Very Large Scale Continuous and Discrete Variable Optimization

Garret N. Vanderplaats*
Vanderplaats Research & Development, Inc.
1767 S. 8th Street
Colorado Springs, CO 80906

An optimization algorithm is presented which is capable of solving nonlinear, constrained optimization tasks of well over one hundred thousand design variables. This method is an advanced exterior penalty function approach which uses very little central memory and very little computational time within the algorithm. The price paid for the large scale capability is that this method requires 3-5 times as many function and gradient evaluations to converge. An ad-hoc method for solving discrete optimization problems is included which reliably obtains a reasonable discrete solution. Examples are presented to demonstrate the method.

Nomenclature

$F(X)$	=	objective function
$g_j(X)$	=	j-th constraint function
m	=	total number of constraints
n	=	number of design variables
$P(X)$	=	penalty to drive design variables to a discrete value
q	=	iteration number in unconstrained sub-problem
q_j^p	=	individual penalty parameter
r_p	=	global penalty parameter
R	=	penalty parameter for discrete variable optimization
S^q	=	search direction at iteration q
X	=	vector of design variables
X_i	=	i-th design variable
X_i^L	=	lower bound on the i-th design variable
X_i^U	=	upper bound on the i-th design variable
Φ	=	pseudo-objective function
β	=	conjugate direction multiplier
∇	=	gradient operator

I. Introduction

With the increased use of optimization, the size of the problems being solved has grown rapidly. In structural optimization, problems with tens of thousands of design variables and millions of constraints are now being solved. For multidiscipline (MDO) problems, the number of variables and constraints can be even larger. In this case, in the past, decomposition methods were developed due to our inability to handle such large problems.

Non gradient methods, such as genetic algorithms, have been found to be both inefficient and unreliable for more than a few variables. Common gradient based methods, such as Sequential Quadratic Programming, can theoretically handle large problems but two issues quickly arise. First, these methods require solution of a large and often time consuming sub-optimization task to find the search direction and, second, they require storage of large

* President, Fellow AIAA.

amounts of information (both gradients and Lagrangian approximation information). This second issue may be handled using spill logic, but this can also be complicated and inefficient.

There is clearly a need for methods which will solver very large problems with limited central memory and which avoid large sub-optimization tasks. Such a method will be presented here. While it was developed primarily for structural optimization, it is equally useful for MDO tasks where gradient information is available. In either case, it is desirable to have high quality approximations since this algorithm requires more function and gradient evaluations than before.

When designing composite structures or piping systems, for example, the design variables must be discrete or integer. Genetic algorithms and branch and bound methods can be used here but are inefficient for problems of more than a few variables. The algorithm described here includes an ad-hoc method for solving large discrete or mixed continuous-discrete variable problems. This method does not insure a theoretical optimum but does produce a reasonable discrete solution efficiently.

II. Historical Background

Since the introduction of numerical optimization to structural design by Schmit in 1960[1], the size of problems solved by these methods has grown exponentially as shown in Figure 1.

In recent years, the focus in gradient based algorithm development has been on Sequential Quadratic Programming and similar methods which possess very strong convergence characteristics. However, as problem size has grown, these methods have been found to have significant limits. First, they require large memory to store necessary gradient information and second, they require a sub-optimization task to find a search direction. The memory requirement may be alleviated by out of memory storage operations but this is complicated and inefficient. The direction finding sub-problem can require significant computational effort which again leads to inefficiencies. In general, these modern methods can efficiently solve large problems with only a few active constraints or small problems with many constraints (because only a few will be critical). The difficulty arises when we have many design variables and many active constraints.

In the 1960's Sequential Unconstrained Minimization Techniques (SUMT) were popular[2] but, as noted above, where replaces by more direct methods. Recently, there has been renewed interest in SUMT, focused primarily on interior point methods[3].

In developing the present capability numerous SUMT methods were investigated leading to adoption of a new exterior penalty function approach[4], implemented in the BIGDOT optimization software[5]. This method will be reviewed here and recent enhancements will be identified. General enhancements have been made, allowing for solution of much larger problems. Also, a discrete variable algorithm has been implemented for solution of discrete or mixed continuous-discrete problems.

