design variables before supplying it to DOT. While blending the basis hulls, the resulting net points after blending are scaled to the dimensions of the new design variable according to:

$$\text{x - net point} = \frac{\text{x - net point (from blending)} * \text{Length (Basis Ship)}}{\text{Length (Current Design Variable)}}$$

There is no way for DOT to determine what reasonable normalization factors might be. Thus the constraints have to be supplied to DOT after being normalized.

Chapter 3

The Problem of Adding Ballast

3.1 Statement of the Problem

With the Number of Tiers on Deck (NT_d) as a design variable, the optimizer was given the tool with which it could directly control the amount of cargo carried independent of the dimensions and shape of the ship. This made the minimum GM constraint more critical. As the objective function decreased with the increase in the amount of cargo carried, the optimizer tended to increase NT_d that resulted in the decrease of the GM. So the optimizer continued to add tiers on deck until the GM was reduced to its minimum value or, in other words, the GM constraint became active. To further increase the cargo carrying capacity of the ship, it became necessary to increase the GM of the ship by lowering the CG without altering the principal dimensions.

Thus, came the idea of ballast. Once ballast was introduced as a new design variable, the optimizer had the power to add ballast in thousands of tons to the double bottom tanks to lower the center of gravity. It was anticipated that the optimizer would recognize ballast as a means to reduce the GM of the ship while adding tiers on deck until either ballast reached its upper bound or the minimum freeboard constraint became active. What actually happened was quite different. To summarize the response of the optimizer with both NT_d and Ballast as independent design variables, we considered different starting points design space with differing values of Ballast. Table 3.1 compares the final design of a case which

does not incorporates ballast as a design variable to a case which incorporates ballast as a design variable and starting with a zero value of ballast.

Design Variables	Lower Bounds	Initial Design	Final Design (without ballast)	Final Design (with ballast)	Upper Bounds
Length	130	200	280.1348	280.81	300
Beam	18	35	42.96	43	43
Depth	15	20	21.5489	21.61	30
Draft	4	12.67	11.39	11.569	20
NTD	0	1	4.25	4.2949	35
Ballast	0	0	0	0.841511	20
Ob Fn		0.003116	0.001395	0.001387	

Table 3.1 Case 1: Starting with zero ballast

Design Variables	Lower Bounds	Initial Design	Final Design	Upper Bounds
Length	130	200	280.81	300
Beam	18	35	43	43
Depth	15	20	23.006	30
Draft	4	12.67	11.95	20
NTD	0	1	3.7032	10
Ballast	0	2	3.65	20
Ob Fn		0.003116	0.001361	

Table 3.2 Case 2: Starting with a non-zero value of ballast (Ballast=2)

Design Variables	Lower Bounds	Initial Design	Final Design	Upper Bounds
Length	130	200	279.497	300
Beam	18	35	43	43
Depth	15	20	21.5	30
Draft	4	12.67	16.348	20
NTD	0	1	7.62	10
Ballast	0	12	19.734	20
Ob Fn		0.003116	0. 001035	

Table 3.3 Case 3: Starting with a non-zero value of ballast (Ballast=12)

The three cases show that the difference between the final value of objective function with ballast and that without ballast, as a design variable, is not appreciable when the starting value of ballast is low. But, when starting with a higher value of ballast, the ballast quickly reaches its upper bound – a result that we had initially anticipated. The reason for this apparently anomalous behavior of the design variable ballast can be understood by delving into the mechanics of the optimization algorithm. In the next section of this chapter we explore the optimization algorithm in some detail.

3.2 Exploration of the Optimization Algorithm

The Design Optimization Tools (DOT version 4.2) by Vanderplaats is a robust numerical optimization code that has a choice of three nonlinear constrained optimization techniques, which are:

1) Modified Method of Feasible Directions (MMFD)
2) Sequential Linear Programming (SLP)
3) Sequential Quadratic Programming (SQP)

Comparing the results obtained from each of the above methods we concluded the SLP to be the most effective for our design problem. Thus our explanation of the optimization

process pertaining to the ballast problem will be based on the SLP method. But we also need to understand MMFD since SLP also uses MMFD as part of the process.

3.2.1 A review of Sequential Linear Programming

This method is used to solve n-dimensional, nonlinear, constrained optimization problems by successive linearization of the nonlinear functions. The objective function and the constraints are linearized about the starting point in the design space using first order Taylor series approximation. The solution to this linear problem is obtained using standard linear programming techniques. The problem is then re-linearized about the approximate solution. This process is repeated until the approximate solutions converge to actual solution of the problem.

Consider a general the linearized version of a nonlinear programming problem:

Minimize: $\mathbb{F}(\mathbf{X}) \approx \mathbb{F}(\mathbf{X}^0) + \nabla\mathbb{F}(\mathbf{X}^0)^T \delta\mathbf{X}$ where $\mathbf{X}^0$ is the point about which Taylor expansion is performed.

Subject to: $g_j(\mathbf{X}) \approx g_j(\mathbf{X}^0) + \nabla g_j(\mathbf{X}^0)^T \delta\mathbf{X}$ j = 1,m

$h_k(\mathbf{X}) \approx h_k(\mathbf{X}^0) + \nabla h_k(\mathbf{X}^0)^T \delta\mathbf{X} = 0$ k = 1, l

$X_i^l \le X_i + \delta X_i \le X_i^u$ i = 1, n

$\delta\mathbf{X} = \mathbf{X} - \mathbf{X}^0$

Clearly this represents a linear programming problem with the design variables in the vector $\delta\mathbf{X}$ and the functions at $\mathbf{X}^0$ as constants.

To physically demonstrate how SLP works the following set of diagrams from a program in Mathematica are used.

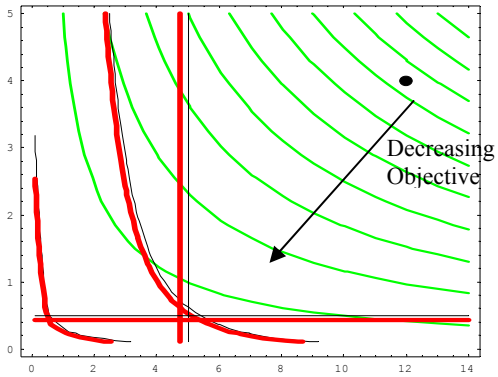


Fig. 3.1 Before Linearization

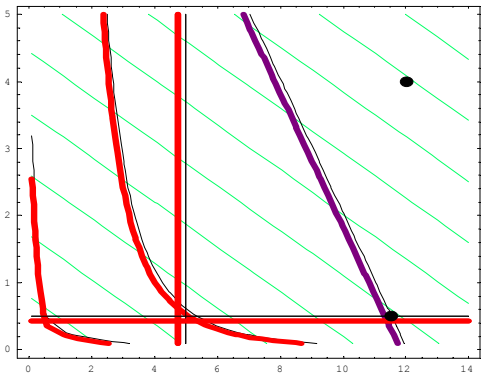


Fig. 3.2 After First Linearization

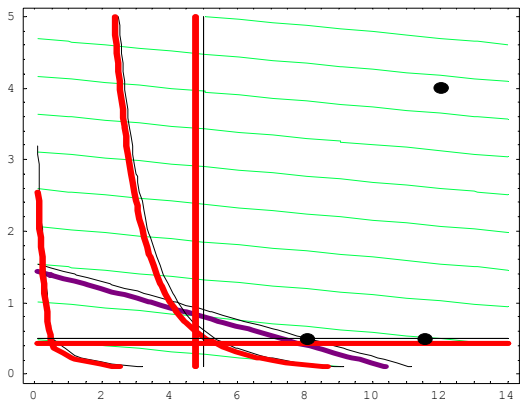


Fig. 3.3 After Second Linearization

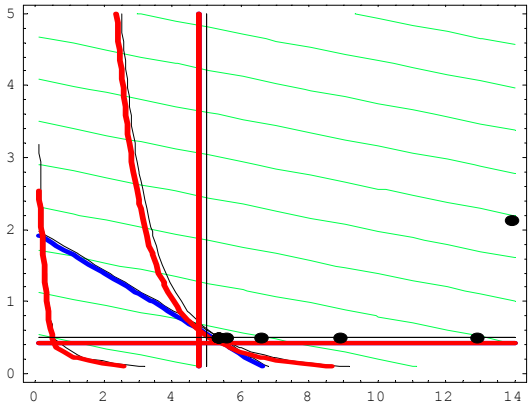


Fig 3.4 Last Iteration

The diagrams show the contours of the objective function in green and two linear and two nonlinear constraints. The red bold line denotes the infeasible side of the design space and we start the optimization from the point (12,4) – shown as the black dot. We can infer upon observation that the constrained minimum will lay near the intersection the two linear constraints and the nonlinear constraint.

After linearization about the starting point, the design space looks as shown in Figure 3.2. The green lines represent the contours of the objective function linearized about the starting point. The purple line is the linearized version of one of the nonlinear constraints. The other linearized constraint is outside the portion of the design space shown. When this linearized approximation to the problem is solved, we get a point shown as the second black dot near (12,0.5). We now re-linearize the objective function and constraints about this approximate solution in the second iteration and solve for the constrained minimum. The design space now looks what is shown in Figure 3.3. Solving for the minimum in the linearized problem gives the third approximate solution. This process is repeated until we arrive at the final solution as shown in Figure 3.4.

For constrained problems such as this, SLP converges rapidly to the solution. For under-constrained problems, the method performs poorly. To deal with such cases, move limits are imposed on the design variables. This ensures that the solution does not go unbounded