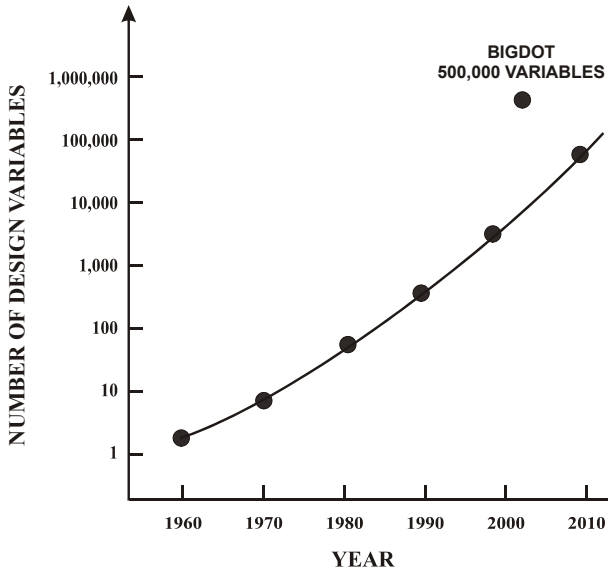

Figure 1. Growth in Optimization Problem Size

III. The BIGDOT Algorithm

The method developed here begins by solving the continuous variable problem and then finding a "reasonable" discrete solution. Therefore, it is required that the problem be solvable as a continuous problem. This is consistent with the assumptions of traditional branch and bound methods.

A. Continuous Variable Optimization

The basic approach used here is to convert the original constrained optimization problem to a sequence of unconstrained problems of the form;

Minimize

$$\Phi(X) = F(X) + r_p \sum_{j=1}^{m} q_j^p \{MAX[0, g_j(X)]\}^2 \tag{1}$$

Subject to;

$$X_i^L \le X_i \le X_i^U \qquad i = 1, n \tag{2}$$

where **X** is the vector of design variables, F(**X**) is the objective function and g_j(**X**) are the constraints.

The subscript/superscript, p is the outer loop counter which we will call the cycle number. The penalty parameter, r_p, is initially set to a small value and then increased after each design cycle. The only difference between this formulation and the traditional exterior penalty function is the addition of individual penalty parameters, q_j^p, on each constraint. These multipliers are similar to the Lagrange multipliers used in the Augmented Lagrange Multiplier Method[6], but are calculated by a proprietary formula. Equation 2 imposes limits on the design variables (side constraints) which are handled directly.

If equality constraints are considered, they can just be converted to two equal and opposite inequality constraints.

The unconstrained penalized function defined by Eq. 1 is solved by the Fletcher-Reeves conjugate direction method[7], which requires virtually no memory.

The gradient of Φ(**X**) is required during the optimization process.

$$\nabla\Phi(X) = \nabla F(X) + 2r_p \sum_{j=1}^{m} q_j^p \{MAX[0, g_j(X)\nabla g_j(X)]\} \tag{3}$$

Here, the choice of the exterior penalty function becomes apparent because only gradients of violated constraints are required. Furthermore, it is not necessary to store all gradients at once. Noting that Eq. 3 is a simple addition of gradient vectors so, in the limit, we can calculate only one gradient (objective or constraint) at a time.

As an indication of computer memory required by various methods, the proposed method is compared with the three methods used by the DOT program[8]. This is presented in Table 1, where MMFD is the Modified Method of Feasible Directions, SLP is the Sequential Linear Programming Method and SQP is the Sequential Quadratic Programming Method.

Table 1. Storage Requirements

	Number of Design Variables		
Method	100	1,000	10,000
MMFD	53,000	$5X10^6$	$5X10^8$
SLP	113,000	$11X10^6$	$11X10^8$
SQP	119,000	$11.5X10^6$	$12X10^8$
BIGDOT	1,400 to 11,000	14,000 to $10X10^5$	140,000 to $10X10^7$

The memory requirements for the DOT methods are the number of words for storage of all internal arrays. For the BIGDOT method, the two memory requirements are the minimum, where only 1 gradient vector is calculated at a time, and the maximum, were all possible gradients are stored in memory. The number of constraints equals the number of design variables.

As can be seen, as the problem size grows, storage requirements for the earlier methods grows exponentially. However, for the present method, storage is much less and the requirement grows only linearly with problem size. If there are many more constraints than design variables, the requested storage for the earlier methods grows even more rapidly.