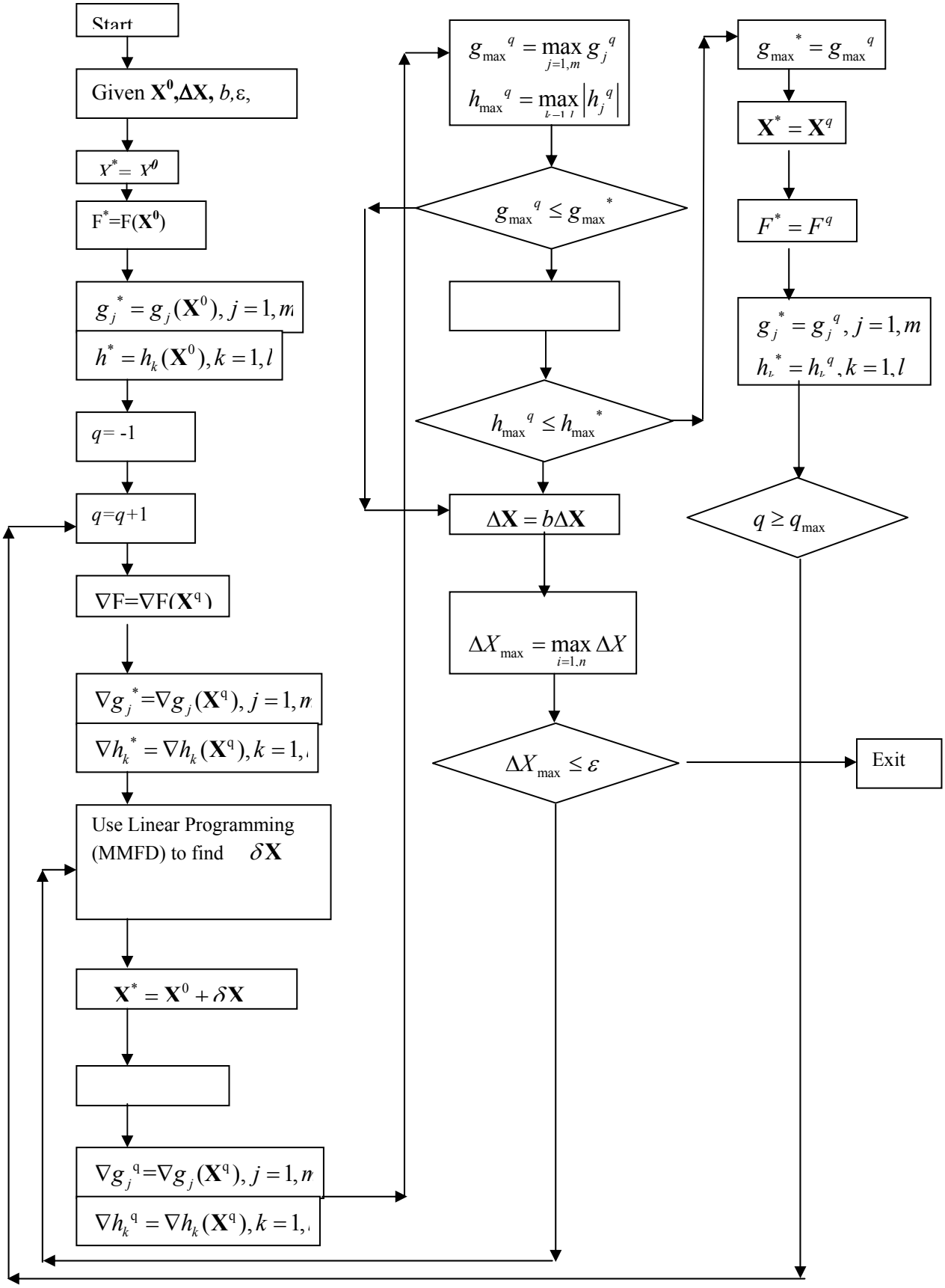
i.e., the values of the design variables do not reach infinity. To increase accuracy these move limits are decreased by a shrinkage factor as the optimization progresses.
There are various convergence criteria that may be used. The most common of them is the comparison between the relative changes in objective value for two consecutive iterations and a specified tolerance. The norm of the search vector cam also be used as a convergence criteria. As an upper limit to the optimization process, a maximum number of iterations may be used.

3.3 SLP with DOT

The SLP method was introduced in DOT version 2 and is the most effective method for our fully constrained type of problem. Using gradients of the objective function and the constraints calculated by finite difference method, the optimizer makes the linear approximation to the problem. DOT gives us a choice of the type of finite difference we want to use, viz. forward difference or central difference. When the problem is linearized, DOT uses the Modified Method of Feasible Direction to solve the linear approximate solution.

Figure 3.5 gives the SLP method with move limits as is used by DOT. This diagram is partly adopted from reference [2] and combined with findings of the algorithm at the program level. The linear programming problem is dealt with detail in the next section. The move limits are initially set to a value and depending on the progress of the optimization, this parameter is sequentially reduced by a shrinkage factor. Typically, the first approximate optimization produces a design that violates one or several constraints. However as the optimization proceeds, this constraint violation should be reduced. If it actually increases, we reduce the move limits. The actual move limit strategy used in DOT is much more complicated than shown in the figure as they are individually adjusted for every design variable.
Let $b, \varepsilon, q_{\max}$ be the shrinkage factor, minimum value for the largest component of $\delta \mathbf{X}$ and the
maximum number of iterations respectively.

Figure 3.5 SLP with DOT [1]



3.3.1 Solving the Linear Programming Problem.[]

Traditionally DOT used the SIMPLEX method but in our version it changed to Modified Method of Feasible Directions, since it works better for problems with numerous design variables normally encountered by DOT. The MMFD is a direct method of constrained optimization that is based on the equation:

$$\mathbf{X}^q = \mathbf{X}^{q-1} + \alpha \mathbf{S}$$

where $\mathbf{X}^q$ and $\mathbf{X}^{q-1}$ are the design variable vectors at the q-th and the (q-1)-th iteration of the optimization process. **S** is the search vector, which changes the design variable vector in direction that minimizes (or maximizes) the objective. α is a scalar that defines the amount of change in design variables in the **S** direction. Once we have a starting design point we can use the above equation iteratively to converge to the optimal solution.

Thus, we can split up the linear programming problem into two parts:

a) Finding the Search Direction, **S**
b) Finding the value of parameter α .

3.3.1.1 Finding the Search Direction

The first step to find the search direction using MMFD is to find out which constraints are active or violated. DOT requires that constraint values have to be negative for the design to be feasible. Therefore for a constraint to be active its value has to be zero. Due to imperfect precision in achieving zero in digital computer, DOT allows constraints to take a small positive value before it is considered violated. This small positive number is named CTMIN and has a value of 0.005. A constraint is considered active if it is greater than a small negative number CT. The value of CT is varied during the optimization process by initially setting it to a higher value to "trap" the design point in the active zone, and later reducing it to a lower value to increase the accuracy. Thus the governing definitions are:

$g_j(\mathbf{X}) \le \text{CT}$	*Inactive*
$\text{CT} \le g_j(\mathbf{X}) \le \text{CTMIN}$	*Active*
$g_j(\mathbf{X}) > \text{CTMIN}$	*Violated*

Using these definitions, all the constraints are sorted to find which ones are active or violated. While finding the usable-feasible direction there are three possibilities:

a) No active or violated constraints
b) Active constraints but no violated constraints
c) One or more violated constraints.

a) No active or violated constraints

In this case the feasibility requirement is already met, so the search direction has to be a usable direction, one that reduces (or increases) the objective function. For the first search direction at any time when there are no active or violated constraints, DOT used the steepest gradient method. The search direction by this method is:

$$\mathbf{S}^q = -\nabla \mathbb{F}(\mathbf{X}^{q-1})$$

If this initial search results in a point where there still are no are active or violated constraints, a conjugate search direction is used given by the Fletcher-Reeves conjugate direction method. The search direction for this method is:

$$\mathbf{S}^q = -\nabla \mathbb{F}(\mathbf{S}^{q-1}) + \beta \mathbf{S}^{q-1}$$

where,

$$\beta = \frac{\left|\nabla \mathbb{F}(\mathbf{X}^{q-1})\right|^2}{\left|\nabla \mathbb{F}(\mathbf{X}^{q-2})\right|^2}$$

The advantage of using the conjugate direction is that it represents a very simple modification of to the basic steepest descent algorithm, but remarkably improves the efficiency. The steepest gradient method searches in directions that are mutually perpendicular, whereas in the method of conjugate direction each search direction is slightly biased towards the previous direction. This simple modification dramatically improves the convergence rate.

b) Active constraints, but no violated constraints

The emphasis here is in finding a search direction that rapidly reduces (or increases) the objective function and at the same time maintains a feasible design. Assuming that we are

minimizing our objective, the dot product of the gradient of the objective function and the search direction must be lesser than or equal to zero that the direction of the search vector points normal to or away from the gradient of the objective function. Thus the condition for usability is:

$$\nabla \mathbb{F}(\mathbf{X})^T \mathbf{S} \leq 0$$

The direction is feasible if a small move in that direction does not violate the active constraint. In other words the dot product of the gradient of the active constraint and the search direction must be non-positive.

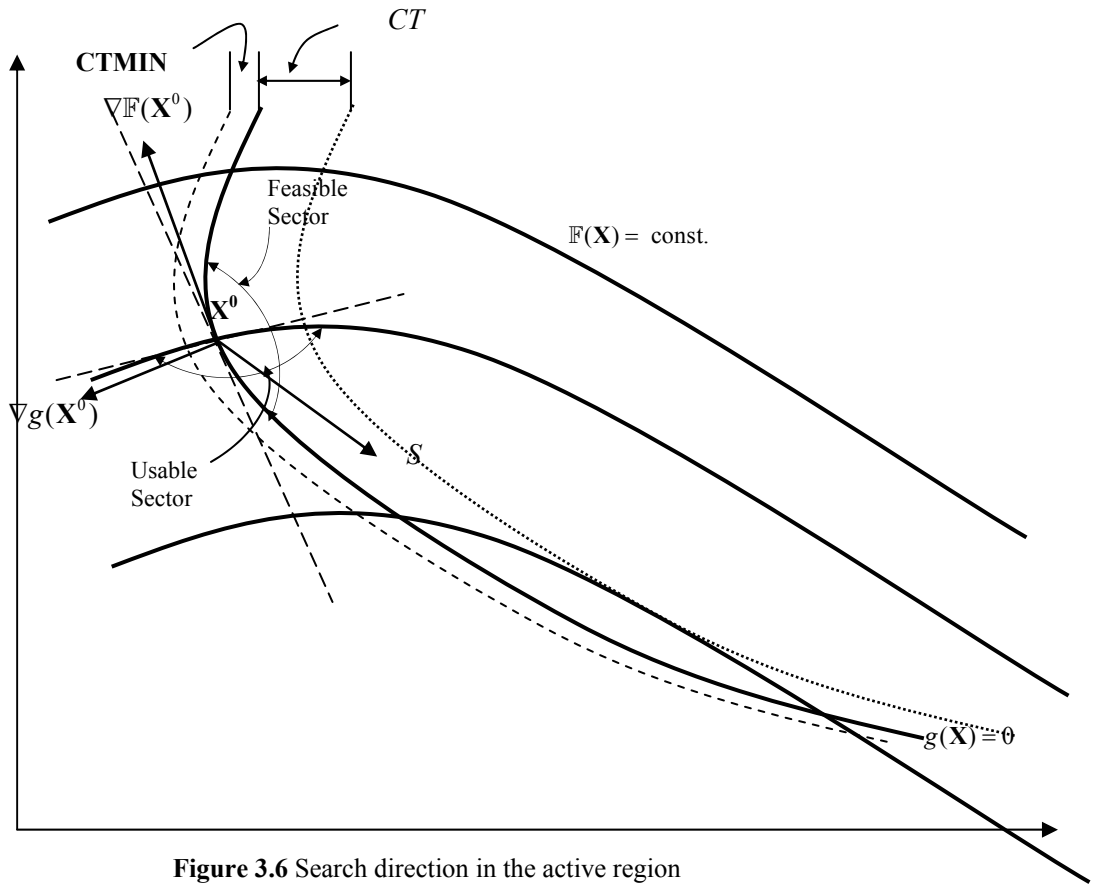


Figure 3.6 Search direction in the active region

Thus the condition for feasibility is:

$$\nabla g(\mathbf{X}^0)^T \mathbf{S} \leq 0$$

Stated mathematically, find the search direction, $\mathbf{S}^q$ that will:

$$\text{Minimize: } \nabla \mathbb{F}(\mathbf{X}^{q-1})^T \mathbf{S}^q$$
$$\text{Subject to: } \nabla g_j(\mathbf{X}^{q-1})^T \mathbf{S}^q \le 0 \qquad \text{Feasibilty Condition}$$
$$(\mathbf{S}^q)^T \mathbf{S}^q \le 1$$

The reason for the third equation is to impose bounds on **S.** It can be very easily understood that once a usable-feasible direction is found the solution is unbounded without imposing bounds on **S**.

The details of the sub-optimization problem are beyond the scope of this discussion and can be found in reference [1]. If no direction can be found that drives the objective function $\nabla \mathbb{F}(\mathbf{X}^{q-1})^T \mathbf{S}^q$ negative, while staying inside of the constraints, the Kuhn-Tucker conditions are met and the optimization problem is terminated.

c) One or more violated constraints

As shown in the Figure 3.7, constraint $g_1(\mathbf{X})$ is active and $g_2(\mathbf{X})$ is violated. It is necessary to find a search direction that results in a feasible design even if it is necessary to increase the objective function to do so. To achieve this DOT augments the objective function by an artificial variable **W**. The new direction finding problem now becomes:

$$\text{Minimize: } \nabla \mathbb{F}(\mathbf{X}^{q-1})^T \mathbf{S}^q - \Phi \mathbf{W}$$
$$\text{Subject to: } \nabla g_j(\mathbf{X}^{q-1})^T \mathbf{S}^q + \theta_j \mathbf{W} \le 0 \qquad j \varepsilon \mathrm{J}$$
$$(\mathbf{S}^q)^T \mathbf{S}^q + \mathbf{W}^2 \le 1$$

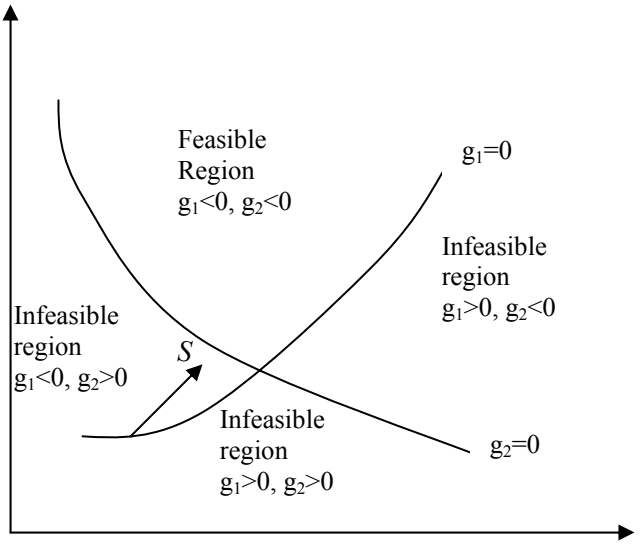


Figure 3.7 Search direction in the violated region

Φ is a large constant positive number and hence dominates the expression to minimize. For W to increase, the first term in the constraint equation has to be more and more negative. For a bounded $\mathbf{S}^q$ the cosine of the angle between the gradient of the constraint and the search direction has to be closer and closer to –1.0. For this to happen, $\mathbf{S}^q$ must point in a direction opposite to the gradient of the constraint. That is $\mathbf{S}^q$ must point straight back toward the feasible region. The reason for including the first term in the expression to minimize is simply to reduce the objective function if possible while finding the feasible design. The parameter θ_j is called the "push-off" factor, since its value determines how hard we push away from the violated constraint. It is chosen using the following equation:

$$\theta_j = \theta_0 \left[1.0 - \frac{g_j(\mathbf{X}^{q-1})}{\mathrm{CT}} \right]^2$$

$$\theta_j \leq 50.0$$

This is chosen so that $\theta_j = 0$ at $g_j(\mathbf{X}^{q-1}) = \mathrm{CT}$. But as the constraint value becomes larger so does the value of θ_j and the search direction is pushed harder and harder away from the constraint. The value of θ_j is limited to 50.0 to avoid numerical ill-conditioning. The value of Φ is initially set to 5.0. If feasible design cannot be found, Φ is increased by a factor of 10.0 but limited to an upper bound of 1000, again to prevent numerical ill-conditioning.

3.3.1.2 One Dimensional Minimization

Once the direction-finding problem has been solved and a usable-feasible search vector is found to exist, the value of parameter α, that is the amount to be moved in the search direction has to be found. Using equation $\mathbf{X}^q = \mathbf{X}^{q-1} + \alpha \mathbf{S}^q$, the objective function and constraints are evaluated as functions of α only. The process of one-dimensional search then determines the value of α that minimizes (or maximizes) the objective function while

remaining within the constraint boundaries. The process of one-dimensional minimization can be divided into three parts:

a) Making an initial estimation of α .
b) Finding the bounds for α .
c) Interpolating for α^*.

a) To make a good first estimate of α^*, the optimal value for α, a first order Maclaurin Series of $\mathbb{F}(\alpha^*)$ in terms of α^* is created.

Now, $\mathbb{F}(\mathbf{X}^q) = \mathbb{F}(\mathbf{X}^{q-1} + \alpha^* \mathbf{S}^q)$

The first order Maclaurin series approximation is

$$\mathbb{F}(\mathbf{X}^q) = \mathbb{F}(\mathbf{X}^{q-1}) + \sum_{i=1}^{N} \frac{\partial \mathbb{F}(\mathbf{X}^{q-1})}{\partial \mathbf{X}_i} \left\{ \frac{\partial \mathbf{X}_i}{\partial \alpha^*} \right\} \alpha^*$$

or, $\mathbb{F}(\mathbf{X}^q) \approx \mathbb{F}(\mathbf{X}^{q-1}) + \frac{d\mathbb{F}(\mathbf{X}^{q-1})}{d\alpha^*} \alpha *$

But $\frac{\partial \mathbb{F}(\mathbf{X}^{q-1})}{\partial \mathbf{X}_i}$ is just the i-th entry into $\nabla \mathbb{F}(\mathbf{X}^{q-1})$. Also, $\mathbf{X}_i^q = \mathbf{X}_i^{q-1} + \alpha^* \mathbf{S}^q$ so $\frac{\partial \mathbf{X}_i}{\partial \alpha^*} = \mathbf{S}_i$

Therefore $\frac{d\mathbb{F}(\mathbf{X}^{q-1})}{d\alpha^*} = \nabla \mathbb{F}(\mathbf{X}^{q-1})^T \mathbf{S}^q$. Since both the terms in this equation are available, the slope of the function at $\alpha = 0$ for any function is also available. This information can be used to find the initial estimate. For a 10% (say) decrease in the objective function:

$$\begin{aligned} \mathbb{F}(\mathbf{X}^q) &\approx \mathbb{F}(\mathbf{X}^{q-1}) + \frac{d\mathbb{F}(\mathbf{X}^{q-1})}{d\alpha *} \alpha * \\ &= \mathbb{F}(\mathbf{X}^{q-1}) - 0.1 \left| \mathbb{F}(\mathbf{X}^{q-1}) \right| \end{aligned}$$

from which we obtain a proposed $\alpha^*, \alpha_{est}^* = \frac{-0.1 \left| \mathbb{F}(\mathbf{X}^{q-1}) \right|}{\left[\frac{d\mathbb{F}(\mathbf{X}^{q-1})}{d\alpha^*} \right]}$

Thus this estimate of α^* reduces the objective by 10%. Since the gradients of the constraints are also available, other initial estimates of α^* can also be made. The value

of α^* that will reduce a constraint value to 0.0 can also be calculated. Using the same treatment used for the objective function, the linear approximation to find $g_j(\mathbf{X}^q) = 0.0$

$$g_j(\mathbf{X}^q) \approx g_j(\mathbf{X}^{q-1}) + \frac{dg_j(\mathbf{X}^{q-1})}{d\alpha^*}\alpha^* = 0.0$$

Therefore one of the estimates of α^* is $\alpha^*_{est} = \frac{-g_j(\mathbf{X}^{q-1})}{\left[\frac{dg_j(\mathbf{X}^{q-1})}{d\alpha}\right]}$

Hence there is one estimated α^* for each constraint. In other words even during the beginning of the optimization process there is enough information to get an initial estimate of α. Basing on these estimates the bounds for α^* is then found.

Finding bounds on α

As soon as an initial estimate is made, DOT now proceeds to find the bounds. Taking the initial point as $\alpha = 0$, DOT calculates the value of the objective function and the constraints at $\alpha_1 = \alpha^*_{est}$. To decide whether α_1 is an upper bound, there are three cases to consider.

Case I: When all constraints are satisfied at α =0.

In this case DOT checks if the objective function is being reduced or not. If the objective value at α_1 is higher, then α_1 becomes the upper bound. Also, since initial constraints are satisfied, if $g_j(\alpha_1) > \text{CTMIN}$, α_1 becomes the upper bound.

Case II: When one or more constraints are violated at α =0 and the objective function is increasing i.e. $\frac{d\mathbb{F}(\mathbf{X}^{q-1})}{d\alpha^*} > 0$.

Here the search direction reduces the constraint violation. If at α_1 the maximum constraint violation is increased, then α_1 becomes the upper bound. This could happen if a new constraint is violated at α_1. If all points between $\alpha = 0$ and α_1 is infeasible,

DOT tries to find a point that would minimize the constraint violation. Alternatively, if α_1 satisfies all the constraints, it is a feasible design and α_1 is made the upper bound as there would be a point between α=0 and α_1 that would have minimum objective value and still be feasible.

Case III: When one or more constraints are violated at α=0 and the objective function is decreasing i.e. $\frac{d\mathbb{F}(\mathbf{X}^{q-1})}{d\alpha^*} < 0$

This is the same, as Case II, except that DOT does not stop if the constraint violations are overcome. If a feasible design space is found, α_1 is multiplied by the golden ratio constant, 2.618. The objective and the constraints are then calculated for this value of α. The process is repeated until a constraint is violated or the objective starts to increase in which case the corresponding α becomes the upper bound. If more than four steps are taken then the best four solutions are retained for interpolation.

Interpolating for α^*

After obtaining the bounds on α^* DOT refines the solution using method of polynomial interpolation to find the value of α with the minimum value of $\mathbb{F}(\alpha)$ while remaining within the bounds. Depending on the amount of information available, i.e. the number of αs and their corresponding values of objective and constraints, DOT uses linear, quadratic or cubic polynomial curve fit. The details of the method of polynomial interpolation are given in reference [1].