As noted above, the unconstrained minimization sub-problem is solved by the Fletcher-Reeves algorithm. Here, at iteration q, the search direction is found as;

If q = 1

$$S^q = -\nabla\Phi(X^0) \tag{4}$$

If q > 1

$$S^q = -\nabla\Phi(X^{q-1}) + \beta S^{q-1} \tag{5}$$

where

$$\beta = \frac{\left|\nabla\Phi(X^{q-1})\right|}{\left|\nabla\Phi(X^{q-2})\right|} \tag{6}$$

Once the search direction is calculated, the one-dimensional search is calculated by polynomial interpolation.

It can be argued that more modern quasi-Newton methods are a better choice for solving the sub-problem. However, these methods require much more storage. Also, computational experience has shown that the Fletcher-Reeves algorithm is comparable in efficiency and reliability if carefully programmed.

During the unconstrained minimization sub-problem, almost no effort is required to calculate the search direction, so the computational time spent in the optimizer itself is negligible. The cost of optimization is almost completely the cost of analysis and gradient computations. Thus, it is desirable (but not essential) that high quality approximations are available, as is the case in modern structural optimization.

Version 1 of BIGDOT required that gradient information be provided directly and, if only a few gradients could be stored in memory, repeated returns would be made to the user to calculated the needed set. Version 2 allows the user to provide gradients in compacted form via a binary file. If convenient, more than the requested number of gradients can be provided. This has the advantage that, for strictly linear problems, gradients need only be calculated and stored once. Also, for problems with a nonlinear objective function but linear constraints, the objective function gradient is calculated as needed but the constraint gradients need to be calculated only once.

B. Discrete Variable Optimization

Traditional branch and bound methods or genetic algorithms become hopelessly inefficient when used for optimization problems of more than a few variables (say 10 or 20). For large scale optimization, no "theoretically correct" method is available for nonlinear constrained optimization problems. Therefore, an ad-hoc method is used here with the goal of finding a "reasonable" discrete solution which is feasible.

One approach to this is offered in ref. 9, where a sin or cosine shaped penalty function was created to drive the design to a nearby discrete solution. Unlike ref. 9, the sin shape was found here to be most reliable. Also, it has been modified slightly from the reference to be of the form;

$$P(X) = R\sum_{i=1}^{n}\frac{1}{2}\left\{1 = \sin 2\pi\left[\frac{X_i - 0.25(X_i^L + 3X_i^U)}{X_i^U - X_i^L}\right]\right\} \tag{7}$$

where X_i^L is the next lower discrete value and X_i^U is the next higher discrete value.

This, by its nature, creates a multitude of relative minima. Therefore, if these penalties are imposed from the beginning, there is a high probability of finding a solution that is not near the true optimum. Thus, it is desirable to first solve the continuous variable problem to provide a good starting point. Even doing this, the approach is very sensitive to penalty parameter values and may produce either a non-discrete solution or an infeasible one.

During this phase of the optimization, it is important to include the original penalty function as well to maintain feasibility with respect to the general constraints.

Equation 7 attempts to drive the variables to a discrete value. However, this penalty function creates numerous relative minima and has little assurance of insuring feasibility with respect to the original constraints or of actually driving all discrete variables to a discrete value. Therefore, after several cycles, progress is evaluated. If all discrete variables are not driven to an allowed value, we ask how to change each variable such that it will move to a discrete value with minimum effect on the objective function and at the same time maintaining feasibility with respect to the original constraints.

To achieve this, we first get the gradient of the objective function and the penalty term of the pseudo-objective function, with the penalty multipliers set to unity (get the sum of violated constraint gradients). We then seek to drive the solution to a discrete value with minimum increase in the objective function while remaining feasible with respect to the constraints.

The general algorithm for this is;

1. Including only discrete variables, and bypassing variables that are already at a discrete value, search for

$$\left| \frac{\partial P / \partial X_i}{\partial F / \partial X_i} \right| \tag{8}$$

2. Calculate the changes in X_i that will move X_i to its next larger and smaller discrete value.
3. For each such δX_i estimate the maximum constraint value based on a linear approximation.
4. Move X_i to the discrete value that maintains feasibility.
5. Repeat from step 1 until all discrete variables are at a discrete value.