When the final value of α^* has been obtained from the interpolation, the values of the objective and constraints are calculated to see if any constraints are violated or if the value of the objective is higher than that at α = 0. If the final design is infeasible or the value of the objective has increased, then the one-dimensional did not yield any result and the process for finding bounds for α is repeated for the next proposed α. This is repeated for every proposed α until a satisfactory final α^* is obtained. This satisfactory final α^* is then updated as α = 0 and the entire one-dimensional search process is

repeated to refine the solution further. The process is terminated when the maximum component of the difference between the design vectors corresponding to two consecutive calculated values of α^* is less than a tolerance ε. The one-dimensional search process is shown in Figure 3.8

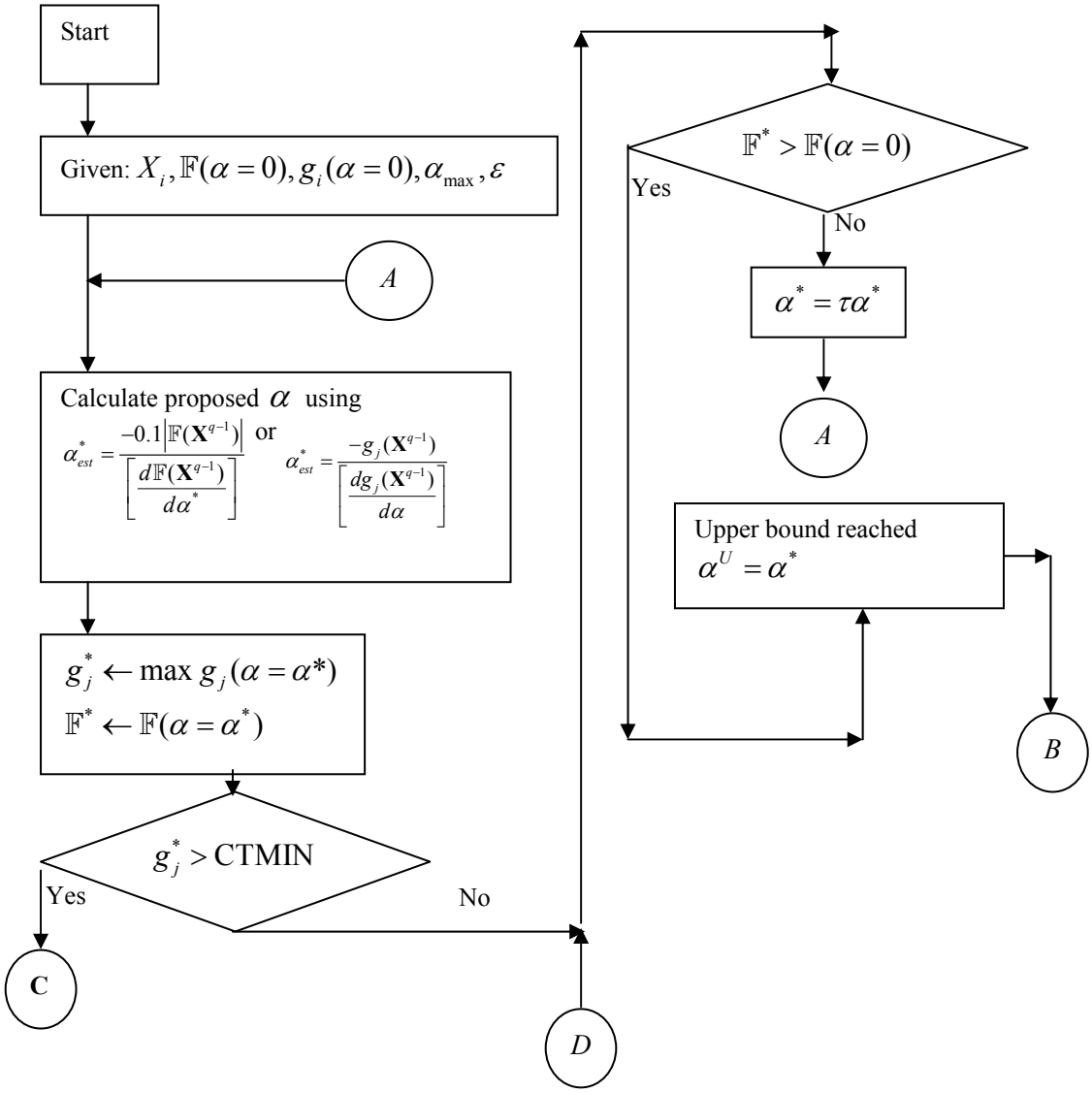


Figure 3.8 One dimensional minimization *(continued next page)*

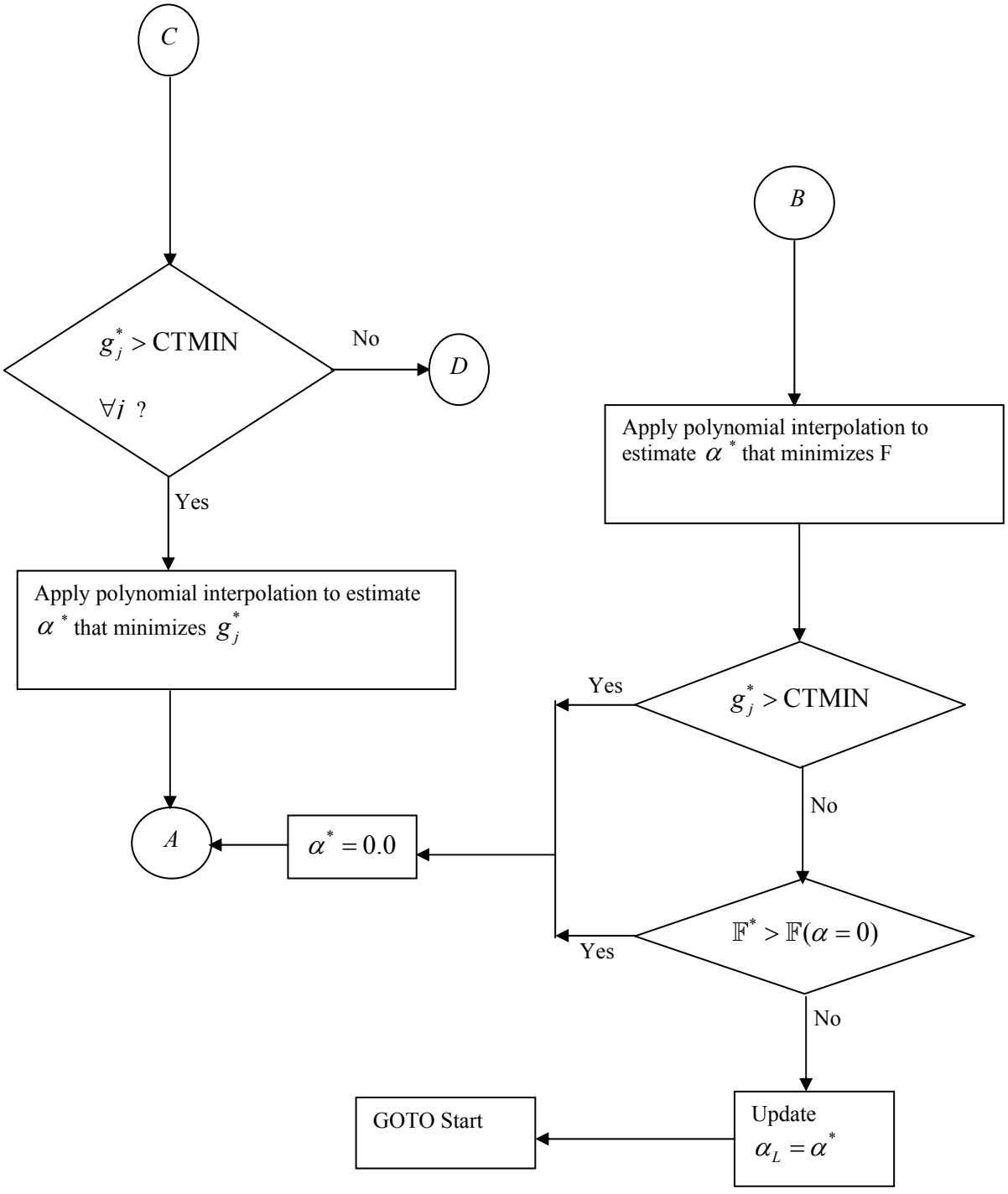


Figure 3.8 One dimensional minimization

Chapter 4

Causes of the Ballast Problem

Now that we have an understanding of the processes inside DOT we can revert back to our problem of trying to use ballast as a design tool to add more tiers on deck and hence lower the value of the objective function. Ballast is used in the design problem as an addition to the total weight of the ship. It is not explicitly represented in the objective function but participates in the optimization problem through constraints. This makes our problem a special case of optimization problems where the dimensionality of the problem is greater than that of the objective function. This specialty becomes important when we realize that the methods used for calculating the search direction in our problem are functions of the gradient of the objective function. Since the derivative of the objective function with regard to the ballast is zero, the search vector does not have any component that directly increases ballast i.e. the search is blind in the direction of ballast. In other words, the algorithm searches in a design hyperspace that is normal to the ballast hyperplane. But, when a constraint containing the ballast variable is active, the search direction also involves the gradient of the constraint and hence has a non-zero component in the direction of increasing ballast.

To summarize, when we are in the feasible region, the search direction is given by
$\mathbf{S}^q = -\nabla \mathbb{F}(\mathbf{X}^{q-1})$ in the first iteration and by

$$\mathbf{S}^q = -\nabla \mathbb{F}(\mathbf{X}^{q-1}) + \beta \mathbf{S}^{q-1}$$

where $$\beta = \frac{\left|\nabla \mathbb{F}(\mathbf{X}^{q-1})\right|^2}{\left|\nabla \mathbb{F}(\mathbf{X}^{q-2})\right|^2}$$

until any constraint(s) becomes active.

Once a constraint is active, the search direction is found by solving the sub-optimization problem of:

Minimize: $\nabla \mathbb{F}(\mathbf{X}^{q-1})^T \mathbf{S}^q$

Subject to: $\nabla g_j(\mathbf{X}^{q-1})^T \mathbf{S}^q \le 0$ Feasibilty Condition

$(\mathbf{S}^q)^T \mathbf{S}^q \le 1$

Thus in our problem, the search vector will contain a non-zero ballast component only when a constraint that is a function of ballast becomes active. Constraints that are functions of ballast are the weight-displacement equality constraint and the minimum GM constraint.

Upon analysis of the output obtained during the optimization process we have identified two causes for the ballast problem. They are:

a) The hypersensitivity of the GM constraint

b) Competing non-linear constraints.

4.1 Hypersensitivity of the GM constraint.

The results of the optimization process shows that it is dominated by the GM constraint. In other words, the tendency of the optimizer to try to change the design variables to obtain a better design is highly subdued by this constraint. Any change in the design variables along a search direction, during the one-dimensional optimization process, causes large changes in the value of the GM constraint. Therefore even a small change in the values of the design variables ends up in

violating the GM constraint. This can be shown from comparing the average change of the constraint values for the first four iterations per unit change of α . Table 4.1 shows this.

Constraints	$\frac{\Delta g_j}{\alpha}$
Weight-Displacement Constraint	3.48
Minimum GM Constraint	111.91
Minimum Freeboard Constraint	4.83
L/D min constraint	1.00
L/D max constraint	0.63
B/D min constraint	0.72

Table 4.1 Sensitivity of the constraints

Table 4.1 shows that the sensitivity of the minimum GM constraint exceeds that of the others by about a factor of 100. Also we could scale the L/D max and the B/D min constraints by 10 to increase its sensitivity. The purpose of the L/D max constraint is to increase the structural stability of the ship(see Chapter 7 for details). It denotes an approximate upper limit of the L/D ratio, and hence we do not try to increase its influence on the optimization process. Also, the results of the optimization show that the reluctance of the optimizer to increase depth almost always makes the B/D min constraint active. Hence we do not unnecessarily increase its sensitivity.

Figure 4.1 shows how the constraint values are changing with the iterations

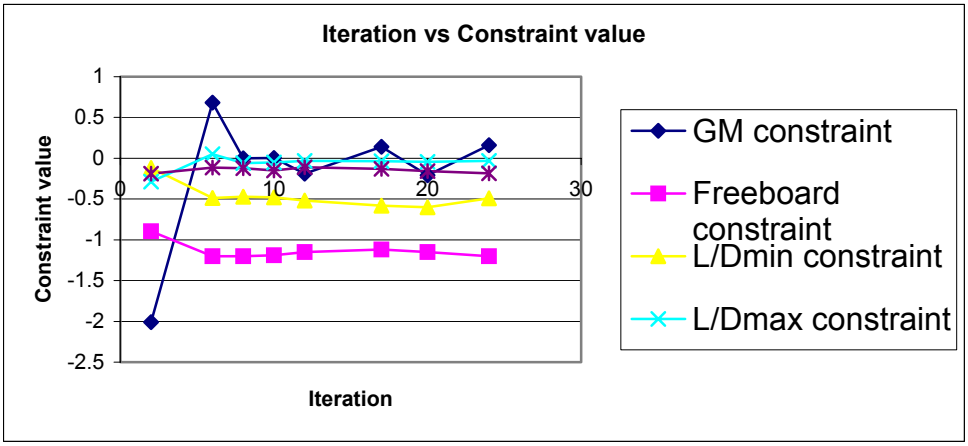


Figure 4.1 Iteration history of constraint values

As a result of this hypersensitivity of the GM constraint, the design point oscillates about the minimum GM constraint. To prove that this oscillation actually dominates the optimization process and is responsible for its premature convergence, we examine the gradients of the objective function and the minimum GM constraint.

$\frac{\partial \mathbb{F}}{\partial x_{length}}$	-2.4351×10^{-3}
$\frac{\partial \mathbb{F}}{\partial x_{breadth}}$	-1.6877×10^{-2}
$\frac{\partial \mathbb{F}}{\partial x_{depth}}$	-4.9501×10^{-3}
$\frac{\partial \mathbb{F}}{\partial x_{draft}}$	-7.2804×10^{-3}
$\frac{\partial \mathbb{F}}{\partial x_{speed}}$	-1.9558×10^{-2}
$\frac{\partial \mathbb{F}}{\partial x_{T\_D}}$	-2.092×10^{-1}
$\frac{\partial \mathbb{F}}{\partial x_{ballast}}$	0.0000

$\frac{\partial g_{GM}}{\partial x_{length}}$	7.3854×10^{-2}
$\frac{\partial g_{GM}}{\partial x_{breadth}}$	-4.7851
$\frac{\partial g_{GM}}{\partial x_{depth}}$	1.0611×10^{1}
$\frac{\partial g_{GM}}{\partial x_{draft}}$	-6.46
$\frac{\partial g_{GM}}{\partial x_{speed}}$	-4.402×10^{-1}
$\frac{\partial g_{GM}}{\partial x_{T\_D}}$	1.6834×10^{1}
$\frac{\partial g_{GM}}{\partial x_{ballast}}$	-3.8252

Table 4.2 Gradients of objective function and minimum GM constraint

Table 4.2 shows that the gradient of the objective function w.r.t the tiers on deck is the highest negative quantity i.e. adding tiers on deck would reduce the objective function at the highest rate. Also, we notice that the gradient of the GM constraint w.r.t the tiers on deck is the largest and is of an order much greater than that of the gradient of the objective function w.r.t tiers on deck. This means that even a slight increase in tiers on deck would severely violate the constraint. We can thus conclude that once the GM constraint is active and the principle dimensions of the ship have reached their corresponding side constraints, the optimizer has little or no chance to increase tiers on deck. As a result, it has little or no chance to increase ballast. This oscillation when combined with the decreasing move-limits leads to a premature convergence of the process.

4.2 Competing Constraints

Almost all initial design points for starting the optimization process are infeasible w.r.t the weight-displacement equality constraint. DOT deals with initially infeasible designs by considering the equality constraint as a violated inequality constraint. Section 3.3.1.1 explains how DOT finds a search direction that is almost normal to the violated constraint. This is to rapidly bring the design to become feasible. Thus the first iteration of the optimization will attempt to get the design point straight onto the weight-displacement equality constraint hypersurface while also trying to reduce the objective. Our results have shown that for initially low values of ballast, this first iteration results in a satisfied equality constraint but a violated GM constraint. This prompts the optimizer to search in a direction normal to the GM constraint that brings the design back to a feasible region w.r.t. the GM constraint. As a result DOT again brings the design back to satisfy the GM constraint thereby violating the equality constraint once again.

The problem of competing constraints is a more pervasive problem and in our case the competing weight-displacement equality constraint and the minimum GM inequality constraint aggravates the oscillation about the GM constraint. This oscillation causes the optimizer to converge to the first design point that has both the constraints active. A two dimensional analogy of the situation shown in Figure 4.2 depicts how two competing constraints cause oscillation.

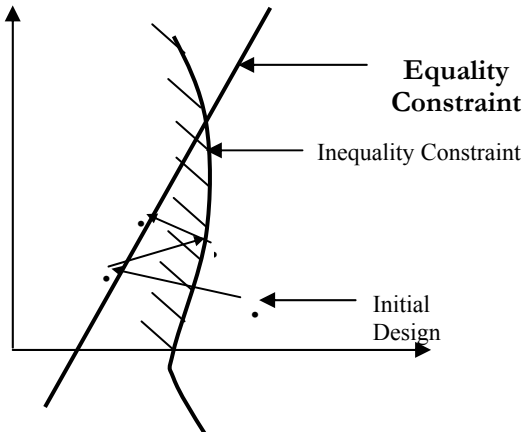


Figure 4.2 A two dimensional analogy

Chapter 5

Solution to the Ballast Problem

5.1 Scaling the GM constraint

An obvious solution to the hypersensitivity of the GM constraint is to scale it by a factor of 0.01. The scaling improves the algorithm in a number of ways.

First, it gives the optimizer a chance to take larger steps during the one-dimensional minimization process. As explained in Section 3.2.1.2, one of the methods of finding an upper bound to α, is to find the smallest value of α that violates a constraint. Previously, due to the hypersensitivity of the GM constraint, a small value of α would cause a large change in the value of the GM constraint. After scaling the constraint the optimizer can choose a much larger value of α for the same change in constraint value. That is, the value of α^* can be larger without violating the GM constraint. This brings us to totally different design points during the iteration history. The search directions at each of theses design points tend to decrease the objective much more than they did without scaling the constraint.

Second, the scaling has the effect of widening the tolerances of the constraint. A point in the design space, which may have been in the violated region w.r.t the un-scaled GM constraint, can be in the active region w.r.t. a scaled GM constraint. For example, a design point, which returns a GM constraint value of 0.3, violates the un-scaled constraint. But by scaling the GM constraint by 0.01, the same design point returns a value of 0.003, a value that lies within the active tolerances of the constraint. The "stretching" of the constraint tolerances, reduces the problem of the competing constraints by allowing the algorithm to quickly arrive at a point that satisfies both constraints. In other words the oscillation about the GM constraint persists much less before a point that satisfies both constraints is reached.

Third, the expansion of the constraint tolerances cause the activation of the GM constraint much earlier in the optimization process than when not scaled. As Section 3.2.1.1 explains, an active GM constraint would cause the search direction to be obtained from solving a sub-optimization problem that involves the gradient of the GM constraint. This means, that the search direction would have a non-zero ballast component from an earlier stage in the optimization process. Thus the optimizer can "see" the advantage of adding ballast before coming really close to the GM constraint. This early addition of ballast helps the optimizer to avoid most of the region of the competing constraints and quickly stabilize the oscillations. Figure 5.1 shows how scaling the constraints affect the optimization results. Note that every design variable is affected due to the scaling. Table 5.1 compares the final designs.

Design Variables	Initial Design	Final Design (Scaled Constraint)	Final Design (Un-scaled Constraint)
Length	200	279.25	281.45
Beam	35	42.9	42.98
Depth	20	21.48	21.65
Draft	12.67	16.33	11.79
Speed	15	17.24	18.45
Tiers_Deck	1	7.64	4.44
Ballast	1	20	1.92
C1	0.6	0.52	0.46
Objective (normalized)	1	0.33222	0.43779

Table 5.1 Comparison of results between scaled and unscaled constraints

Scaling the GM constraint reduces our ballast problem but does not eliminate it. After starting with several initial designs, we observed that when the initial design has no ballast, we do not converge at the global optimum. Table 5.2 shows this.

Design Variables	Initial Design	Final Design (Scaled Constraint)	Initial Design	Final Design (Scaled Constraint)
Length	200	279.64	200	280.86
Beam	35	43	35	43
Depth	20	21.51	20	21.60
Draft	12.67	11.90	12.67	15.92
Speed	15	17.14	15	17.41
Tiers_Deck	0	5.09	0	7.47
Ballast	0	1.16	5	19.57
C1	0.6	0.905	0.6	0.235
Objective (Normalized)	1	0.7637	Objective (Normalized)	0.3572

Table 5.2 Comparison of final results between differing values of ballast

Analysis of the results output during the optimization process has shown that the cause for this premature termination is the competing constraints. In spite of the GM constraint being scaled, when starting from zero ballast, the optimizer reaches a region in the design space where the design point keeps oscillating between the equality and inequality constraints. As the design keeps hopping between the two constraints, the move limits steadily decreases until the optimization process stops at the first feasible design that satisfy both constraints. In other words, the stretching of the constraint tolerances due the scaling of the constraints does not totally overcome the problem of competing constraints. Figure 5.1 shows the change of constraint values in the last few iterations of the optimization process for an initial design with zero ballast and zero tiers on deck.

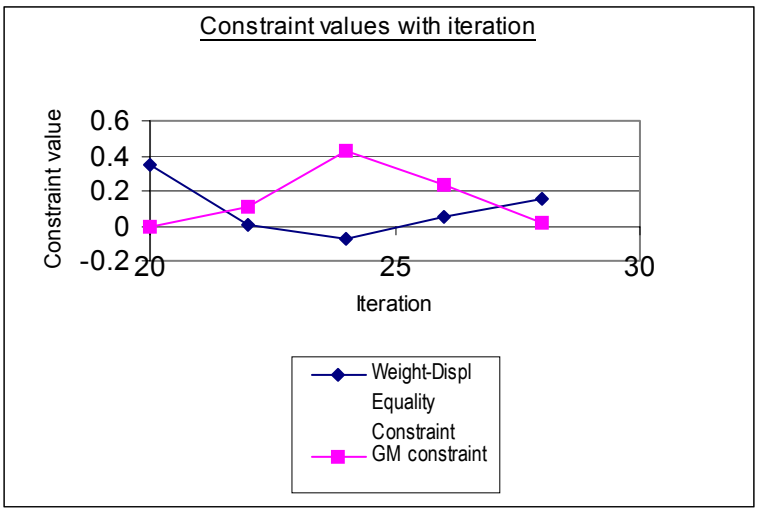


Figure 5.1 Constraint values for the last few iterations

5.2 Modification of the Search Direction

As explained earlier in Chapter 4, Causes for the Ballast Problem, that almost always the initial design is infeasible with regard to the weight-displacement inequality constraint. Thus the optimizer searches in a direction close to the normal of the equality hypersurface. Now, comparing the results of the optimization process for the case with a zero starting value of ballast with the case with a higher starting value of ballast, we can see that a higher value of ballast helps the optimizer avoid the region of competing constraints. For example, when we start with 6000 tons of ballast, we are at a region where we do not have the problem of competing constraints, but when we start with 1000 tons of ballast, we get stuck in the region of competing constraints. The scaling of the GM constraint helped in reducing the oscillations due to competing constraints for starting values of ballast of 500 tons or higher but failed to solve the problem for lower starting values of ballast. Thus we needed to think of an alternative solution for achieving a better result with low starting values of ballast.