This algorithm drives the design variables to a discrete value while still including the original constraints via the original SUMT algorithm. This has the limitation that variables can move only one discrete value up or down from the continuous solution during each cycle.

The final algorithm finds a feasible, discrete solution efficiently and seldom fails. However, it must be remembered that this is not guaranteed to be a theoretical optimum, only a good discrete solution.

This algorithm has the advantage that it is a straightforward addition to the continuous variable algorithm and that it requires very little memory and computational effort.

IV. Examples

Examples are presented here to demonstrate the algorithm described above.

A. Cantilevered Beam

The cantilevered beam shown in Figure 2 is designed for minimum material volume. Here, five segments are shown. In general, N segments will be considered. The design variables are the width, b_i, and height, h_i of each of the N segments. The beam is subject to stress limits, $\bar{\sigma}$, at the left end of each segment and the geometric requirement that the height of any segment does not exceed twenty times the width. There are a total of 2N design variables and 2N constraints, as well as lower bounds on the variables. Therefore, at the continuous optimum, it is expected that the design will be fully constrained (as many active constraints as design variables) because the structure is statically determinate. If the beam was allowed to be completely continuous without lower bounds on the design variables, the theoretical optimum is 53,714.

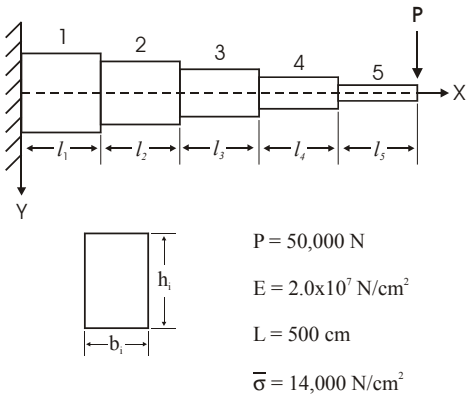

Figure 2. Cantilevered Beam

The design task is now defined as;

Minimize

$$V = \sum_{i=1}^{N} b_i h_i l_i \tag{9}$$

Subject to;

$$\frac{\sigma_i}{\bar{\sigma}} - 1 \le 0 \quad i = 1, N \tag{10}$$

$$h_i - 20 b_i \le 0 \quad i = 1, N \tag{11}$$

$$b_i \ge 1.0 \quad i = 1, N \tag{12}$$

$$h_i \ge 5.0 \quad i = 1, N \tag{13}$$

Here, N=25000, 50000, 125000 and 250000 was used to create problems of 50000, 100000, 250000 and 500000 design variables, respectively.

For the discrete solution, each variable was chosen from sets of discrete values in increments of 0.1.

Table 2. Optimization Results

	Number of Design Variables			
	50,000	100,000	250,000	500,000
Continuous Optimum	53,744 (243/46) [49,979/46]	53,720 (209/38) [99,927/150]	53,755 (262/49) [249,919/211]	53,730 (266/50) [499,700/1732]
Discrete Optimum	54,864 (92/38)	54,848 (96/25)	54,887 (143/24)	54,821 Infeasible $g_{max} = 0.011$

The results are presented in Table 2. The information in parentheses (../..) are the number of function evaluations over the number of gradient evaluations. The numbers in brackets [../..] are the number of active constraints over the number of active side constraints at the optimum, where a constraint is considered active if its value is greater than -0.05 and no more positive than 0.003. Except for the 500,000 variable problem, all constraints were satisfied at the optimum. The 500,000 variable problem failed to achieve a feasible discrete solution, even though it generated 500,000 discrete values. This problem ended with 79 violated constraints having a maximum violation of just over one percent.

In each case, the optimum achieved was about the same and the discrete solution was only slightly larger, indicating a "good" discrete solution. It is particularly noteworthy that the number of function and gradient evaluations is nearly constant. This suggests that the algorithm is very scalable regardless of problem size.

B. Car Body Reinforcement

The BIGDOT optimizer has been added to the GENESIS structural optimization program[10] to perform large scale structural optimization. To date, topology optimization problems in excess of 2,000,000 variables and member sizing problems in excess of 450,000 variables have been solved.