It is usual to use various starting points in gradient-based methods to get the best design. In the MDO problem we realized that starting with a higher value of ballast helps the optimizer to avoid the region of competing constraints. But for any design optimization problem, as the number of design variable and constrains grow, it becomes difficult to get the right combination of initial values that converge to the best design. Thus we wanted the optimizer to identify and avoid regions of competing constraints automatically.

Our proposal for a modified search direction is applicable to the class of optimization problems that have a region of competing constraints. The next chapter shows how this modification is done and how it can be used favorably in the MDO model as well as any other gradient-based optimization algorithms. It must be remembered that this method is a modification to the existing gradient-based algorithm and helps in getting a better optimum result but does not guarantee a global optimum and consequently bears all the disadvantages of gradient-based algorithms.

Chapter 6

Modification of the search direction and its general applicability

6.1 Deriving the modification

To understand the method let us first consider the following two-dimensional optimization problem.

Design Variables: x_1, x_2

Minimize : x_1

Subject to:

$g(x_1, x_2) \leq 0$ Inequality Constraint

$h(x_1, x_2) = 0$ Equality Constraint

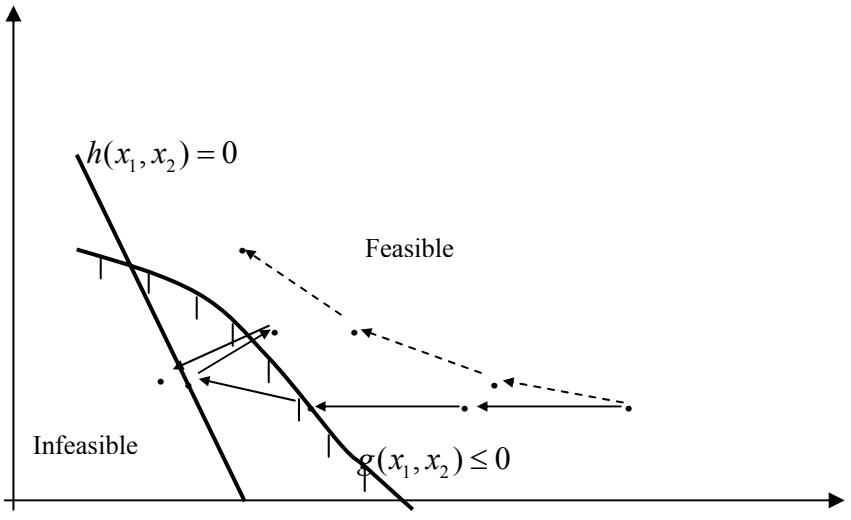


Figure 6.1 Two Dimensional Example

The solid arrows in the diagram show how the optimizer would normally behave. Once in the region of competing constraints, the diagram shows how the design oscillates in the region. We can also see that the lower the starting value of x_2, the more difficult it becomes for the optimizer to get through the region of competing constraints.

Now, if we were to modify the search direction so that the optimizer followed the path of the dotted arrows, we could reasonably reduce the chances of being trapped in the region of competing constrained even for very low starting values of x_2. We can also imagine that even when there is no region of competing constraints, by taking the path of the dotted arrows, the optimizer can actually improve the rate at which it reaches the optimum. Clearly, for the optimizer to follow the path of the dotted lines, it needs to have a search direction, which figures how to move along the constraint contours in a direction that reduces the objective function. This "turning" of the direction vector along the constraint contour needs to be activated form a position much before the constraint is active. In other words, the design should move up or down a constraint depending on which direction reduces the objective, from a point much before the constraint is active. Figure 5.2 shows the basis of our proposed modification.

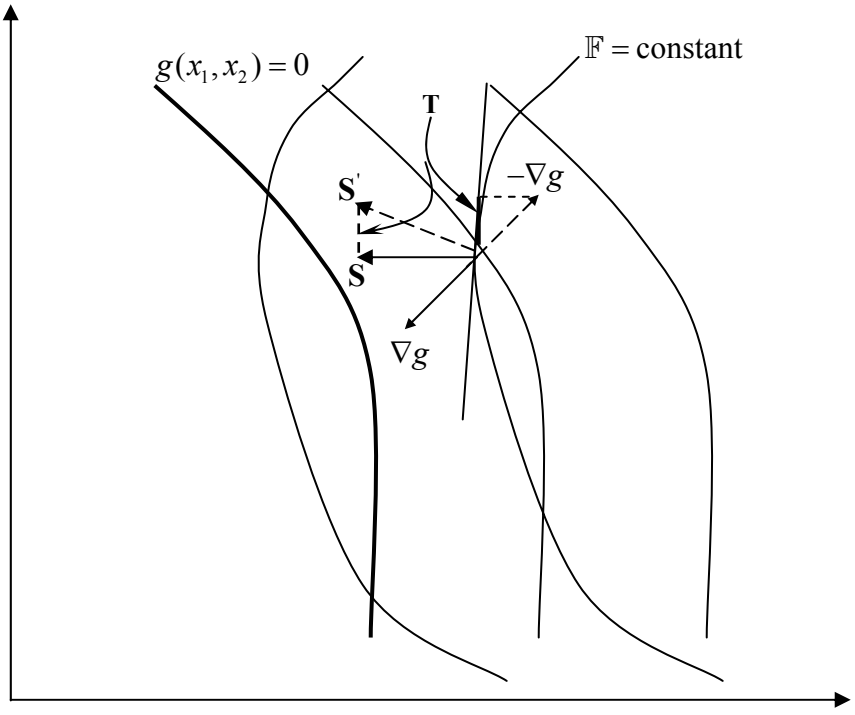


Figure 6.2 Modification of the search direction

To move up or down a constraint at a point where the constraint is not active, we project the negative of the gradient of the constraint onto the tangent of the objective function at that point. We then add this projection vector to the original search direction. The direction of the projection automatically determines which way the design point has to move along the constraint in order to reduce the objective function. For the problem stated earlier, where the objective function is not a function of x_2, the search direction is now aware of the full dimensionality of the problem and moves the design along a path shown by the dotted arrows.

To implement this modification in our MDO problem we project the negative of the gradient of the GM constraint onto the hypersurface tangential to the hypersurface of the objective function and add a multiple of this projection vector to the original search direction. Figure 6.3 shows how we obtain the projection in the two-dimensional case. Let S be the search direction.

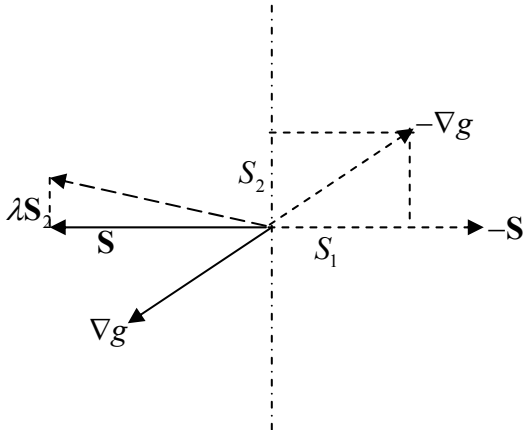


Figure 6.3 Modified search direction

The unit vector in the direction of S is given by $\hat{\mathbf{s}} = \dfrac{\mathbf{S}}{\|\mathbf{S}\|}$

$$S_1 = -\nabla g.(-\hat{\mathbf{s}}) = \nabla g.\hat{\mathbf{s}}$$
$$\mathbf{S_2} = -\nabla g - S_1(-\hat{\mathbf{s}}) = -\nabla g + S_1\hat{\mathbf{s}}$$

The unit vector in the direction $\mathbf{S_2}$ is given by $\hat{\mathbf{s}}_2 = \dfrac{\mathbf{S_2}}{\|\mathbf{S_2}\|}$

If S' is the new search direction, then,

$$\mathbf{S}' = \mathbf{S} + \lambda \mathbf{S_2}$$

Where λ is the scalar multiplier that adjusts the magnitude of the correction vector depending on the position of the design point. Typically, we would like λ to have larger values for points far away from the active region and smaller values for points closer to the active region. After trial for a number of design points we have come up with the following formulation:

$$\lambda = 0 \qquad \text{when } g(x_1, x_2) \le -50$$

$$= \frac{1}{50 g(x_1, x_2)} \quad \text{when CT} \ge g(x_1, x_2) > -50$$

$$= 0 \qquad \text{when } g(x_1, x_2) > \text{CT}$$

6.2 Implementation in the MDO model

The modification of the search algorithm was implemented in the DOT source code. This was done only for academic and experimental purpose with no commercial intensions. Here our competing constraints are the weight-displacement equality constraint and the minimum GM inequality constraint. To prevent the design point from entering a region of competing constraints we have modified our search direction so that the design points follow a path along the GM constraint in a direction that lowers the objective function. Table 6.1shows how the final designs differ.

Design Variables	Initial Design	Final Design (Scaled Constraint)	Final Design (Scaled Constraint And Modified Search Direction)
Length	200	2.79.64	279.58
Beam	35	43	43
Depth	20	21.51	21.67
Draft	12.67	11.90	16.35
Speed	15	17.14	17.59
Tiers_Deck	0	5.09	7.65
Ballast	0	1.16	20
C1	0.6	0.009	0.512
Objective (Normalized)	1	0.763751	3.57288E-01

Table 6.1 Comparison of final design

Figure 6.4 shows the iteration histories of the cases with and without modifying the search direction.

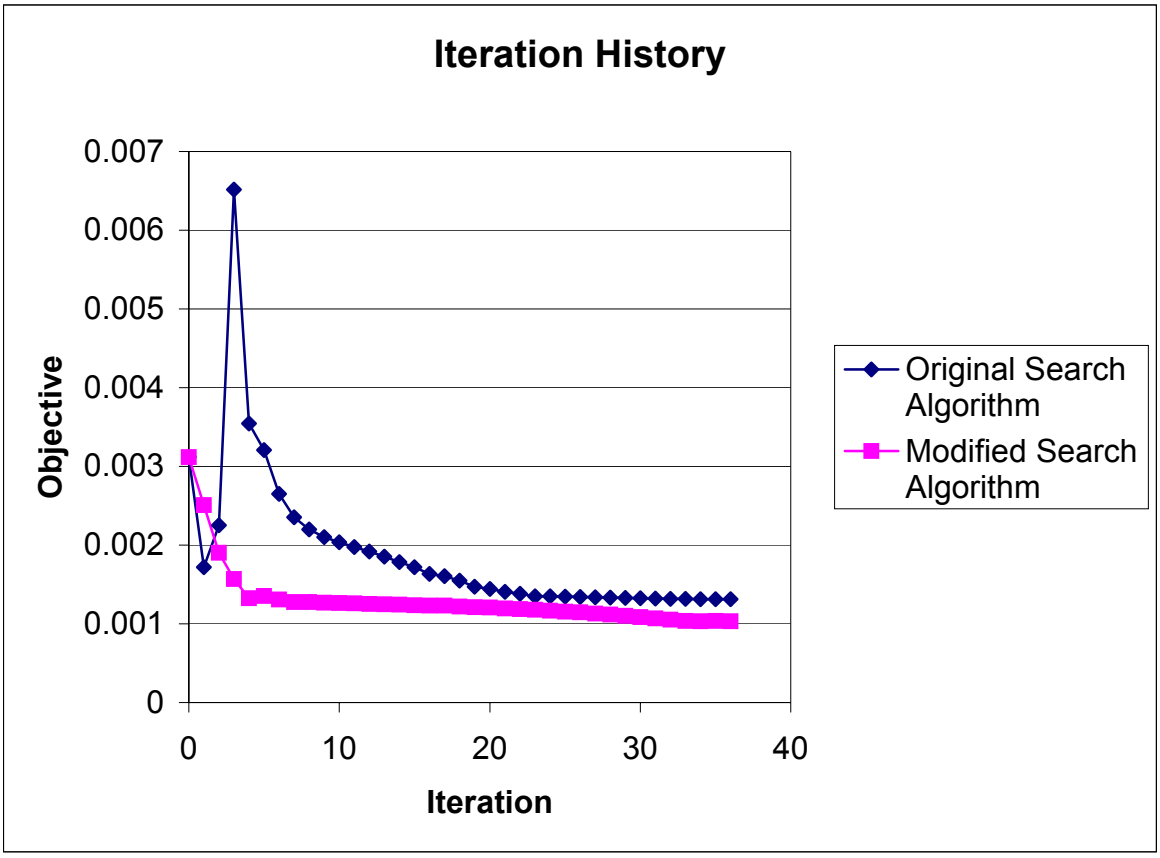


Figure 6.4 Iteration History comparing the two search algorithms

Table 6.2 shows the designs after starting from three arbitrary designs. With tolerances due to numerical "noise", the table shows that the modification is not susceptible to the initial design.

6.3 Implementation in general design optimization problems

To verify that our proposed modification works for any design optimization problem and to visualize its effect, we decided to implement it in a simple two-dimensional example that uses DOT as the optimization code. See Appendix 1 for the program.

Design Variables: x, y

Minimize : x

Subject to:

$y - 5x^4 - 14 \le 0$ Inequality Constraint

$y - 28x - 10 = 0$ Equality Constraint

$0 \le x \le 50$

$0 \le y \le 50$

Figure 6.5 shows how miserably the Modified Method of Feasible Directions fails in a situation where there are two competing constraints.

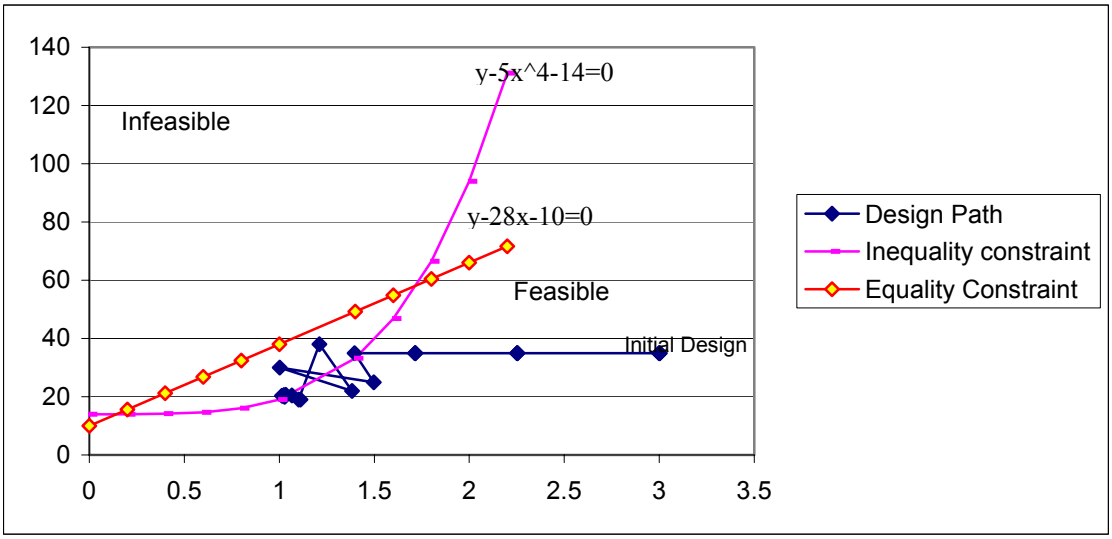


Figure 6.5 How DOT using MMFD performs in situation of competing constraints

Figure 6.6 shows how the modification in the search direction helps in overcoming the problem of competing constraints.

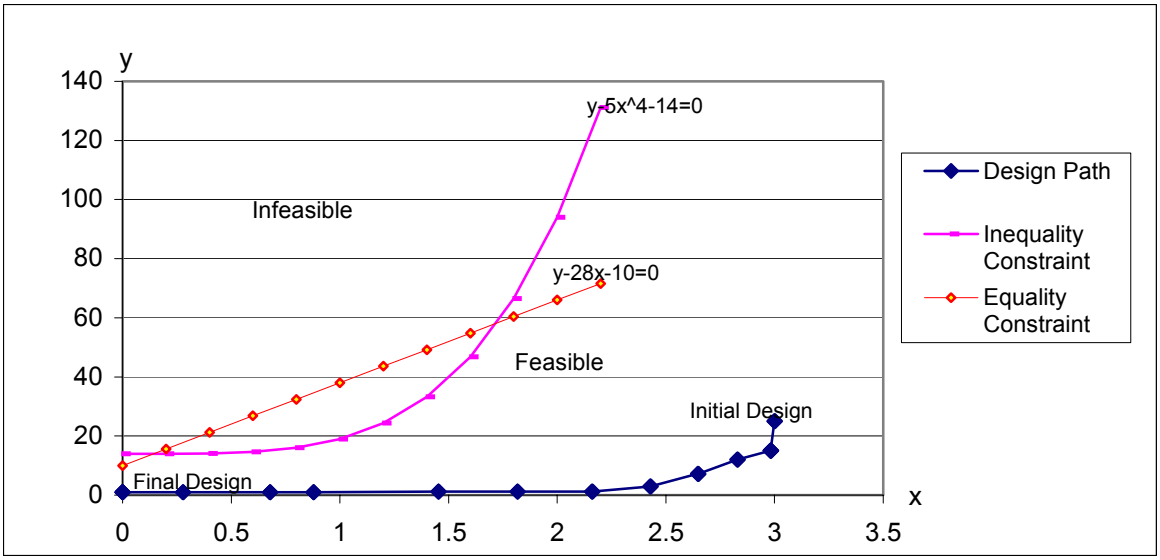


Figure 6.6 How modification of search direction helps overcome the problem of competing constraints.

6.4 General formulation of the search modification

To implement the search modification for any design optimization process that uses a gradient-based algorithm, we do not need to know if there are regions of competing constraints. All we need to do is to calculate the $\lambda \mathbf{S}_2$ vector for each constraint and vectorically add them to the original search direction. This provides the algorithm with a number of advantages as mentioned in the next section.

6.5 Advantages of using the search modification factor

The search modification is most useful in situations of competing constraints. Most design problems are multidimensional and involve a large number of constraints. Thus it may be impossible to analyze situations of competing constraints. The simple modification in the search vector allows for the reduction in and in some cases, like the MDO problem, to overcome the problems due to competing constraints, if such a situation exists in the design space. If competing constraints do not exist then it may add to the efficiency of the optimization process, depending on the design space.

The search modification also increases the convergence rate for problems where some design variables are not in the objective function. Traditional gradient-based algorithms use search directions that are functions of the derivatives of the objective function. Thus in cases where the dimensionality of the objective function is lesser than that of the whole problem, the search direction will have a zero component for the design variables not explicitly present in the objective function. In such as case, one of the constraints containing the design variables, not present in the objective function, has to be active for the search direction to be aware of the full dimensionality of the problem. The modification factor increases the convergence rate by the appropriate turning of the search direction even when constraints are not active.

The modification does not involve computation of any new quantities. It uses values already available to the optimization code. Thus it does not increase the computation costs.

Chapter 7

Changes to the Analysis Process

7.1 A New Constraint

This chapter details the changes to the analysis code that were made to pose the MDO problem more realistically. Any design of a ship is defined by a set of principle ratios. These are the ratios of the principle dimensions of the ship, for example, Length/Beam, Length/Depth, and Beam/Depth. Experience has shown that for better structural stability the principle ratios need to be within certain bounds. In this regard, an upper bound of the Length/Depth constraint was needed. This would not only ensure better structural stability but also improve the optimization process as explained in this section.

At the beginning of the optimization process, the optimizer looks for the best ways to increase the amount of cargo carried. Increasing the Length and the Beam is its favorite way of doing this. But once these two principle dimensions reach their upper bounds the other principle dimension left is the Depth. Unfortunately, the increase in cargo carrying capacity as a result of increase in Depth does not offset the cost of steel in increasing the Depth. This in fact, prompts the optimizer to decrease the depth as much as possible. The existing Beam/Depth constraint prevents the Depth from falling less that half Beam. So as the Beam reaches its upper limit, the Depth settles at half this value.

The benefits of increasing the Depth for the optimization process are indirect. A larger Depth would give the optimizer a chance for a larger Draft. A larger Draft would increase the transverse stability and hence make it possible for the optimizer to add more tiers on deck. This indirect payoff for increasing the Depth cannot be realized until there was some reduction in the negative sensitivity of Depth.

The incorporation of a maximum Length/Depth constraint directed some of the tendency of the optimizer to increase the Length to also increase the Depth in the process. Using a database of twelve container ships a maximum Length/ Depth value of 13.0 is reached. The results as shown in Table 7.1 illustrates the effect of adding the constraint. It is apparent that in there is no appreciable decrease in the objective function even though there is considerable decrease in length. The increase in the number of tiers on deck and the slight increase in depth offsets the increase in objective function due to a shorter ship. Thus we have a structurally stable ship with approximately the same measure of merit.

Design Variables	**Initial Design**	**Final Design (Without L/D_{max} constraint)**	**Final Design (With L/D_{max} constraint)**
Length	200	300	279.885
Beam	35	43	43
Depth	20	21.5	22.8
Draft	12.67	14.32	16.37
Speed	15	18.9	17.37
Ballast	0	16.87	19.99
Tiers_Deck	1	6.95	7.62
C	0.6	0.5	0.552
Objective	0.00311555	0.00103538	0.00103543

Table 7.1 The effect of adding L/D_{max} constraint

7.2 New Weight Formulation

To update our old weight formula to a new formula of the same form but different coefficients that reflect modern ships more closely, we chose twelve new ships from the Maritime Administration's, *Characteristics and Index of Maritime Administration Ship Design* [].
Here the Benford's [3] form of the estimate of weight of steel was used but the coefficients were changed to match closely the weight of twelve new ships. This was through a Least Squares fit.

Table 7.2 gives the comparison between the actual weights and weights obtained by our regression formula.

Ships	Length	Beam	Depth	Block Coeff.	Actual Weight	Weight by Fit	Error Square
1	204.12	27.43	16.5	0.5521	6949	7044.915	9199.674781
2	263.09	32.23	20.1	0.5524	13004	12540.31	215009.219
3	249.94	30.48	18.3	0.5941	10037	10914.76	770467.0533
4	219.61	28.96	16.5	0.5889	8666	8316.482	122163.0728
5	213.51	25.91	15.4	0.6129	6827	7130.635	92193.91456
6	247.88	27.43	16.2	0.6279	9529	9607.98	6237.766572
7	185.93	23.77	16.6	0.6431	4500	5806.464	1706847.657
8	203.8	23.16	13.6	0.6656	6451	5939.896	261227.7756
9	203.99	27.43	16.2	0.4634	6420	6644.468	50386.03911
10	211.86	32.16	20.7	0.5647	11937	9840.116	4396921.244
11	289.56	32.21	21.5	0.7411	16031	16167.79	18710.40857
12	205.99	32.31	18.8	0.598	8249	9117.846	754894.0812
						Sum =	8404257.907
						RMS. Error=	836.8720485

Table 7.2 Estimating the goodness of fit for the new weight formula

By removing the 10^{th} and the 7^{th} ship, there could be a considerable reduction in the RMS error. Figure 7.1 shows how the fit looks like after the 7^{th} and the 10^{th} ship has been removed.