Figure 3 shows a car body model which we wish to reinforce to increase the bending and/or torsion frequency. The approach used here was to allow every element in the model was optimized for thickness (with a lower bound of the original design) with the constraint that only a specified fraction of the material may be used. Here, 34,560

sizing variables were used. While somewhat difficult to see in Figure 3 (unless viewed in color), reinforcement was added in the areas of the firewall, rocker panels and rear fender areas.

Table 3 gives the increase in bending or torsion frequency for different values of added mass.

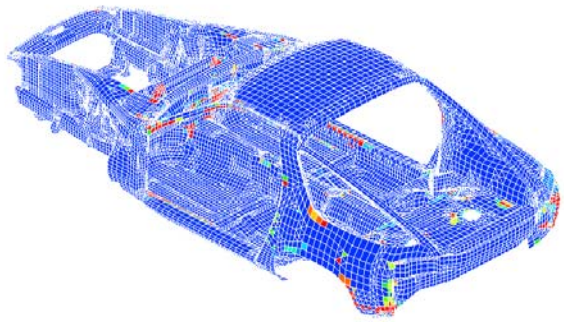

Figure 3. Car Body Reinforcement

Table 3. Frequency Increases

Added Mass (Kg)	Increased Frequency (Hz)	
	Maximize First Torsion Frequency	Maximize First Bending Frequency
2.64	4.81	6.42
7.32	7.56	9.89
15.06	9.66	112.15

C. Topology Optimization of a Support

Figure 4 is a topology optimization example where just over one million design variables were used. This structure was optimized to minimize strain energy under the applied load.

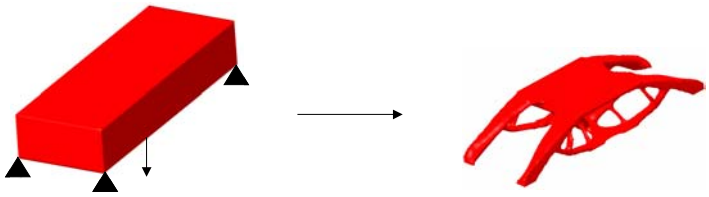

Figure 14. Skeletal Support

V. Summary

An algorithm has been developed for solving very large optimization tasks, which uses limited central memory and which avoids solution of a large sub-optimization task. The memory requirements grow only linearly with problem size. The algorithm is based on a modern exterior penalty function method which exhibits approximately constant efficiency, independent of problem size. Once a continuous optimum has been achieved, a discrete solution is found using an ad-hoc algorithm which drives the optimum to a feasible discrete solution with minimal increase in the objective function. Experience has shown that the optimization problem size is no longer limited by the optimizer, but instead by the ability of the analysis and sensitivity software to provide the needed information.

VI. References

[1] Schmit, L.A., "Structural Design by Systematic Synthesis," Proceedings, 2nd Conference on Electronic Computation, ASCE, New York, 1960, pp. 105-132.

[2]Fiacco, A. V., and G. P. McCormick: *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, John Wiley and Sons, New York, 1968.

[3]Hager, W.W., D. W. Hearn and P. M. Pardalos: "Large Scale Optimization; State of the Art," Kluwer Academic Publishers, 1994, pp. 45-67.

[4] Very Large Scale Optimization. NASA Langley Research Center Phase I SBIR Contract NAS1-99026 and continued under a Phase II Contract NAS1-00102.

[5]BIGDOT User's Manual, Version 2.0: Vanderplaats Research & Development, Inc., Colorado Springs, CO, 2004.

[6]Rockafellar, R. T.: The Multiplier Method of Hestenes and Powell Applied to Convex Programming, J. Optimization Theory Applications, vol. 12, no. 6, pp. 555 – 562, 1973.

[7]Fletcher, R. and C. M. Reeves: Function Minimization by Conjugate Gradients, Br. Computer J., vol. 7, no. 2, pp. 149-154, 1964.
[8]DOT User's Manual, Version 5.0: Vanderplaats Research & Development, Inc., Colorado Springs, CO, 1999.
[9]Shin, D. K., Gurdal, Z. and Griffin, O. H., Jr., "A Penalty Approach for Nonlinear Optimization with Discrete Design Variables," Eng. Opt., Vol. 16, pp. 29-42, 1990.
[10]GENESIS User's Manual, Version 7.5: Vanderplaats Research & Development, Inc., Colorado Springs, CO, 2004.