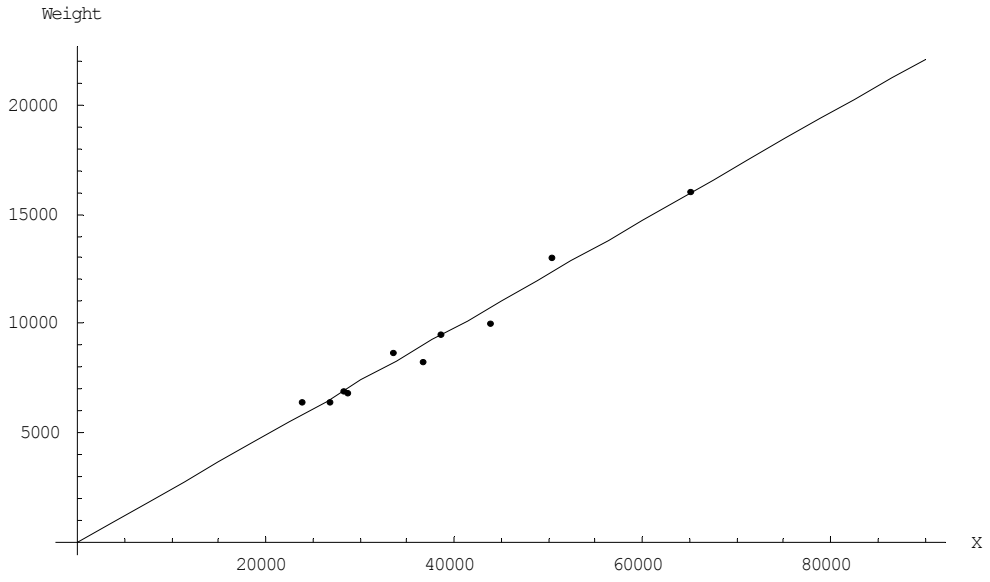


Figure 7.1 Demonstrating the goodness of fit of the weight formulation.

Chapter 8

Software Review

The FIRST project involves the development of software, simultaneously, at a number of places. Software developed at different locations, like the MDO project at Virginia Tech, will finally be integrated into a single large design tool. As such, FIRST needed a development environment that could put applications together quickly and cheaply from reusable, maintainable code, written by various people in various languages. The need for language neutral and object-oriented components of code made the Component Object Model (COM) a good choice. COM according to Grimes and Stockton [], is a specification and a set of services that allows the developer to create modular, object-oriented, customizable and upgradeable, distributed applications using a number of languages.

COM is an encapsulated chunk of code that can be accessed through interfaces. These interfaces are classes that contain an array of pointers, which map to individual set of methods or properties within the COM object. A method in a component can be accessed if it is exposed to the interface i.e. if the interface has a pointer to that function. For a client to access a particular method, it must have a pointer to the interface that exposes the desired method. If the client seeks another interface in the same object, it can call a method within the current interface to get transferred to another interface. Figure 8.1 shows how COM objects are represented. The interface **IUnknown** identifies that the object is a COM object. Details of the COM Programming are available in reference [11].

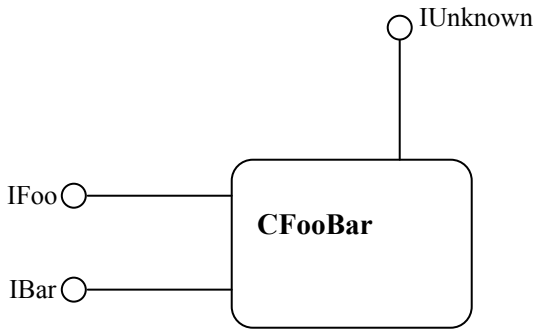


Figure 8.1 A representation of COM object with interfaces

The MDO project is divided into an analysis process and an optimization process. Each process consists of a set of modules, which are actually COM objects. These modules have their own set methods, which they expose to their interfaces and call methods or properties from other modules. The coordination of data flow is done by a Visual Basic subroutine called Optimization Manager.

The following gives an example of a typical module, its interfaces and the methods exposed.

Example: Input Module

This module consists of nine interfaces. Table 8.1 gives the list of the interfaces.

COM	***VTInput***
No	Interface
1	IVTInputFile
2	IVTShipParameterVoyage
3	IVTShipParameterOperationCost
4	IVTShipParameterBuildCost
5	IVTMDOSpecification
6	IVTMDOSpecificationDesignVar
7	IVTMDOSpecificationConstraint
8	IVTOptimizer
9	IVTOptimizerParameter

Table 8.1 Interfaces in the Input Module

Table 8.2 gives the methods that can be called from the interface **IVTInputFile**.

Methods	**Properties Called**	**Properties Exposed**
ReadFile	None	All properties Exposed to the Other interfaces
Save		
Save As		

Table 8.2 Methods and properties in interface IVTInputFile

Table 8.3 gives the properties exposed in the **IVTShipParameterVoyage** interface.

Properties	*Description*	*Units*
Distance	Round Trip Distance	nm
PortWaitTime	Port Waiting Time	day
OnlineTime	On-hire time per year	day
NrCrane_init	Number of cranes	
NrTierTEUonDeck_init	Number of TEUs on deck	

Table 8.3 Properties Exposed in IVTShipParameterVoyage interface

Similarly, other interfaces have their properties exposed. These properties are called from other modules for their calculations and methods and they in turn expose their own properties. The advantages of using COM are many. Since these analysis modules are surrogate codes that will be finally replaced by high fidelity codes, COM makes it easy to replace or upgrade these modules. COM components locate dynamically and COM defines standard ways of locating components and identifying their functionality so that individual components can be swapped without the need to recompile the entire application.

Chapter 9

Conclusion

The efficiency and the reliability of the numerical optimization process, such as the one used in the MDO model, depend on how well the analysis process is posed. A numerical optimization code cannot physically visualize a design problem. It relies on the magnitudes of certain parameters to "numerically understand" the problem and decide how to proceed. Given such an optimization process, the magnitudes of parameters input into the optimization process from the analysis process is an important factor for correct numerical conditioning. Optimization algorithms, such as DOT, do not have a way to regulate the sensitivity of the constraints or how much any constraint influences the optimization process. The influence of a constraint on the optimization process has to be determined in the analysis process by the use of scaling factors in addition to normalization of the constraints. The correct magnitude of the scaling factors is obtained from the analysis of the iteration histories. Scaling factors of the constraints are useful tools that can regulate the optimization process from outside the optimization algorithm.

The reliability of the solution obtained from an optimization problem depends on how well the algorithm is suited for the type of problem. A major drawback of gradient-based algorithms is its poor performance in situations of competing constraints. This drawback is augmented by problems, such as the containership design problem, where the dimensionality of the design space is greater than that of the objective function. A simple modification of the search direction helps in not only reducing and in some cases overcoming the problem of competing constraints but also in decreasing the total number of iterations.

References

[1] Garret A.Vanderplaats , *Numerical Optimization Techniques for Engineering Design: With Applications*, McGraw Hill, N.Y., 1984.

[2] Vanderplaats Research and Development, *Design Optimization Tool User Manual Version 4.2*, Colorado Springs, CO, 1995.

[3] Benford Harry, *The Practical Application of Economics of Merchant Ship Design,* Marine Technology, SNAME, January, 1967 .

[4] *Characteristics and Index of Maritime Administration Ship Designs*, US Department of Transportation, Maritime Administration, January 1991.

[5] W.L.Neu, W.H.Mason, S. Ni, Z.Lin, A.Dasgupta, Y.Chen, *A Multidisciplinary Design Optimization Scheme for Containerships,* 8th AIAA Symposium on Multidisciplinary Analysis and Optimization, 6-8 September 2000, Long Beach, CA

[6] W.L.Neu, O.Hughes, W.H.Mason, S. Ni, Z.Lin, Y.Chen, *A Prototype Tool for Multidisciplinary Design Optimization of Ships*, 9th Congress of International Maritime Association of the Mediterranean, April 2-6, 2000, Naples, Italy.

[7] Vikram Ganesan, *A Model for Multidisciplinary Design Optimization of Containerships*, Department of Aerospace and Ocean Engineering, MS Thesis, July 1999.

[8] Y Chen, *Formulation of Multidisciplinary Design Optimization of Containerships*, Department of Aerospace and Ocean Engineering, MS Thesis, May 2000.

[9] A Dasgupta, *Addition of Features to an Existing MDO Model for Containerships*, Department of Aerospace and Ocean Engineering, MS Thesis, May 2001.

[10] Grimes, Richard and Stockton, Alex, *Beginning ATL COM Programming*

915 University City Blvd #C6 Phone 540 951 8065
Blacksburg, VA 24060 E-mail sganguly@vt.edu

Sandipan Ganguly

Objective

Seeking an career in applied optimization.

Education

2002 Fall - **Industrial and Systems Engineering**, **Virginia Tech** Blacksburg, VA
Master of Science ***Major: Operations Research***

2000 Fall- 2002 Spring **Aerospace and Ocean Engineering**, **Virginia Tech** Blacksburg, VA
Master of Science(*Program ranked 12th by US News and World Report)* ***Major: Ocean Engineering***

- Program GPA : 3.5, Cumm: 3.33
- Graduate Research Assistant, specialized in **Design Optimization**. Developed a software in VC++ environme using Microsoft's Component Object Modelling, to optimize the design of a containership. Modified the optimization process changing the search-direction-finding algorithm to get a much better final result.

1992-1996 **Marine Engineering and Research Institute, India**
Bachelor of Engineering(*Premier Institute in India that selects only 126 candidates from the entire nation)*

- Received the prestigeous President's Gold Medal Award for outstanding academic performance, technical aptitude and leadership qualities.
- Received First Class With Distinction.

Employment

1996 Dec – 1998 April **Mitsui OSK Lines** Tokyo, Japan
Project Engineer

- Analyzed and re-designed the propulsion plant of an oil tanker that led to 5% increase in its utilization.
- Implemented Total Quality Management for better performance of a fleet of ships plying between Japan and US and reduced offhire time by 7%.

1998 April – 2000 May **Wallem Group Limited** Hongkong
Engineer

- Identified and rectified cronic losses in power generation plants of supertankers leading to the reduction in operating cost by $20000 per year.
- Applied CPM to minimize turnaround time of ships.

Computer Skills

Operating Systems: Windows 2000/NT, Linux ; **Languages**: C/C++, FORTRAN, Mathematica, Matlab; **Packages**: ARENA, MINITAB, AutoCad 2000, MS Office, Crystal Ball. **Special Skills**: COM programming using ATL and MFC.

Professional memberships

INFORMS, student member; SME, student member; SNAME, student member; ASNE, Member; Institution of Engineers (India), Associate Member.

Hobbies

Reading, music, trekking

References Available on request.